Engineering of IT Management Automation along Task Analysis, Loops, Function Allocation, Machine Capabilities

Dissertation

an der
Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Ralf König

Tag der Einreichung: 12. April 2010

Engineering of IT Management Automation along Task Analysis, Loops, Function Allocation, Machine Capabilities

Dissertation

an der
Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Ralf König

Tag der Einreichung: 12. April 2010
Tag des Rigorosums: 26. April 2010

1. Berichterstatter: Prof. Dr. Heinz-Gerd Hegering, Ludwig-Maximilians-Universität München
2. Berichterstatter: Prof. Dr. Bernhard Neumair, Georg-August-Universität Göttingen

Abstract

This thesis deals with the problem, that IT management automation projects are all tackled in a different manner with a different general approach and different resulting system architecture.

It is a relevant problem for at least two reasons: 1) more and more IT resources with built-in or associated IT management automation systems are built today. It is inefficient to try to solve each on their own. And 2) doing so, reuse of knowledge between IT management automation systems, as well as reuse of knowledge from other domains is severely limited. While this worked with simple stand-alone remote monitoring and remote control facilities, automation of cognitive tasks will more and more profit from existing knowledge in domains such as artificial intelligence, statistics, control theory, and automated planning. A common structure also would ease integration and coupling of such systems, delegating cognitive partial tasks, and switching between commonly defined levels of automation.

So far, this problem is only partly solved. Prior work on approaching systems design includes systems engineering with its steps task analysis and function allocation. But so far systems engineering has not yet been widely applied to IT management automation systems, in contrast to technical application domains such as automotive or aerospace engineering.

The state of the art in IT management automation itself on the technical side includes IBM's Autonomic Computing Initiative, proposing a structure of feedback loops for autonomic systems. On the organizational side, a certain standardization of IT service management processes has been reached with the introduction of process management standardization work like the IT infrastructure library (ITIL), which is a collection of best practices in IT service management, and ISO 20.000, a standard derived from ITIL V2. So far, the mentioned developments all stand each on their own.

The idea of this thesis is now to combine this prior knowledge to create a universal approach to the design of IT management automation systems in four steps:

Step 1: Task analysis in the IT management automation scenario, with an implicit goal to associate the identified tasks with a subset of the ITIL reference processes (see Chapter 3).

Step 2: Reformulation of the tasks into feedback loops of a common structure for cognitive loops (see Chapter 4).

Step 3: Function allocation of the loop steps to either performed by humans or machine components (see Chapter 5).

Step 4: Identification of a set of relevant machine capabilities in a catalog structured along the previous three steps (see Chapter 6).

Each of the steps 1–3 is the result of combining existing knowledge in systems engineering and existing knowledge in IT management automation.

During this combination the author of this thesis also creates new structures, such as an automation-oriented subset of ITIL V2 called "AutoITIL", the "MAPDEK" (monitor, analyze, plan, decide, execute based on knowledge) loop as basic loop and simpler ways of function allocation to humans/machines. The catalog of machine capabilities, presented in Step 4 then adds a non-exhaustive catalog of machine capabilities in IT management automation. It is domain-specific by design and clearly exceeds similar more general attempts such as Fitt's "Men are better at, machines are better at" list, 1951. Other

concepts are taken over unaltered after a review during this combination, such as proposals by Sheridan (2000) on the structure and description of tasks and levels of automation. All in all, this way we gain a domain-specific method for engineered IT management automation (EIMA).

The EIMA method is intended to be used in the analysis and design of IT management automation systems. Its purpose is to first align the structure of the automation problem to EIMA and then to quickly identify automation potential enabled by existing mathematical methods and machine capabilities.

In an application example (see Chapter 7), it is shown how an existing complex proposal for partial automation in a wide area network scenario is analyzed, and a new proposal is derived that improves automation along known concepts of control systems, and at the same time simplifies the organizational structure. It does so by decomposing the overall problem into a set of smaller problems that can be solved by domain experts in economics, graph theory, network configuration management. A comparison shows that while the previous proposal has created a UML-based draft model for a protocol that eases information exchange, the new proposal actually includes the cognitive tasks that had been left to system administrators in the first proposal. A variable level of automation leaves enough influence by network operators on the automation routine, and therefore provides the basis for better acceptance.

This work differs from related work in not focusing on single automation scenario instances (regarded as point solutions), but instead proposing a general method applicable to all such problems and bringing a benefit by its association with existing concepts in systems engineering and IT management. As the method is partly rooted in systems engineering, we can build on the experience that has been gathered there in the design of control systems like autopilots and driver assistance systems.

The main benefit of EIMA for system designers is a uniform way of describing and decomposing IT management automation problems even of high complexity at a high level of abstraction. Only after this high-level decomposition, system designers and domain specialists will take care of the details by further refining the tasks and implementing system components. This top-down approach makes even complex IT management automation systems comprehensible and concise in their description. Also for system administrators this is a clear benefit.

To achieve this simplicity, EIMA also must leave certain aspects unconsidered. Due to its universality, we can hardly include scenario-specific aspects: EIMA makes no advice on the "best" automation. Instead, it leaves out scenario-specific cost benefit analysis or return-on-investment analysis as this depends on external policies as well. EIMA also does not give detailed cookbook-like advice on which machine capabilities to use for what task. In this point it must rely on the experience, background knowledge and creativity of a user of the EIMA method, all of which is supported by EIMA, but not guided along a single path of cause effect relationships to one particular systems design.

Potential follow-up work includes the creation of tools for EIMA itself: modeling tools, a knowledge base of machine capabilities, the electronic representation of partial solutions such as loop patterns, and simple visualization and simulation. In the same way as the design of electronic circuits today is supported with circuit simulation programs such as SPICE, or design of 3D objects such as machines or buildings with CAD applications such as AutoCAD. Once such tools were available, EIMA models could be exchanged in terms of formal, machine-interpretable files instead of mainly verbal system descriptions interpreted by humans.

Zusammenfassung

Diese Dissertation beschäftigt sich mit dem Problem, dass Projekte zur IT-Management-Automatisierung jedes für sich anders angegangen werden und daraus unterschiedliche Systemarchitekturen entstehen.

Die entstehende Heterogenität ist aus mindestens zwei Gründen problematisch: 1) Immer mehr IT-Systeme mit eingebauten oder gekoppelten IT-Management-Automatisierungsfunktionen werden heute entworfen. 2) Dabei wird nur in sehr begrenztem Maße auf vorhandenes Wissen aus anderen IT-Management-Automatisierungsystemen oder verwandten Wissensfeldern zurückgegriffen. Während der bestehende Ansatz für einfache Überwachungssysteme und Fernsteuersysteme noch ausreichte, würde die Automatisierung der kognitiven Aufgaben immer mehr von Methoden aus den Gebieten Künstliche Intelligenz, Statistik, Steuer- und Regelungstechnik, Automatisiertes Planen profitieren. Eine einheitliche Struktur von IT-Management-Automatisierungssystemen würde auch die Integration und Kopplung solcher Systeme vereinfachen, sowie die Delegation von kognitiven Teilaufgaben und das Umschalten zwischen einheitlich definierten Automatisierungsgraden.

Bisher ist dieses Problem nur teilweise gelöst. Ein umfassender Ansatz zu Systemanalyse und Systementwurf wird im Systems Engineering beschrieben mit den Teilschritten Tätigkeitsanalyse (task analysis) und Funktionszuordnung (function allocation). Im Gegensatz zu Anwendungsdomänen wie der Automobiltechnik und der Luft- und Raumfahrttechnik wird ein vergleichbarer Ansatz bisher aber kaum auf IT-Management-Automatisierungssysteme angewendet. Der aktuelle Wissensstand in der IT-Management-Automati-sierung an sich beinhaltet auf der technischen Seite unter anderem IBM's Autonomic Computing Initiative, die für "autonomic systems" eine Struktur aus Rückkopplungsschleifen vorschlägt. Auf der organisatorischen Seite hat die Arbeit in der Standardisierung des IT-Prozess-Management, unter anderem durch ITIL und ISO 20.000, Verbreitung gefunden. Bisher jedoch stehen diese Entwicklungen jeweils für sich.

Die Idee der hier vorgelegten Arbeit ist, dieses vorhandene Wissen zu kombinieren. Damit wird ein universeller Ansatz geschaffen, mit dem man IT-Management-Automatisierungssysteme in vier Schritten entwerfen kann:

Schritt 1: Tätigkeitsanalyse (task analysis) des IT-Management-Automatisierungs-Szenarios mit dem impliziten Ziel, die gefundenen Aufgaben einem Prozess aus einer Untermenge der ITIL-Prozesse zuzuordnen (siehe Kapitel 3).

Schritt 2: Umstrukturierung der Aufgaben in ein Schema aus Rückkopplungsschleifen mit einheitlicher Struktur für kognitive Schleifen (siehe Kapitel 4).

Schritt 3: Funktionszuordnung (function allocation) der Schritte in den Schleifen zu Rollen, die entweder von Menschen oder Maschinenkomponenten ausgefüllt werden (siehe Kapitel 5).

Schritt 4: Identifikation einer Menge von maschinellen Fähigkeiten in einem Katalog, der anhand der Schritte 1–3 gegliedert ist (siehe Kapitel 6).

Jeder der Schritte 1–3 ist dabei das Ergebnis einer Verschmelzung von vorhandenem Wissen im Systems Engineering mit vorhandenem Wissen aus dem IT-Management.

Bei dieser Verschmelzung entstehen durch die Arbeit des Autors auch neue Strukturen, wie unter anderem eine automatisierungsorientierte Untermenge von ITIL V2 namens "AutoITIL", die "MAPDEK-

Schleife" (überwachen, analysieren, planen, entscheiden, ausführen, basierend auf Wissen) als Basisbaustein und einfachere Arten der Funktionszuordnung zu Mensch/Maschine. Der in Schritt 4 vorgeschlagene Katalog geht deutlich in Umfang und Struktur über ähnliche Ansätze (z.B. Fitt's Liste "Men are better at, machines are better at", 1951) hinaus und ist bewusst domänenspezifisch für IT-Management-Automatisierung angelegt. Andere Konzepte werden bei der Verschmelzung nach Prüfung unverändert übernommen, wie z.B. Vorschläge von Sheridan (2000) zur Strukturierung und Beschreibung von Aufgaben (tasks) oder Automatisierungsgraden (levels of automation). Insgesamt erhalten wir eine domänenspezifische Methode namens "EIMA" (Engineered IT Management Automation) für IT-Management-Automatisierung nach ingenieurwissenschaftlichen Grundsätzen.

Die beschriebene Methode EIMA soll bei der Analyse und dem Entwurf von IT-Management-Automatisierungssystemen eingesetzt werden. Ihr Zweck ist dabei, vorhandene Szenarien an der neuen Struktur auszurichten und dann vorhandenes Automatisierungspotenzial schnell zu erkennen, das durch vorhandene mathematische Methoden oder Maschinenfähigkeiten genutzt werden kann.

In einem Anwendungsbeispiel (siehe Kapitel 7) wird exemplarisch gezeigt, wie ein vorhandener komplexer Vorschlag für die Teil-Automatisierung in einem Weitverkehrsnetz-Szenario erst analysiert und dann ein neuer Vorschlag abgeleitet wird, der entlang bekannter Prinzipien die Automatisierung erhöht und die organisatorische Struktur vereinfacht. Dies wird erreicht, indem man das Gesamtproblem in kleinere Probleme zerlegt, die dann von Domänenexperten in Betriebswirtschaft, Graphentheorie und dem Konfigurationsmanagement von WAN-Komponenten jeweils einfach gelöst werden können. Ein Vergleich der beiden Vorschläge vor/nach EIMA zeigt, dass der erste Vorschlag zwar den Informationsaustausch mit einem in UML modellierten Protokollentwurf automatisiert hat, aber der EIMA-basierte Vorschlag die Automatisierung der kognitiven Aufgaben einschließt, die im ersten Fall den beteiligten Menschen überlassen blieb. Um die gewünschten Einflussmöglichkeiten durch die Partner des WAN-Verbundes zu behalten, wird ein variabler Automatisierungsgrad vorgesehen.

Diese Arbeit unterscheidet sich von verwandten Arbeiten, indem sie nicht auf ein bestimmtes Punkt-Szenario abzielt, das zu automatisieren ist. Stattdessen wird hier eine allgemeine Methode angegeben, die sich für viele IT-Management-Automatisierungsszenarien eignet, mit dem Vorteil existierende Konzepte aus Systems Engineering und IT Management in sich zu vereinigen. So kann auch auf Erfahrungen aufgebaut werden, die im Zusammenhang mit Steuerungssystemen wie Autopiloten oder Fahrerassistenzsystemen gesammelt wurden.

Der Hauptvorteil von EIMA für Systemdesigner ist eine eine einheitliche Art der Beschreibung und Zerlegung von selbst komplexen IT-Management-Automatisierungsproblemen auf einer hohen Abstraktionsebene. Erst nachdem man diese Zerlegung gemacht hat, würden Systemdesigner wie Fachleute die Detailfragestellungen angehen, in dem die Aufgaben weiter verfeinert werden und Systemmodule implementiert. Dieser Ansatz der hierarchischen Dekomposition von oben nach unten ermöglicht es, selbst komplexe IT-Management-Automatisierungssystems verständlich und knapp zu beschreiben. Auch für Systemadministratoren ist dies ein deutlicher Vorteil.

Um diese Vereinfachung zu erreichen, muss EIMA aber auch bestimmte Aspekte unberücksichtigt lassen. So können wegen der Universalität bestimmte szenariospezifischen Fragen kaum berücksichtigt werden: EIMA gibt damit also keinen Rat zur "bestmöglichen" Automatisierung. Da diese Bewertung auch von den äußeren Rahmenbedingungen abhängt, bleiben eine Kosten-Nutzen-Analyse oder eine Amortisationsrechnung außen vor. EIMA macht auch keine punktgenauen Vorgaben, welche Maschinenfähigkeiten für welche Aufgaben herzunehmen sind. In diesem Punkt muss es auf die Erfahrung, das Hintergrundwissen, sowie die Kreativität des Systemdesigners vertrauen, die zwar von EIMA mit Vorschlägen unterstützt werden, aber wo "die Lösung" nicht entlang eines einzig möglichen Pfades aus Ursache-Wirkung-Beziehungen hergeleitet wird.

Mögliche Folgearbeiten könnten insbesondere die Anwendung von EIMA unterstützen, indem Werkzeuge an EIMA angepasst oder für EIMA konzipiert und entwickelt werden, z.B. Modellierungswerkzeuge, eine elektronische Wissensdatenbank als Implementierung des Kataloges der Maschinenfähigkeiten, die elektronische Repräsentation von Teillösungen wie z.B. bestimmten Schleifenmustern, sowie Funktionen zur Visualisierung und Simulation. In vergleichbarer Weise dazu, wie heute der Schaltkreisentwurf durch Schaltungssimulationsprogramme wie SPICE unterstützt wird, oder der Entwurf von 3D-Objekten wie Maschinen oder Gebäuden durch CAD-Anwendungen wie AutoCAD. Wenn solche Tools einmal verfügbar sind, könnten EIMA-Modelle als formale, maschinen-interpretierbare Dateien unter Entwicklern ausgetauscht und wiederverwendet werden. Bisher beschränken sich Systembeschreibungen meist auf eine weitgehend verbale Beschreibung, die von Menschen interpretiert wird.

Contents

1 Introduction

Contents

To make the line of argument of this thesis better comprehensible to readers, this chapter opens with a simple model of IT management, that shows how organizational and technical aspects work together to manage changing IT environments. Along these lines, problems in current IT management automation are identified, of which one was chosen as the topic of this work: the anticipation of future management operations for a certain resource at design time, and the improvement of automation of resource-related IT management procedures by adding self-management capabilities closely associated with the managed resource. This shall significantly reduce ad hoc IT management automation happening in IT data centers today with all its consequences.

After giving an overview on related work regarding the chosen problem, the idea of a method named "Engineered IT management automation" (EIMA) is presented, that aims to take over knowledge from systems engineering and industrial process control and automation to IT management automation. This method's structure of task analysis, loop modeling, function allocation and task implementation also determines the structure of this work. The steps of this method are later elaborated in a separate chapter each, followed by a proof of concept along the example of life-cycle management automation for an IT solution.

This introduction closes with a vision statement of a development system that allows to model IT environments from components and to simulate the interaction of human IT managers and machine management subsystems, similar to existing systems that combine construction and simulation in mechanical/electrical engineering but with the new property to incorporate, model, simulate and improve the associated systems management processes at design time as well.

1.1 A simple model for IT management

Layers in IT management The following definition from [HAN99] describes well the general meaning of IT management:

> The management of networked IT systems in its broadest sense includes all tasks, that ensure an effective and efficient operation of systems and their resources, aligned to enterprise objectives. It serves to provide services and applications of the networked systems at the desired quality and to ensure their availability.

Starting from this definition and adding business processes and cross-enterprise relationships, this definition can be transformed into Figure 1.1.

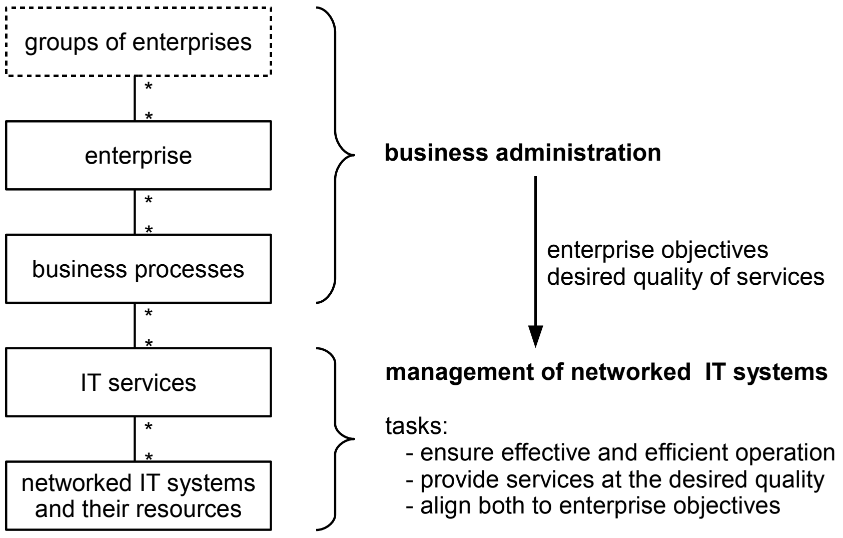

groups of enterprises
enterprise
business processes
IT services
networked IT systems and their resources
*
business administration
enterprise objectives
desired quality of services
management of networked IT systems
tasks:
- ensure effective and efficient operation
- provide services at the desired quality
- align both to enterprise objectives

Figure 1.1: IT management and business administration, a layer model

The layers of resources and services are typically associated with IT management, while the layers on top are generally associated with business administration. This simple model of hierarchical management assumes that the managed entities depend on each other in n:m relationships.

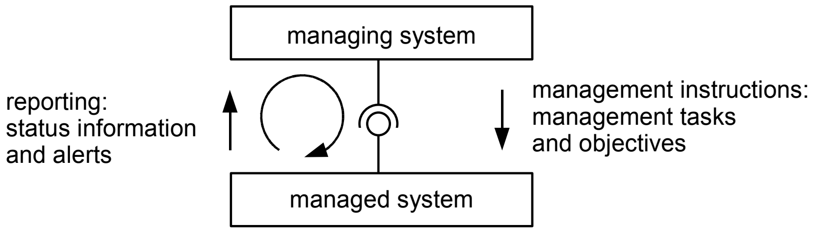


Figure 1.2: Simple model for a basic management interaction

Basic management interaction A basic management interaction is shown in Figure 1.2. In this figure, both the role of a manager system as well as of a managed system can be associated with a human being or a technical system, forming human-computer systems. Managing system and managed system interact via a management interface over a management protocol, with the managing system sending management instructions, while the managed system returns mainly status information or in some cases sends alerts. This interaction typically forms a loop where in hierarchical management the managing system supervises the managed system. Management tasks can be organizational or technical, depending on the layer of abstraction. In distributed systems, managing system and managed system typically interact with each other via a communication system. This communication system is neglected here for reasons of brevity.

Cascaded management interaction The basic management interaction shown in Figure 1.2 can be cascaded to form multiple hierarchical management layers. A common combination of a technical managed system, a technical management system and a human manager system is shown in Figure 1.3.

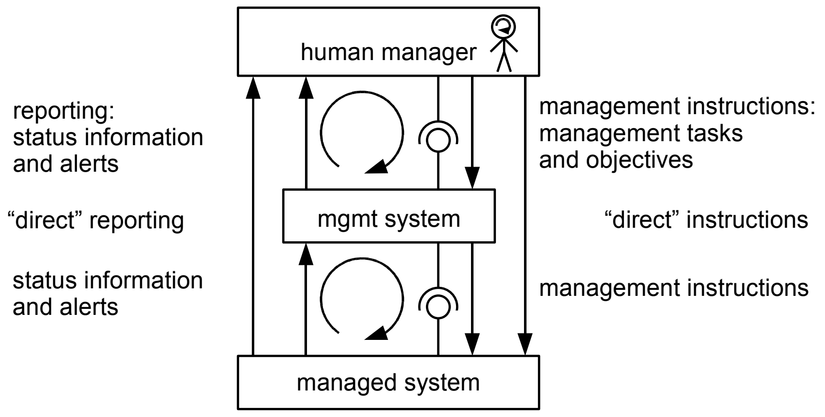


Figure 1.3: Common management hierarchy: A management system supports a human manager

By delegating tasks to the management system, the management system supports or assists the human manager in certain tasks. Special attention should be paid to the "side-channels" of management, that can occur in management hierarchies: In certain circumstances the in-between management system is bypassed and the human manager directly interacts with the managed system.

Composed model All of the statements on management systems up to here apply for human to human management, human to machine management as well as machine to machine management. Therefore the model of a cascaded management system can be combined with the layer hierarchy of dependent managed entities to form Figure 1.4.

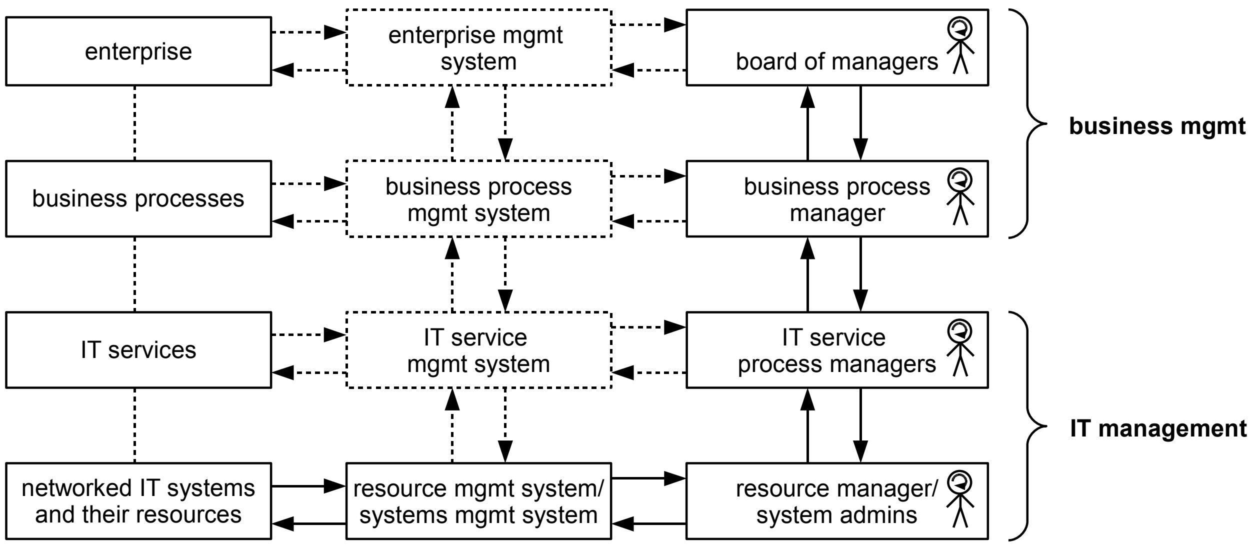

enterprise
enterprise mgmt system
board of managers
business processes
business process mgmt system
business process manager
business mgmt
IT services
IT service mgmt system
IT service process managers
networked IT systems and their resources
resource mgmt system/ systems mgmt system
resource manager/ system admins
IT management

Figure 1.4: Layers of managed entities combined with cascaded management loops. This figure is an extended version of a graphics in [Bre07]. The dashed boxes in this figure are not universally established in most enterprises.

1.2 Problems in current IT management automation

In this section, three problems are introduced: 1) the problem of rising complexity in IT management, 2) the problem of automation glue code written in an ad hoc manner, and 3) insufficient communication between systems management and systems development.

This thesis provides a partial solution to all three of them: 1) by proposing a method along which system administrators and system developers can map their respective knowledge to the other party's level of abstraction, 2) by moving automation from the individual customer owning a set of resources to the resource vendor, and 3) reducing heterogeneity and scale by vendors offering resource bundles of managed systems and management systems, preconfigured for a certain application scenario.

1.2.1 Problem 1: IT management complexity as opponent of efficiency and effectivity

As it can be seen in the definition of IT management, ensuring effective and efficient operation is a major objective in IT management. This is partly contradictory to the other two objectives: providing services at the desired quality and to align IT management with enterprise objectives.

Both, a high desired quality of services as well as the alignment with (changes in) enterprise objectives increase management complexity. To these business-related causes, increasing scale and heterogeneity add technology-related causes of increasing IT management complexity. In turn, this increasing complexity of IT management is a major opponent of efficiency and effectivity mainly because most of this complexity is immediately perceived by people working in IT management.

In an intuitive approach to qualify complexity of IT management, the following statements reflect the essence of the author's perception on the causes of the rising complexity in IT management, that are illustrated in Figure 1.5.

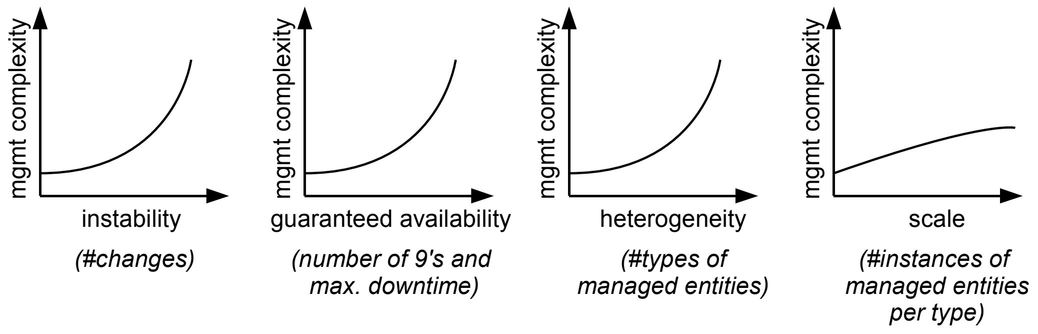


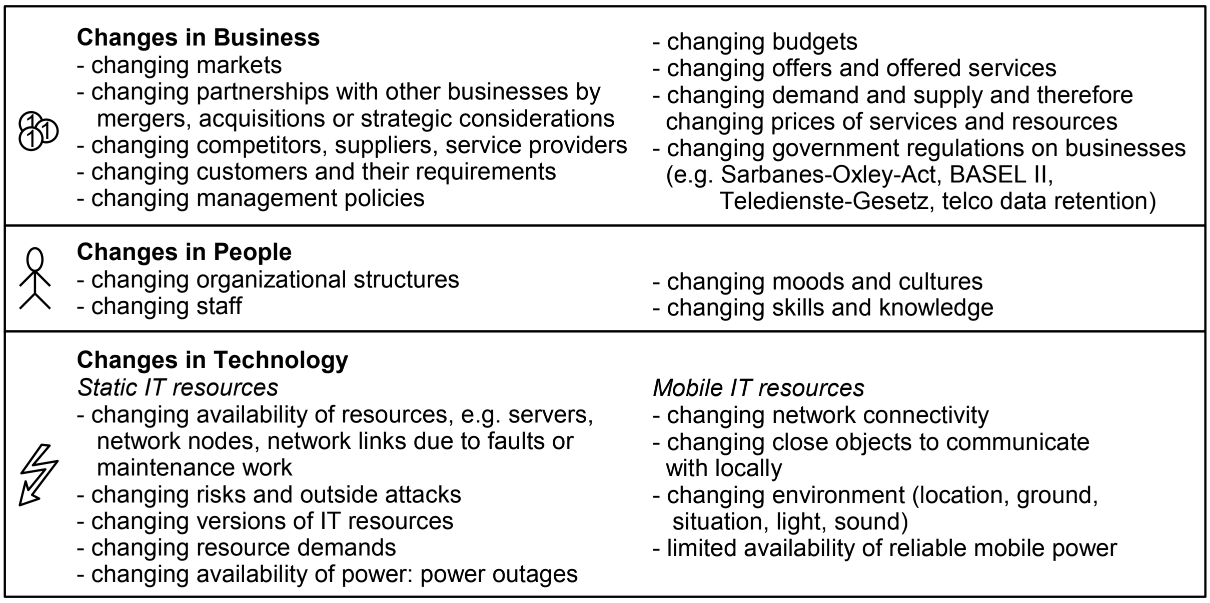
Figure 1.5: Influence factors on (IT) management complexity

1. IT management complexity grows considerably, when the *number of changes* grows.

   Several kinds of influences and changes can be identified in business, people and technology around IT management (see Figure 1.6). They all are potential reasons for change. Along this line of thought, IT management serves to align changes in information technology and changes in the business.

2. IT management complexity grows considerably, when a business more and more depends on IT operations and therefore the required level of *guaranteed quality of services* increases.

   The more an enterprise depends on its IT for its daily business, the more will IT management have to incorporate availability in all kinds of cases, reliable backup and secure fail-over pro-



Changes in Business - changing markets - changing partnerships with other businesses by mergers, acquisitions or strategic considerations - changing competitors, suppliers, service providers - changing customers and their requirements - changing management policies	- changing budgets - changing offers and offered services - changing demand and supply and therefore changing prices of services and resources - changing government regulations on businesses (e.g. Sarbanes-Oxley-Act, BASEL II, Teledienste-Gesetz, telco data retention)
Changes in People - changing organizational structures - changing staff	- changing moods and cultures - changing skills and knowledge
Changes in Technology *Static IT resources* - changing availability of resources, e.g. servers, network nodes, network links due to faults or maintenance work - changing risks and outside attacks - changing versions of IT resources - changing resource demands - changing availability of power: power outages	*Mobile IT resources* - changing network connectivity - changing close objects to communicate with locally - changing environment (location, ground, situation, light, sound) - limited availability of reliable mobile power

Figure 1.6: Changes influencing IT management

cedures, risk analysis. In addition, we are facing the conflict of increasing IT dependency demanding improved quality of IT services, while increasing awareness of IT controlling demands reduced expenditures. IT management will have to live with this fact and needs to prepare for even stronger requirements in the future.

3. IT management complexity grows considerably, when the *heterogeneity of managed entities* grows.

   The more heterogeneous an IT landscape becomes, the more different aspects, procedures, commands an operator has to remember. When considering binary or higher relations between heterogeneous resources (e.g. on compatibility) this number grows potentially over the number of types of managed entities.

4. IT management complexity grows slightly, when the *scale/number of managed entities* grows.

   The more managed objects there are, the more time it will take to manage them all, as the number of potential changes and faults increases.

1.2.2 Problem 2: Ad hoc management and ad hoc automation

Ad hoc management A certain culture of ad hoc management has developed and still can be witnessed in IT management, being a young discipline, and less "scientifically mature" than most engineering domains with less standardized education/training of staff. For a long time, proactive prediction of most likely next events did not take place, instead system administrators reactively took care of upcoming incidents and problems. There hardly existed established, documented processes in resource and service management and responsibilities were rarely assigned to roles, especially for cases of emergency.

In addition, ad hoc direct interaction with managed entities that are also managed by a technical management system (see Figure 1.3) or other human managers has been a major source of inconsistencies (multiple manager problem).

Ad hoc automation Unfortunately, administrators have been getting little support from management systems so far. Currently, their "toolbox" consists of a) general purpose integrated management systems, b) resource-specific management tools and c) scripting facilities typically with a set of command-line tools.

General purpose IT management systems by third parties are available, which promise a certain level of integration when managing heterogeneous IT infrastructures. These tools are typically designed in an open-loop fashion with a main focus on monitoring and visualizing the state of certain parts of an IT infrastructure, as well as offering basic control elements to send management instructions to the managed systems. In between, human operators analyze the monitored data, plan changes, and decide about their execution. In effect, the overall level of achievable management automation of these tools is generally low. In addition, general purpose management systems have limited abilities in allowing system administrators to manage all aspects of the heterogeneous infrastructure.

Resource-specific management tools typically well cover the set of manageable properties of a managed system. However, with the administrators having to learn about all those tools, they are rather applicable to special tasks instead of every-day use.

Resource-related IT management tasks like fixing faults in managed entities or performance tuning are usually carried out by a trial-and-error based progress of system administrators learning about the managed entities and then—still with a limited knowledge about the internal working of the managed resources—implementing an ad hoc "works, at least well enough, at least for this case, at least for me" style automation routine, usually in form of a script. These point solutions lack standardization, documentation, testing, in general terms: excellence in software engineering. Typically, scripts are written in interpreted languages in a text editor with very limited checking at development time by development tools.

This current practice can be compared to the goal of effective and efficient IT management demanded at the start of this chapter:

- Regarding effectivity, modeling the management task is done ad hoc as well, simulation hardly takes place, let alone verification. Testing is reduced to the minimum with a few typical test cases, but not exhaustively. As a result of such ad hoc automation, critical management actions being triggered automatically with a certain periodicity without any safety precautions impose a high risk to fail or even to influence parts of the infrastructure in an unexpected manner.
- Regarding efficiency, even though the set of managed resources is often similar to many other IT departments in the world, many custom scripts are written. This way of developing automation glue code lacks excellence in system development as well as leveraging an economy of scale. Instead, point solutions are created which can a) considerably increase the heterogeneity of overall IT management and b) cause a negative attitude towards automation in general as the effects of ad hoc automation can be much worse than the effects of not automating a task at all.

1.2.3 Problem 3: Insufficient communication between systems management and systems design/development

So far, system development and system management are not considered as parts of a common life-cycle of a managed resource, but as a matter of fact, all developed resources will be used in operations and all resources in operations must have been developed. Figure 1.7 shows the PLAN-BUILD-RUN cycle, that links system development and system management.

In most cases, there is a split view between systems development and systems management, in other

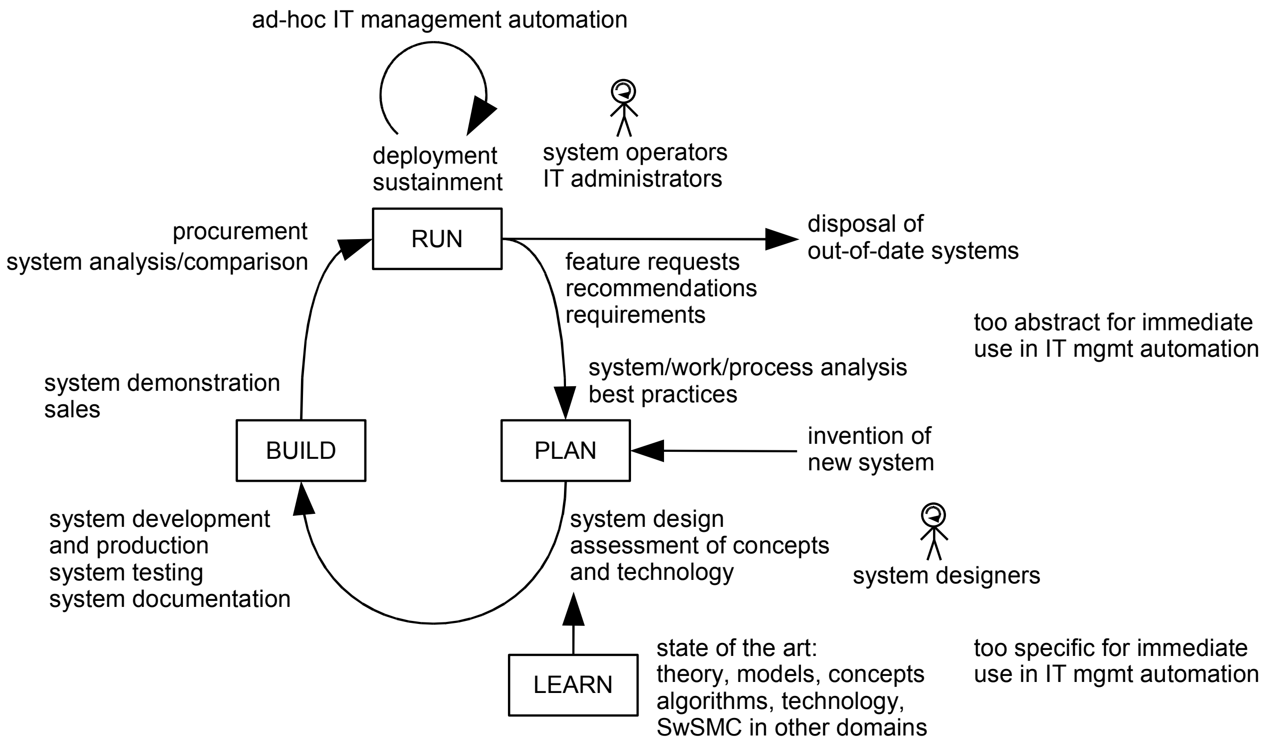


Figure 1.7: The IT system life-cycle shows the connection of systems management and systems design. System operators and systems designers each have a different focus on systems. To successfully replace ad hoc automation in IT management we need more and better communication between designers and operators. This work presents a mapping between these views along principles from systems engineering.

words system developers rarely run and manage the developed systems in every-day life and system operators typically do not develop the resources they run. The split view can be described in the following manner: in the same ways as system developers of managed resources have not well enough predicted the necessary management tasks and its associated effort and therefore cost, system operators have rarely given feedback to developers about the necessary management work. In general, system operators have hardly been involved into the redesign process. They were not asked to, and for various reasons they did not insist to be involved. Systems designers and operators hardly get in touch with each other. The overall interaction between these groups is mostly unclear, mainly because of their different focus.

The reason for this split focus lays in the domain-centric view that many researchers, education institutes, and publishers apply. Design, build and test processes are typically regarded to be associated with systems/software development. Theory, models and formal methods are typically regarded to be domain-specific, e.g. with engineering domains, or mainly independent from applications such as artificial intelligence.

Today, requirements are regarded the typical information either given to designers by a particular customer, or as general requirements sensed by market research for commodity products. Requirements alone in turn do not solve the problem for the following reasons: 1) operators are rarely involved into the requirements specification for a new system, and 2) customers hardly know about the consequences of their requirements. 3) Systems designers can hardly anticipate, whether the set of requirements is up-to-date and complete, as well as the potential compromises a customer would make if some of the stated requirements were changed or left out.

1.3 Potential solutions to these problems

Enterprises can choose to tackle the problems mentioned before with improvements in a) people, b) processes and/or c) technology. Improvements in people, processes and technology each address different contributors to complexity mentioned before.

Improvements in staff/people

Today's IT resource and service infrastructures have grown beyond a state where human administrators can understand, memorize, and communicate all the details and management-related properties of the large number of heterogeneous resources and services in an IT landscape.

Upper management can improve operators' education and knowledge of the managed systems by hiring qualified staff and specialists, finally getting better workers. Training can be established, improved and checked on a regular basis, to make sure, that operators are knowledgeable and have enough practical expertise. Self-management of operators can be improved, e.g. with courses on time management, knowledge management and team-work.

From the current point of view, people are at their limits related to complexity and time efforts due to their necessary involvement into repetitive tasks. While training will lead to more effective and efficient work by humans, expected improvements are probably small. In addition, more investment into people (e.g. better management staff, time-planning (self-management of administrators), task planning, education and certification of staff, hiring better educated or qualified staff) can be associated with both high sustaining costs and unknown overall improvement on effectiveness and efficiency. In contrast to improvements in processes or technology, achievements in better staff can hardly be copied/multiplied. While in-house education and collaboration among staff may work well, knowledge transfer between people is a slow process.

All in all, improvement in people will be necessary, but improvement in people alone does not appear to be sufficient to address all four factors that contribute to complexity and the problems stated above. Improvement in staff will not be a central topic of this work.

Improvements in processes

Only with increasing awareness of the criticality of IT infrastructures and proper IT management for a whole enterprise, attempts of process orientation have been made. There is much potential in defining, streamlining and continuously improving processes, defining performance metrics, measuring results, all of which had classically been a task of IT management controlling. Done the right way, the increased level of bureaucracy will have a positive impact on business agility and flexibility, not a negative one.

In an effort to structure resource management tasks, the OSI management architecture defined a management functions model which lists five systems management functional areas, commonly known as FCAPS. Process management frameworks such as ITIL and eTOM fight ad hoc management by defining IT service management processes that combine human actors (people) with tools for process and service management to manage the IT services of a service provider, which in turn are based on resources that need to be managed as well. Finally, CobiT proposes standard actions in IT controlling. See Table 1.1 for a rough comparison of ad hoc management, FCAPS and ITIL V2.

Table 1.1: Alignment of tasks in Ad hoc management, OSI FCAPS and ITIL V2

ad hoc management	OSI managent SMFA's	ITIL V2
ad hoc fault analysis and correction	Fault Mgmt	Incident Mgmt, Problem Mgmt, IT Service Continuity Mgmt
ad hoc configuration	Configuration Mgmt	Configuration Mgmt, Change Mgmt
flat pricing	Accounting Mgmt	Service Level Mgmt
best effort performance	Performance Mgmt	Service Level Mgmt, Capacity Mgmt, Availability Mgmt
ad hoc reaction to security incidents	Security Mgmt	Security Mgmt

In addition to these frameworks, a shift in resource management from resources being managed by local administrators to a combination of local management by humans and machines, and remote management by humans and machines, can be seen as well (outsourcing). This development can reduce the complexity perceived by the operator of the IT resources.

All in all, the introduction of standard service management processes is necessary to structure tasks and organizational matters and to take care of many changes that can hardly be automated, but where the corresponding workflow can be standardized to a certain extent. IT service management processes can have a positive impact in reacting to the rising frequency of changes and increased IT dependency. They can also provide a better interface to business process management in an enterprise. However, the processes will hardly address heterogeneity and scale of managed entities. The overall effect on efficiency or effectivity therefore is unknown since added bureaucracy and a whole management layer of people and management systems can increase overall costs. Improvement in processes will not be a central topic of this work, as well.

Improvements in management systems (technology)

Improvements in people and processes address mainly business-related types of change but do hardly address resource-related changes, as well as heterogeneity and scale of resources. In contrast, standardization and automation in resources does not directly cover business-related aspects but deals well with change in technology, as well as heterogeneity and scale.

Automation shifts IT management tasks Looking back at Figure 1.4, it can be said, that resource management systems are relatively established, but many enterprises have only started to install and use management systems for the enterprise, business processes and IT services. While enterprise resource planning software is widely established, there are hardly links to IT management.

When estimating intensity of the communication relationships in this figure, most of the coordination and cooperation between business and IT management is done via **human to human** management and communication (right-most column in the figure). Done initially via voice, phone, mail, and e-mail, recently collaboration tools like workflow tools and groupware speed up and add structure to human to human communication.

There is also much communication from **machines to human** managers, mainly as result of monitoring systems. In fact, administrators' mailboxes are often so full of mails sent automatically by machines, that it is hard to keep track which of them needs what kind of reaction. With the cautious attitude to "never touch a running system" actual management instructions in form of adjustments from **humans to machines** are relatively rare.

Only a small part of the overall communication and coordination among managed entities is done in an automated **machine to machine** communication without human intervention.

Projected into Figure 1.4, automation in IT management would lead to two major shifts in the distribution of tasks:

1. In each layer, a left to right shift: Many tasks will shift from management interaction between a human manager and a managed entity or a human manager and a management system to management interaction between a management system and the managed entity with little intervention of human managers. (see Figure 1.8)
2. Across layers, many tasks will shift from the top to the bottom layer. In the end, the majority of tasks in IT management incorporate the configuration of the actual resources. So, everything that can be done at the resource management layer should be done there with minimal intervention of the other layers.

When these two directions are combined, they show a trend towards better resources and resource management systems as the main contributors in IT management automation. However, automation in IT management will need to be prepared by a) identifying suitable tasks for automation and b) reducing heterogeneity to increase the number of resources of the same type.

The central topic of this work is therefore **automation** in IT management, but shifted to the design phase of managed entities forming systems with self-management capabilities.

Enabling automation by reducing heterogeneity To improve the applicability of automation, relatively uniform environments are favorable, which can be achieved for example by standardization efforts and procurement policies. System vendors have the task to reduce heterogeneity with modeling and standardization, both of which being a key issue in coping with complexity. Once modeling, abstraction and standardization tackle the issue of heterogeneity, i.e. reduce the number of types of managed entities, automation can tackle the growing scale of management entities and management interactions, e.g. with scalability as major requirement in systems design.

Progressive transition to systems with management atuomation routines Operators can be given assistant systems (management systems), that they can delegate tasks or parts of tasks to, in order to free them from repetitive tasks and to increase accessibility of information. Self-management capabilities or remote-management can be programmed into devices and software, so operators no longer have to deal with the problems as the systems are programmed to take care of problems based on policies.

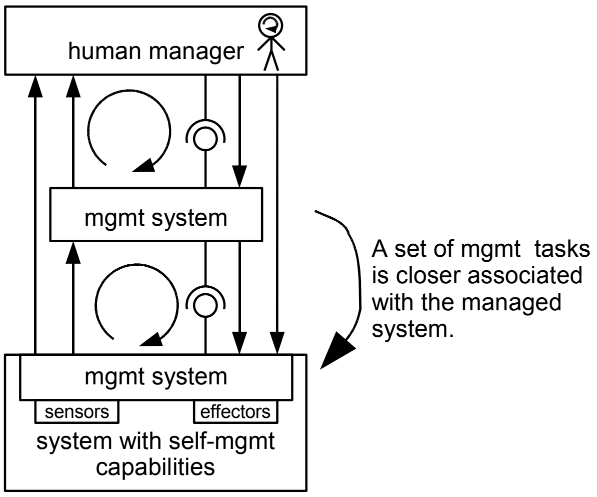

human manager
mgmt system
A set of mgmt tasks is closer associated with the managed system.
mgmt system
sensors
effectors
system with self-mgmt capabilities

Figure 1.8: In systems with self-management capabilities certain tasks move from an external mgmt system to a mgmt system tied with the resource

To put the delegation of tasks from human managers to management systems into practice, the following improvements in management systems are necessary:

1. Increasing the depth of automation, i.o.w. the level of autonomy of the management system.

   Today, management systems are widely used mainly for monitoring of managed entities and execution of management commands. Analysis, planning and decisions are mostly associated with human managers, future mgmt systems will need to support managers here and – depending on the circumstances – take over analysis, planning and decision making, thus forming a closed feedback loop.

2. Improved management systems must give human managers less reason for direct interaction with the managed entities, bypassing the management system. They should handle direct management correctly though, as well as concurrent interaction with multiple human managers.

3. At the same time increase the level of abstraction of management interactions as well as reducing the frequency of necessary management interactions to achieve a certain goal. To approach this goal, the management system is likely to profit from internal domain knowledge as well as sensors/interfaces for context information.

This list of features becomes realistic when tying all the resource-related management tasks more closely with the resource. This way, a part of former tasks of the management system moves into the managed system, forming a **management automation systems**, see Figure 1.8.

While administrators already use a wide spectrum of management tools, the focus has always been on raising the level of abstraction. In other words: future IT management will be more diverse and spread all over the IT landscape from devices with self-management capabilities to management systems with various functions at the different levels.

1.4 Problem statement

For upcoming systems with self-management capabilities among other issues two major remaining problems can be identified: a) to be designed efficiently and effectively by using available knowledge by theoreticians and systems designers and b) to be accepted and trusted by system operators. Both of these problems can be traced to insufficient communication between operators and designers, as well as lack of structure and content of documentation, once at design time, once at operation time. Ideally, system developers and system operators together would predict frequent management tasks at design time and associate automated management routines with the managed system. This way, developers would learn about management tasks and operators would learn about the implementation.

Instead, system managers and system developers have a different focus in current practice and as a result there is limited communication between these groups. Current knowledge with theoreticians includes methods from artificial intelligence, graph and control theory in text books, knowledge of software engineers like design patterns, both of which system operators cannot align to their daily work, to apply them to automate their routine tasks. The other way around, theoreticians and software engineers find it difficult to understand the current practice in IT management to find the right spots where existing or new algorithms may fit. We would therefore profit from additional layers of abstraction that associate implementation methods (designer view) with management tasks (operator view).

The problem can be stated like this:

> **How can IT management tasks be mapped to an implementation where human operators are supported by machine managers which are closely associated with the managed resources?** (direction from IT management tasks to implementation in terms of systems with self-management capabilities)
>
> A kind of reverse question is: **Once new algorithms and methods come up or are used successfully in other domains, how can they be leveraged in the automation of IT management?** (direction from implementation to applicable use in IT management automation)

To address this problem, we need a method, which is generic enough to cover many use cases, but specific enough to give actual value in a certain application scenario.

1.5 Current related work

A review of current related work shows several point solutions in the automation of IT management, e.g. automated updates in operating systems, drivers and applications, basic fault reporting, adaptivity in parameters to self-tune performance, fail-over configurations in virtual environments, but they were introduced on a case-by-case basis without any visible method. They are accompanied by protocols that support operators in certain network management tasks: routing and spanning tree protocols to route around failed network components, DHCP for dynamic address configuration, and "plug and play" technologies taking care of low-level tasks, like hardware compatibility and configuration.

In addition to these technical developments, there is a trend towards standardization of organizational aspects like IT management processes, with ISO 20000, ITIL and eTOM being prominent examples. In compliance with these standards, there are efforts to certify staff and organizations independent from a specific vendor or platform. With respect to the problem statement, with ITIL V3 there is a certain trend to link service management and service development, but the implementation in practice and resources remains unclear. eTOM with the NGOSS specification in turn applies well to telecom industry but does not cover most other IT management tasks.

System engineering tells us, that fixing flaws in a system in the operational phase is e.g. a thousand times more costly than fixing/taking care of things in the design phase. In a back-of-an-envelope style estimation, a 10% improvement in management will therefore lead to a reduction from 1000 to 900 units of effort. Instead when we spend the same effort of 100 in system development, then every fixed error/automated routine will give us a return of 1000. This way, 100 saved effort units (improving systems management) is opposing a potential saving of 100.000 effort units (when improving systems development). This rough calculation makes it more than plausible, that an approach in systems management alone will hardly bring the same return as a combined investigation of systems development and management. In essence: An approach that combines improvements in systems management and systems development looks by far more promising than an approach for improvements in IT management alone.

Engineering domains have experienced similar problems decades ago and come up with systems to assist, support or take over management tasks carried out by human operators. To name a few examples, autopilots and fly-by-wire systems have been introduced into aircrafts, driver assistance systems into cars, industrial robots into factory buildings. In space and aeronautics, where remote control faces serious limitations, unmanned systems have achieved a high level of automatic local control. The typical results are control systems in many different artifacts like planes, cars, rockets, robots, military systems, industrial machines, power plants, building technology. So far, IT systems have only been in their scope as far as they have been associated with the mentioned artifacts.

These demanding results need demanding development processes. Because of the often high risks and high costs of the final artifacts, engineering domains have been forced by principals to high standards in system development. Judging from the artifacts like the Mars robot, the Airbus A380, modern fighter jets, or autonomously acting vehicles, communication between designers and operators must have worked in engineering! Indeed, remarkable knowledge has been gathered in the design, testing and operation of SwSMC. Across the individual engineering disciplines, system engineering, systems analysis and systems design have been established, as in the different engineering domains the same issues came up again and again. In addition to domain-specific theory (e.g. in mechanical and electrical engineering), modeling systems in Computer Aided Design (CAD) tools and doing simulations are common-place in these domains. This way, SwSMC in engineering can be associated with mathematical foundations, formal methods and - in advanced systems - algorithms that are associated with artificial intelligence to implement analysis and planning.

Surprisingly, in IT management automation in data centers similar standard tools or algorithms are hardly known. Instead, IT management automation lacks a tradition in formal methods, modeling and simulation, and a consequent link to theory in general. Instead, routines are typically based on best practices instead of algorithms and proofs about their behavior in various conditions. When aligning automation in engineering and automation in IT management along common principles, profit in terms

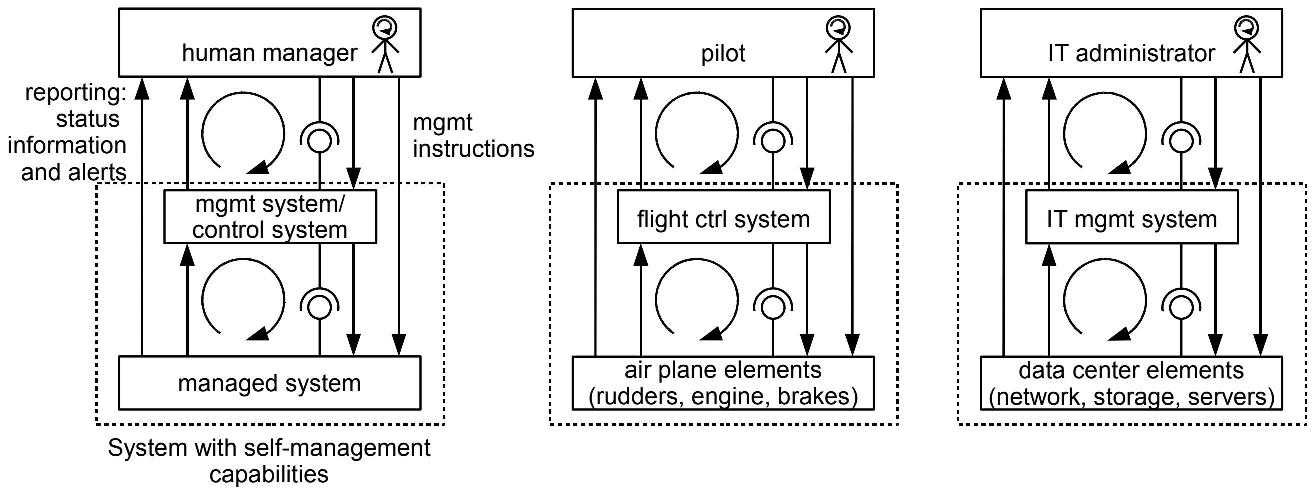

human manager
reporting: status information and alerts
mgmt instructions
mgmt system/ control system
managed system
System with self-management capabilities
pilot
flight ctrl system
air plane elements (rudders, engine, brakes)
IT administrator
IT mgmt system
data center elements (network, storage, servers)

Figure 1.9: Systems with self-management capabilities are managed systems that are tightly connected to management systems. Example systems can be found in typical artifacts from engineering domains. By learning from these domains we can transfer knowledge to the design of IT systems with self-management capabilities.

of knowledge transfer (see Figure 1.9) can be anticipated. To automate a certain IT management task, we can refer to the implementation methods associated with the respective engineering sub-domain.

To approach solutions to the problem stated above, modeling, simulation and visualization are key to predict and virtually test automatic routines for periodic management tasks. They would be an ideal environment for system developers and system operators to jointly develop a new system that incorporates automated management routines. While there are a few simulation tools, they cover either resources or processes/workflows. Not one could be found, that allows to graphically link changes in a model of managed resources with advancements in a certain process/workflow. Typical theoretical and generic simulation frameworks on the basis of discrete event simulation, petri networks are far to abstract to be of practical use.

All in all, with few exceptions, advanced self-management capabilities (SMC) are a vision for most IT systems, not current reality.

1.6 Idea: Engineered IT Management Automation (EIMA)

This thesis coins the term of “Engineered IT Management Automation” (abbreviated as EIMA) which is intended to be understood as a scientific way to automate routine management tasks in contrast to the term of “ad hoc IT management automation”.

Unlike existing process frameworks, it is not independent from technology by design, but instead aims to link resource-associated management tasks with a potential implementation which is mainly driven by technology with a minimal intervention of human operators.

Unlike general purpose management suites, when management systems are more closely associated with certain managed resources to automate routine management tasks, systems with self-management capabilities are formed. When compared to current general purpose management systems, in SwSMC advanced analysis and planning methods can be expected as a benefit of specialization, leading to more closed-loop management. In essence, SwSMC reduce the necessary frequency of management interactions between human operators and managed systems and at the same time raise the level of abstraction in management interactions. In contrast to ad hoc automation, these automation efforts typically happen at the vendor/producer side of managed systems not at the operator side or being done by third parties.

This work addresses both systems management and systems development. It aims at finding a way how these two groups can find a better level of abstraction to communicate about systems, which is a necessary prerequisite for better IT management automation. The central idea is to take over concepts from systems engineering and process control automation which have successfully applied automation in artifacts like industrial plants, aircraft and space vehicles, and cars, to name just a few.

Based on these domains, we can enable cooperation of systems management and systems development along a common level of abstraction in terms of well-engineered automation patterns. The concept of abstract patterns can improve communication, as operators as well as developers can understand them and map them into their particular view: Operators can associate a pattern with a certain behavior and derive instructions about what an operator still has to/is allowed to take care of, while designers may have a tool-kit to map patterns to solutions/methods to enable the intended behavior. Patterns also ease the process of system managers giving input to system developers/designers. In the same way, systems designers can associate certain automated management tasks to certain functional units in a system.

1.7 Approach

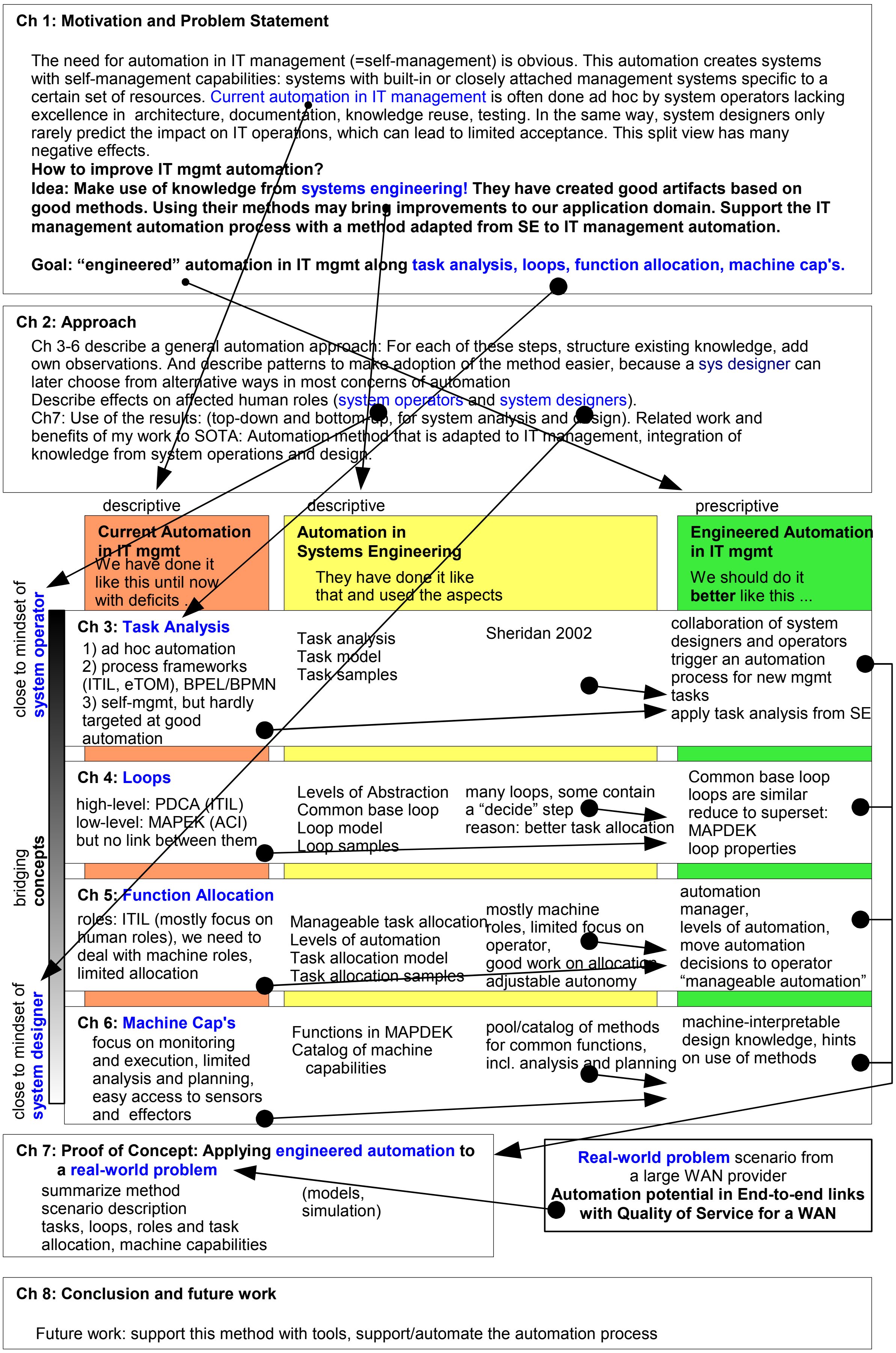
Re-stating the problem of this thesis, a mapping of resource-related IT management tasks to an implementation in terms of automated management routines needs to be found. A major obstacle in cross-domain knowledge transfer is that the descriptions of engineering artifacts are typically targeted at domain experts. Aspects that would be applicable in other domains are often hidden between loads of domain-specific information. Instead, the system function needs to be abstracted while focusing on the system's internal and external management. Therefore, for the automation approach itself, we can refer to results from systems engineering.

Reviewing related work in systems engineering—Sheridan, Humans and Automation [She02] with the author having a background in aviation and space applications—suggests the steps task analysis, breaking down tasks to functions, then function allocation to either humans or machines or collaboration of both. In task analysis he proposes to use the steps *acquire, analyze, decide, implement*. This proven method for existing automation scenarios in Engineering has been adopted to be applied to IT management automation.

The following four concepts are a combination of automation concepts described in [She02] (1. and 3.) and own work (2. and 4.). The method of Engineered IT Management Automation argues to model, design, and describe SwSMC along the patterns of:

1. **Task analysis** is applied to break down these tasks to individual steps and to find relationships among them in terms of control/management layers. The known concept of IT management processes is replaced by IT management tasks to match Sheridan's model.

2. Because of their repetitive nature, IT management tasks well fit a common **loop model**. Instead of the observe, analyze, decide, execute loop, in IT management we are already familiar with the monitor, analyze, plan, execute based on knowledge loop proposed by IBM in its Autonomic Computing Initiative, and more loops from the high-level Deming process improvement cycle (plan, do, check, act) down to control loops in engineering. The different kinds of loops are compared for commonalities and differences to find a single loop to align the different task steps to.



Ch 1: Motivation and Problem Statement

The need for automation in IT management (=self-management) is obvious. This automation creates systems with self-management capabilities: systems with built-in or closely attached management systems specific to a certain set of resources. Current automation in IT management is often done ad hoc by system operators lacking excellence in architecture, documentation, knowledge reuse, testing. In the same way, system designers only rarely predict the impact on IT operations, which can lead to limited acceptance. This split view has many negative effects.

How to improve IT mgmt automation?

Idea: Make use of knowledge from systems engineering! They have created good artifacts based on good methods. Using their methods may bring improvements to our application domain. Support the IT management automation process with a method adapted from SE to IT management automation.

Goal: "engineered" automation in IT mgmt along task analysis, loops, function allocation, machine cap's.

Ch 2: Approach

Ch 3-6 describe a general automation approach: For each of these steps, structure existing knowledge, add own observations. And describe patterns to make adoption of the method easier, because a sys designer can later choose from alternative ways in most concerns of automation

Describe effects on affected human roles (system operators and system designers).

Ch7: Use of the results: (top-down and bottom-up, for system analysis and design). Related work and benefits of my work to SOTA: Automation method that is adapted to IT management, integration of knowledge from system operations and design.

	descriptive	descriptive		prescriptive
	Current Automation in IT mgmt We have done it like this until now with deficits .	**Automation in Systems Engineering** They have done it like that and used the aspects		**Engineered Automation in IT mgmt** We should do it **better** like this ...
Ch 3: Task Analysis	1) ad hoc automation 2) process frameworks (ITIL, eTOM), BPEL/BPMN 3) self-mgmt, but hardly targeted at good automation	Task analysis Task model Task samples	Sheridan 2002	collaboration of system designers and operators trigger an automation process for new mgmt tasks apply task analysis from SE
Ch 4: Loops	high-level: PDCA (ITIL) low-level: MAPEK (ACI) but no link between them	Levels of Abstraction Common base loop Loop model Loop samples	many loops, some contain a "decide" step reason: better task allocation	Common base loop loops are similar reduce to superset: MAPDEK loop properties
Ch 5: Function Allocation	roles: ITIL (mostly focus on human roles), we need to deal with machine roles, limited allocation	Manageable task allocation Levels of automation Task allocation model Task allocation samples	mostly machine roles, limited focus on operator, good work on allocation adjustable autonomy	automation manager, levels of automation, move automation decisions to operator "manageable automation"
Ch 6: Machine Cap's	focus on monitoring and execution, limited analysis and planning, easy access to sensors and effectors	Functions in MAPDEK Catalog of machine capabilities	pool/catalog of methods for common functions, incl. analysis and planning	machine-interpretable design knowledge, hints on use of methods

close to mindset of system operator
bridging concepts
close to mindset of system designer

Ch 7: Proof of Concept: Applying engineered automation to a real-world problem

summarize method
scenario description
tasks, loops, roles and task allocation, machine capabilities

(models, simulation)

Real-world problem scenario from a large WAN provider
Automation potential in End-to-end links with Quality of Service for a WAN

Ch 8: Conclusion and future work

Future work: support this method with tools, support/automate the automation process

Figure 1.10: Structure of this work

3. Then, Sheridan's proposal is followed to **allocate task steps/functions to humans/machines**. Additionally, levels of automation summarize findings on function allocation in IT management automation.

4. Finally, individual **machine steps are associated with algorithms/methods** to implement them.

For each of the concepts in Chapter 3 to 6, requirements are named and a review of existing contributions in Systems Engineering (yellow column) as well as current IT management automation (red column) is performed. The author then combines the existing knowledge and adds own concepts to form the respective step in EIMA (green column). While the work is presented in a top-down manner from IT management tasks to implementation, the resulting structure of knowledge can also be used in the opposite direction to find applicable spots for new methods in IT management automation. The four steps represent information at a level of abstraction that educated system operators and system developers can make use of, understand and memorize. They have been carefully chosen to combine features of behavior (mainly relevant to operators to assess their remaining influence) and architecture (mainly relevant to developers to assess implementation).

In Chapter 7, the EIMA method is then summarized and applied to a real-world scenario in a proof-of-concept style.

The thesis closes with a Chapter 8 on a conclusion of the results and a list of potential follow-up work.

This work concentrates on IT management automation at the technology level, even though new technology may influence processes and the work of people. Model-based automation routines closely associated with a resource are a promising idea to reduce the visible level of complexity, if these systems with self-management capabilities leave just the right level of control to human operators and work as intended by their designers at least as reliable, effective and efficient as the (human) activities they support or replace. Better model-based tools which automate more and more steps, that had been carried out manually before, make CEOs want leaner IT departments with fewer people, on the other hand. Social consequences are however out of the scope of this work.

The following sections appear in sequential order along the top-down view of the approach, illustrated in Figure 1.10. A more detailed description of the approach can be found in Chapter 2.

1.8 Vision: A tool-set for the engineering of IT management automation

In IT management automation, the basic overall scenario is to specify, design, and build a SwSMC from scratch or to add SMC to an existing legacy system. As argued before, it is beneficial to start IT management automation during the design phase of systems. In a similar situation, CAD/CAM software has been the basis for enormous progress in engineering, enabled by computer support. Modern systems like Autodesk Inventor add basic simulation of the mechanical movement of parts. In a similar but less commonly used manner than CAD tools, CASE tools provide an aid in software engineering. The vision therefore is to have a similar development framework for IT management automation solutions.

This integrated development platform—a sketch of a potential user interface is shown in Figure1.11—would allow to:

- model and design SwSMC
  - model behavioral/architectural building blocks, which each encapsulate complexity and provide a simplified view at a certain level of abstraction.

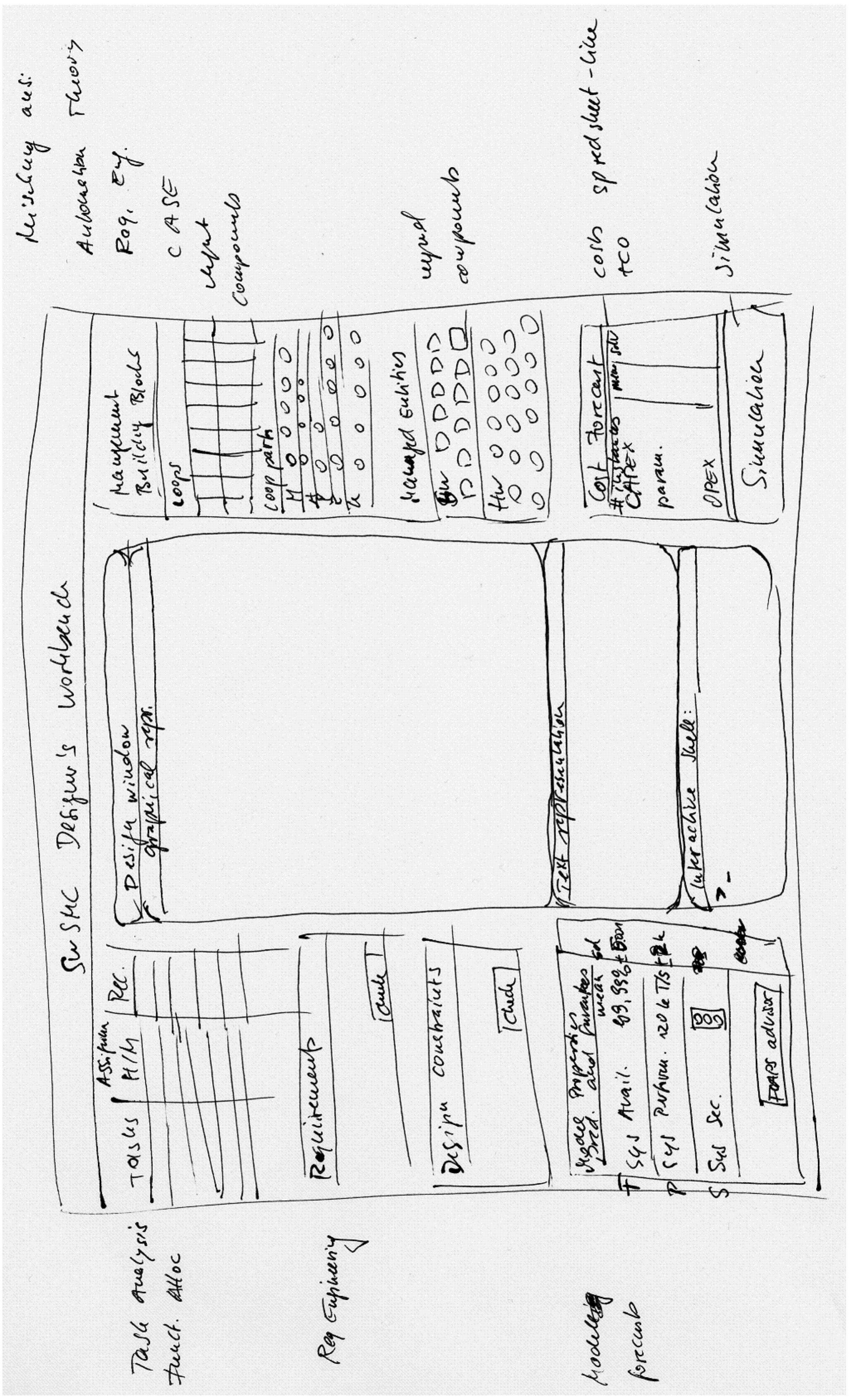


Figure 1.11: A sketch of a potential GUI for a development environment for IT systems with self-management capabilities.

  - construct IT systems with self-management capabilities from functional components, self-management components, and human-ecxecuted actions.
  - associate resource-related IT management tasks with different ways of implementation along multiple levels of abstraction.

- allow systems analysis
  - forecast of system properties (e.g. total cost of ownership, availability, cooperation of system components, management activities)
  - comparison of models/design alternatives
  - computer-aided task analysis and function allocation
- simulate the behavior and visualize the steps of the executed actions of collaboration between machine and human managers.
- iteratively improve the design of the system and the related human management processes. It is important that the system and its associated human management tasks are constructed concurrently.

The author of this thesis is confident, that the idea of a set of models for tasks, task steps, loops, function allocation, levels of autonomy and implementation patterns provided by this thesis is a necessary prerequisite to enable this vision.

2 Requirements and detailed approach on a method for engineered IT management automation

Contents

This chapter presents the deduction of the approach and the overall structure of this work. It first details the general phases and roles in IT management automation and defines the role of a system automation engineer for IT resource related management tasks.

The central task of this role is to map resource related IT management tasks to a well-conceived combination of automated implementation methods and human-done tasks and, vice versa, to find the right human-done tasks that can be automated by new implementation methods. Doing so for an IT resource pool, e.g. an IT appliance, will equip the solution with automated management capabilities, such as automated configuration management, update management, backup and restore, capacity and availability management.

2.1 The scope of an IT systems automation engineer

In a simplified manner, IT management automation can be described as a sequence of five phases involving various human roles:

Phase 1) Before automation Before automation, a number of resource-related IT management tasks are not automated or only automated to a low extent.

IT operators as well as automation engineers have had a certain level of success with automation in the past: some tasks are already automated and run fine. This means, that after an initial effort to setup and configure the automation, a certain time and cost saving as well as potentially a better service can be witnessed.

At the same time, system operators and automation engineers have also experienced the faults of automation in the past and felt that immature automation can be worse than non-automation. This results in a slow adoption process of new, potentially immature IT management automation methods, ruled by skepticism or even rejection at first. A management automation system's state will only gradually change to indispensability once it is up and running, accepted, comprehended and trusted by the operators.

As a result of this collective experience, operators as well as automation engineers see both, the chances but also the risks of automation in general. Therefore, it is necessary to deal with IT management automation responsibly as these systems often belong to the most critical ones for business. In the style of critical systems in other domains, such as medical care, air traffic control, power generation, we see the solution in engineering:

> the discipline and profession of applying technical, scientific and mathematical knowledge in order to utilize natural laws and physical resources to help design and implement materials, structures, machines, devices, systems, and processes that safely realize a desired objective. [*Engineering.* Wikipedia, The Free Encyclopedia. 23 Jul 2009.]

Phase 2) Trigger for automation In general, each automation project needs two pieces of information: a set of tasks to automate, and a set of automation implementation solutions.

The trigger for automation can therefore come from two sides:

1) From system operators, who wish to be supported or unburdened from certain repetitive tasks or their upper management wishing to cut down on labor costs. Here, the tasks to automate are given, the automation engineer has to search for a proper implementation solution.

2) From automation engineers having discovered new automation implementation solutions and searching for applicable repetitive resource-related IT management tasks done by humans to support or to replace.

Phase 3) Design of the automation system In this phase automation engineers together with systems designers use design tools to create the relevant automation system. Ideally, they plan in advance people, processes, and technology. Regarding people, abstract staff that later deals with the automation system needs to be planned. Regarding resource-related IT management processes, the tasks and automation procedures need to be planned. Regarding technology, an automation engineer needs to plan the management automation system itself, as well as plan the managed IT resources (hardware, software) which the management automation system interfaces to. As one can see, considering only one or two of those three items does not suffice. It is essential that all three are planned concurrently. This thesis deals with this phase of automation.

Phase 4) Implementation of the automation system System implementers develop the automation system according to the design or add automation functions to an existing legacy system. They think in terms of functional blocks and implementations in terms of algorithms, libraries, and code in various programming languages. This step is out of scope of this thesis, but we pay attention to the fact, that a design needs to contain enough hints for implementation. This requirement is considered by the implementation method catalog. In addition, an application example is shown in Chapter 7 as a proof of concept.

Phase 5) Operation of the automation system The actual automation system later runs at an operator's site. There it manages actual IT resources and is operated by actual IT management staff. The overall success of the automation will clearly depend on how well actual resources, actual management tasks and actual staff have been predicted and modeled in the design phase.

System operators regard the automation system as a tool to support them in their daily work. They either consider their work to cover multiple tasks on the same IT resource (such as server administrators, database administrators, network administrators) or to cover one task but on multiple IT resources (e.g. incident manager, service level manager).

The dependence of the automation system on the actual managed resources and actual management tasks leads to the fact, that "IT solutions" or "IT appliances" are more and more popular. Here, the managed resource pool (hardware and software) is known to the vendor upfront, as well as the overall usage function of the appliance, and the vendor can add to the bundle an appropriate automation system. In such a scenario, staff and organization as well as existing interfacing technology on the customer side remain the only factors which have to be approximated at design time.

Finally, at run-time users or service subscribers make use of the functionality of the the actual system. While they do not interact with managing the system, and management automation essentially would not matter to them, at least they should benefit from an overall better service from the system, when it had been equipped with IT management automation routines. We leave out the users' view in the rest of the thesis.

The role of an automation engineer Concluding from the phases, the special role of the automation engineer / systems designer can be seen, who needs to take into account both: to understand the tasks to automate, to know applicable automation solutions, to make sure that the system is implementable and to predict future operation of the automation system. The essential tasks of the automation engineer role are therefore to 1) map trouble-some human tasks to machine tasks (given a set of tasks to search for automation solutions) and 2) to use technology/theory to support existing work (given a set of solutions, searching for tasks to be supported). This mapping is therefore the central aspect of IT management automation. It is particularly interesting due to its different levels of abstraction at which an automation engineer can work and think.

Today, the role of the automation engineer too often coincides with the system operator (automation at operational phase) or system implementers (automation at implementation phase). Instead, we argue in favour of a dedicated role to do the automation design for the following reasons: 1) given the necessary specialization of experts in systems operation and systems implementation it is more and more unlikely, that experts in either of these two fields possess enough of the relevant knowledge of automation design. 2) Other domains likewise have created systems designers or automation engineers, e.g. architects design buildings. They neither do the actual construction work themselves, nor do they live in the designed buildings or operate them. Likewise, mechanical engineers design new machines, but do not assemble them, nor operate them. Instead, it's their job to preplan both assembly as well as future operation.

We take over this split of roles to IT management automation.

2.2 Requirements on a method that improves IT management automation by engineering

This thesis proposes a method for IT management automation. The result of this method shall serve as preconsiderations for a systems design that automates resource-related IT management tasks. The general idea is to replace IT management automation as an art by IT management automation as an engineering task. The goal of this thesis is to support an automation engineer of resource-related IT management tasks with a method, when he blends resource-related IT management tasks formerly done by human operators to an implementation where operators are more and more supported or replaced by IT management automation routines implemented in a management automation system.

A method aids in transforming input, that meets certain criteria to an output with desired properties. The following subsections will first show the criteria on the input of the method, and then the requirements on the output of the method.

2.2.1 Constraints on the input of the method

The following constraints are imposed on the input of the method to limit the focus of this thesis:

Constraints of input	Reason	Implications
The method is made to automate IT management tasks bound to IT resources.	IT management automation is bound to resource-related tasks because only here the automation engineer has good knowledge about the predicted/known set of resources and its management interfaces. In the end, self-management is about IT management automation closely associated with resources.	We need to consider relevant related work from the domain of IT management automation.
The resource-related IT management tasks should be given in terms of workflows.	Otherwise, automation engineers would spend much time on documenting and formalizing the current practice in systems operation. With given workflows they have a good starting point to analyse, simplify and automate the most relevant parts of the tasks.	We need to revise workflows to fit the method.
As managed IT resources, we consider resource pools and IT solutions instead of single resources.	IT management automation in single resources is already well covered with automation. There is considerable value in cross-resource automation for IT solutions.	We can leverage similarities between IT management actions on multiple resources.

2.2.2 Requirements on the output of the method

The method provides a drill-down of the tasks (operators' view) and derives from them a set of automation implementation methods (implementers' view). While mere patterns from pattern books or mere lists of implementation methods do already exist, the value of this thesis is in linking them to tasks in IT management automation. We therefore must investigate the mapping from tasks to implementation methods, and keep in mind the automation phases and roles above, that the systems design needs to stand up to requirements by system operators and system implementers.

Requirements on the output of the method to serve the needs of operators The output needs to be an appropriate communication artifact between automation engineers and system operators. We need the involvement of the operators at design time of the automation system to use their experience. They shall early influence the design so that it better fits their needs.

Requirement	Reason	Implications
Automation must be dependable for operators. Ideally, it is comprehensible, reliable, verifiable, and verified by system operators. Each of these steps will improve acceptance. At the level of abstraction they can understand, operators need to be involved even at design time to assess early whether the designed system will serve their needs. Later, at operations time, if necessary, they may need an insight into implementation.	Perfect automation would not need control. However, as automation is derived from imperfect input, it is man-made, and men make errors, IT management automation is inherently imperfect. Operators need a certain link to the implementation of an automation system, e.g. as preparation to the roll-out of an automation system or in case of wrong behavior of the system. Source code would be by far inappropriate in most cases, as the difference in abstraction to their daily work is enormous. Instead, the method must be designed in a way, so that later operators can conclude from the task the target behavior of the automation, and from the target behavior to the implementation methods. In most cases, this should be sufficient to operate and administer the system.	Operators can build up trust by knowing more about the system. They can also better assess the true capabilities of the system, for better or for worse. It is not important, that they understand every line of source code or be able to fix the system. It is essential, however, that they can work around occuring incidents with quick fixes and workarounds in incident management. A problem management process will make sure, that those workarounds will be replaced with a permanent solution, once this is available.
Automation itself must be manageable (monitor and control). Therefore operators need automation levels.	To detect and work around potential problems with automation and to use the strengths of human administrators.	Design or implementation errors in an automation system have much less impact, when operators can take over and work around them.

Requirements on the output of the method to serve the needs of implementers The output needs to be an appropriate communication artifact between automation engineers and system implementers. We need the involvement of the implementers at design time of the automation system to use their experience. They shall early inflence the design so that the design has a good implementation.

Requirement	Reason	Implications
From the design of an automation system, the implementers shall have enough hints to really implement the functionality of the system.	A design which leaves too many open questions in implementation is incomplete.	We need to refer to known implementation methods to enable automation. However, we leave performance and security to the implementers.
When a design is handed over to implementers, the links to the tasks in IT management still need to be visible.	Transparency into the supported tasks. Many detail questions will pop up at implementation time to the system implementer. Implementers can do a better job, when they can get more info on the supported tasks. Linking tasks to function blocks and function blocks to implementation blocks enables implementers to better assess the required properties of the implemented subsystem.	A system and its management automation should be conceptualized at the same time by automation engineers, and future administrators need to be taken into account.
In the design, we need to differentiate between a) IT management automation “intelligence” and b) interfacing to management resources.	We can reuse management automation intelligence on different resources by exchanging adapters to the resources.	This split needs to be made in the method as well.
This thesis shall encourage automation engineers to discover new implementation methods only rarely used so far.	Resource monitoring and execution of commands, as well as message exchange are already widely automated. With new methods, we can automate tasks that have not been automated before but where it is technically feasible to automate (analysis, planning, decision support).	The implementation method pattern catalog should span a number of research domains, e.g. AI, automation engineering, maths, industrial automation and find its methods in advanced systems such as assistance systems in cars and airplanes, digital factory planning. Yet, to show their applicability in IT management automation, the method shall show the ancor points where they fit.

2.2.3 Non-requirements on the method

IT management automation as a whole involves many more questions than this thesis can answer. Thus, some of the questions around IT management automation must be directed to related work. The following table gives a rough summary on topics in IT management automation, which are not covered.

Non-Requirement	Reason	Implications
The method will *support* an automation engineer in systems design and system modeling. It will not, however, automatically produce a complete systems design.	The design of IT management automation remains a task involving humans and informal specifications, as well as compromising many criteria. Full automation of this design process itself is therefore unlikely.	We should aim nevertheless for making the method appropriate for a machine-interpretable model. This will be the basis for modeling, analysis, and simulation tools.
We leave out an explicit cost-benefit-analysis as well as return-on-investment analyses.	While cost and time savings are the major driver for automation, this thesis concentrates on giving hints on *enabling* automation, not on justifying it in a certain scenario. Indeed, with the dynamic automation levels such an analysis becomes much more complex than the typical estimation for a "good" static level of automation typically done at design time.	We leave this topic for follow-up work.
The cooperation of loops, that goes beyond hierarchies, will not be investigated.	Hierarchical control structures have the welcome property that it is known in advance, how a final decision is composed from the decisions in each of the loops, as we can assume, that higher loops supervise lower ones and eventually overrule them. In the same way, hierarchical structures have a well-known data flow along the edges of the hierarchy tree. Engineering-wise, both of these properties are welcome to create large structures from small building blocks with a well-predictable and explainable outcome.	If automation designers have good reasons to implement *non*-hierarchical structures in management automation systems, they should have a look into the following domains. Technology-wise, most of the relevant knowledge is tagged to belong to peer-to-peer systems, multi-agent systems, self-organization, and inter-organizational IT management. In terms of concepts, many of the phenomena artificially recreated are rooted in social studies of humans, business studies and business organization, as well as animal studies and sociology.
Human capabilities are not covered.	Humans are diverse and their capabilities will be abstracted in this thesis by roles.	For a closer look, the reader is directed to work in medicine, psychology, ergonomics and science of labor.

2.3 Preconsiderations on the steps of the method

The preconsiderations in this section are about how to get the required output, essentially implementation methods associated with tasks, from the required input of the method, essentially IT resource pool related management tasks. At first, it explains that a trivial approach to map them directly (resource × task → implementation method) results in a structure far too complex to be of actual use, because the similarities among tasks, and similarities among implementation methods are not taken advantage of.

After a review of related work, intermediate layers are introduced that create a step-wise mapping from tasks to implementation methods. These steps provide a welcome ordering framework, along which automation knowledge can be aligned to create Engineered IT management automation from knowledge in IT management and systems engineering.

2.3.1 The automation problem

Figure 2.1 shows how resource-related IT management tasks, that interact with a managed resource pool, are mapped to a number of human-performed management activities and a number of machine performed management activities. As a result of automation, human IT management processes are supported by management automation systems where feasible and beneficial. The essence of automation is mapping as many tasks as possible to machines.

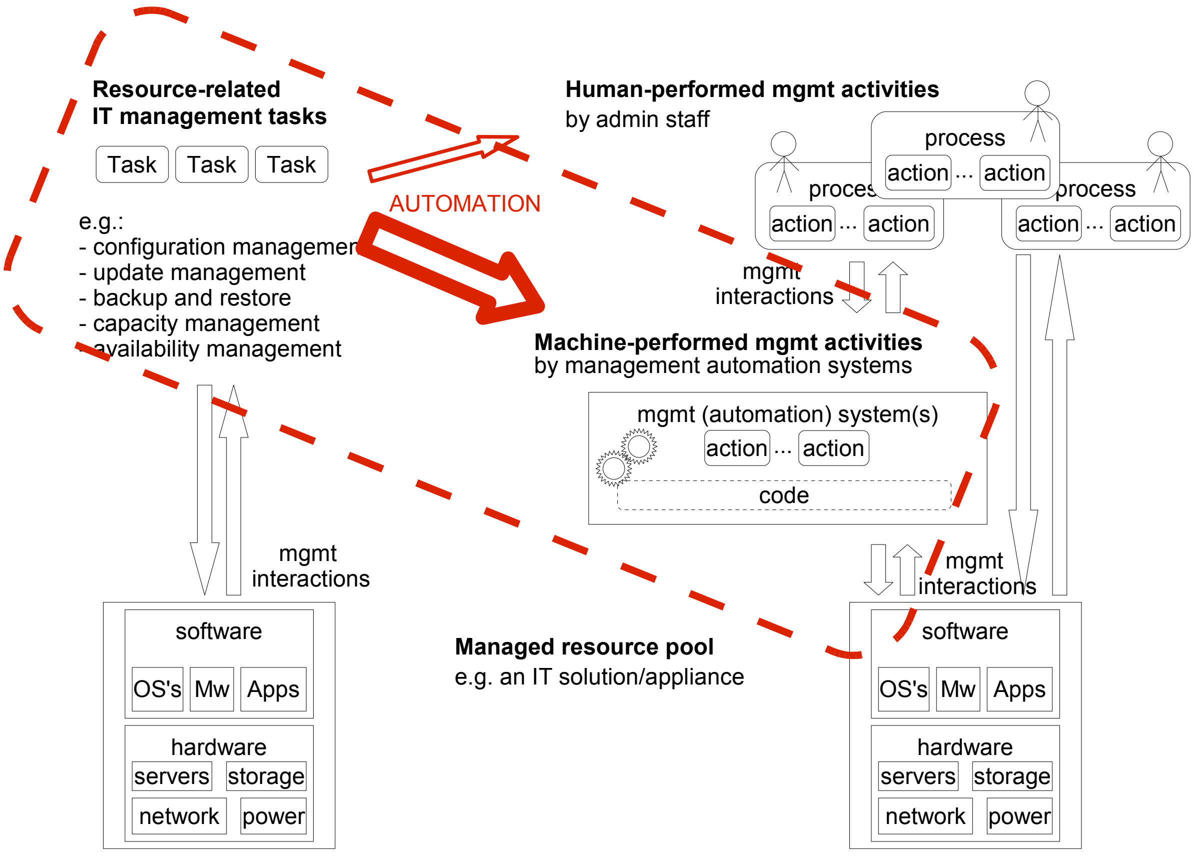


Figure 2.1: IT management automation is about the creation of IT management automation systems

We can now concentrate on the dashed box in Figure 2.2 and refine it:

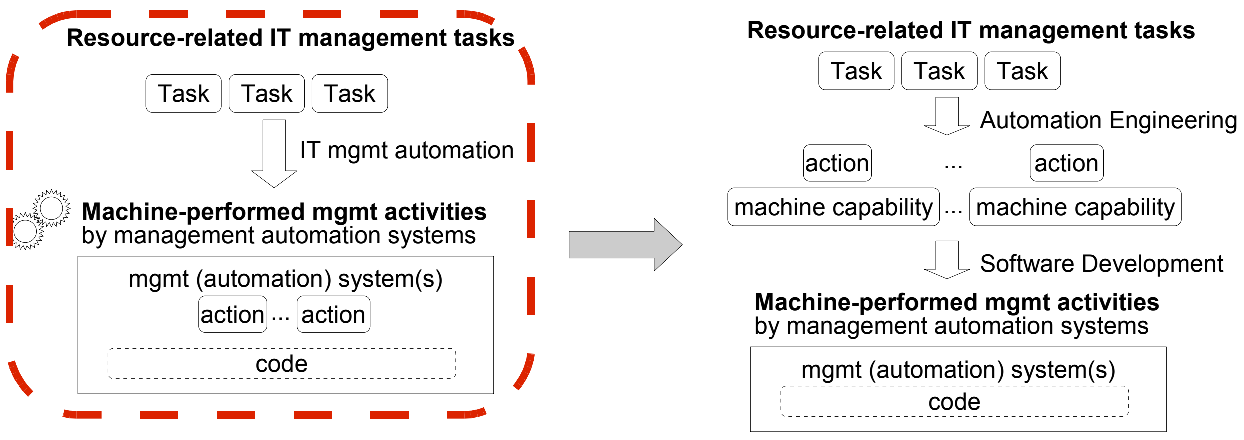

Resource-related IT management tasks
Task
Task
Task
IT mgmt automation
Machine-performed mgmt activities
by management automation systems
mgmt (automation) system(s)
action ... action
code
Resource-related IT management tasks
Task
Task
Task
Automation Engineering
action ... action
machine capability ... machine capability
Software Development
Machine-performed mgmt activities
by management automation systems
mgmt (automation) system(s)
code

Figure 2.2: IT management automation can be split into Automation Engineering (conceptual work) and Software Development (implementation work). The benefit of this split is that experts in their respective domains can work on the topics they know best.

The difference in level of abstraction between tasks and code actually contains two steps: a) automation engineering and b) software development. We should not keep concept work and coding work together, and instead split the two steps and assign each one to the respective specialists.

a) Automation engineers must not be forced to think in code, because this is not the right level of abstraction. They should think in terms of capabilities and skills of humans and machines. We call machine capabilities “implementation methods”. We collect them in an implementation method catalog.

b) The assumption is, that once a system is defined in terms of actions and corresponding, appropriate and implementable implementation methods, system implementers will take care of coding.

This way, we have reduced the necessary level of details, that automation engineers have to think at. No longer, code is essential, but knowledge about concepts.

In the following sections, we will first outline the drawbacks of a trivial approach: to keep a direct mapping of tasks and machine capabilities. After that, we will introduce two abstraction layers that reduce the complexity of this mapping.

2.3.2 Trivial approach: Direct association of tasks and implementation methods

Considering Figure 2.2, automation engineering has been reduced to mapping tasks to machine capabilities/implementation methods. In this case, we would deal with three entities: managed resources, management tasks to be performed on them, and applicable machine capabilities for management automation. In the past, automation engineers chose the applicable methods from experience. This way, automation was more an art and act of intuition than a matter of deduction.

In a simple way to formalize this one-step approach, a large recipe-book like look-up table with three columns could be created from these three entities: managed resource $\times$ task $\rightarrow$ implementation method. Each row in this table would read like: If the managed resource is the following, and the task is like this one, then the given implementation method is known to be appropriate, e.g.

IT resource: router
$\times$
Task: on-going optimization of the route table in reaction to changes in the availability and load of links in a network

→
Solution: modeling the network as a graph, then running a dynamic routing protocol based on Dijkstra algorithm to calculate shortest paths in graphs

This table would essentially conclude point solutions or individual conference papers. Each of them picks a certain resource, a certain task and a certain method. The authors then typically either evaluate current methods for the task, or they present a new solution. In effect, a huge collection of case studies would be formed which would read like a recipe book. However, there are a number of drawbacks to this approach:

- The three sets of resources, tasks, and implementation methods are large quantity-wise. A cross product of all three (the automation look-up table) would be huge, incomprehensive and necessarily incomplete.
- Unfortunately, many academic contributions are incomplete regarding the three entries in each row of the table.
- Commonalities in the structure and nature of tasks would not be utilized. The rows of the table would essentially be independent from each other.
- Cooperation of human administrators and management automation routines would be neglected.

All in all, while such a table would enable the quick lookup of solution ideas for automation problems, it includes only a subset of the necessary knowledge. It also gives too few links and associations to find relevant ideas. If a certain task or resource is not covered, then the table would have to be skimmed by an automation engineer with difficulty due to the rows not being interlinked.

As a result, a direct mapping of resource × task → implementation methods does not consider similarities among tasks and implicitly assumes full automation of the respective task. Additionally, a mere unsorted list is a bad representation for searching IT resources and tasks in this index structure that do not match any of the existing keywords. In addition, none of the entries in the table make a statement on automation levels and the split of work between humans and machines. We would have to improve the reusability of this existing knowledge. The aim is to rephrase these cases as representatives of automation patterns, to group them according to their similarities and to make them resusable along the lines of a method. With this goal in mind, related work in automation egineering, systems engineering and IT management was skimmed to find applicable knowledge on mapping tasks to implementation methods.

2.3.3 Related work on automation methods in IT management and systems engineering

The trivial approach leads us to search for related work that explains—regardless of application domain—how to automate tasks in general. Breaking up the title of this thesis "Engineered IT management automation", the following domains can be identified:

- "Information technology", e.g. IT systems in which management automation is already performed. They would provide a number of example systems each being a point solution. However, there is limited knowledge of general applicability.
- "IT management", e.g. IT management processes, IT management tasks and IT management sytems. These domains provide the task knowledge in IT management automation.
- "Management science" (also called operations research), the science to make rational (management) decisions based on mathematical models.

- “automation”, e.g. transformation of tasks into sensors, actors and machine-executed control
- “engineering”, e.g. automation engineering and systems engineering

As the domain of IT management alone did not provide us with the desired automation method, literature in other domains was scanned as well with the aim to learn from these domains and to apply their knowledge to our application domain, IT management automation.

When searching for relevant methods in other application domains we need to strip the domain-specific descriptions in these books and concentrate on the properties, which are of interest in this thesis: automation knowledge which is described in a domain-independent manner and therefore can be applied to IT management automation.

A selection of relevant literature from these domains was scanned to find relevant knowledge. The selection criteria for literature were, that the content of the publication must propose a solution *how* to do automation for certain tasks. We were not so much interested in what tasks of what domain were automated, but rather the steps of automation itself. In this chapter, we list only the most influential items of related work, while more literature will be used and referenced in the individual chapters of this thesis.

Information technology

As a replacement of many point solutions we use the “Hype Cycle for IT Operations Management 2008” (see Fig. 2.3), by Gartner Inc., an IT research and advisory firm. It shows a whole range of existing and upcoming management tools for IT operations and gives a good insight to the perception of Gartner analysts into the current state of the art and the near future. Obviously, as can be seen in the hype cycle, future IT operations will be supported by a whole range of tools.

While here we can see the trend towards more and more automation in IT operations management, we can also interpret the forecast in a way that it raises a number of upcoming problems in the future:

- Feature overlap. To name two examples: “Network Configuration and Change Management Tools” will likely overlap with “IT Change Management Tools”, and “Run Book Automation” with “IT Workload Automation Broker Tools”. We should have a better view on their individual capabilities and levels of automation.
- Cooperation or competition of automatic routines. It can be forecasted that those tools will work in parallel and manage the same IT resources. Cooperation and compatibility of the tools cannot be taken for granted. As with multiple humans, a multiple manager problem will show up, only this time, administrators have to read logfiles to know what an automated admin tool did, instead of asking his collegue next door.
- Unclear delegation of control and responsibility between humans and IT management systems.
- Network and processing overhead in terms of parallel monitoring, analysis by many tools, concurrent planning based on different assumptions, and eventually conflicting actions.

All in all, a number of tools are presented, presumably one for each task. A general approach on IT management automation can not be seen in the Gartner publications, however. Instead, we can predict a range of potential conflicts. We later see, that the method proposed in this thesis can partly prevent such conflicts or at least make them more obvious.

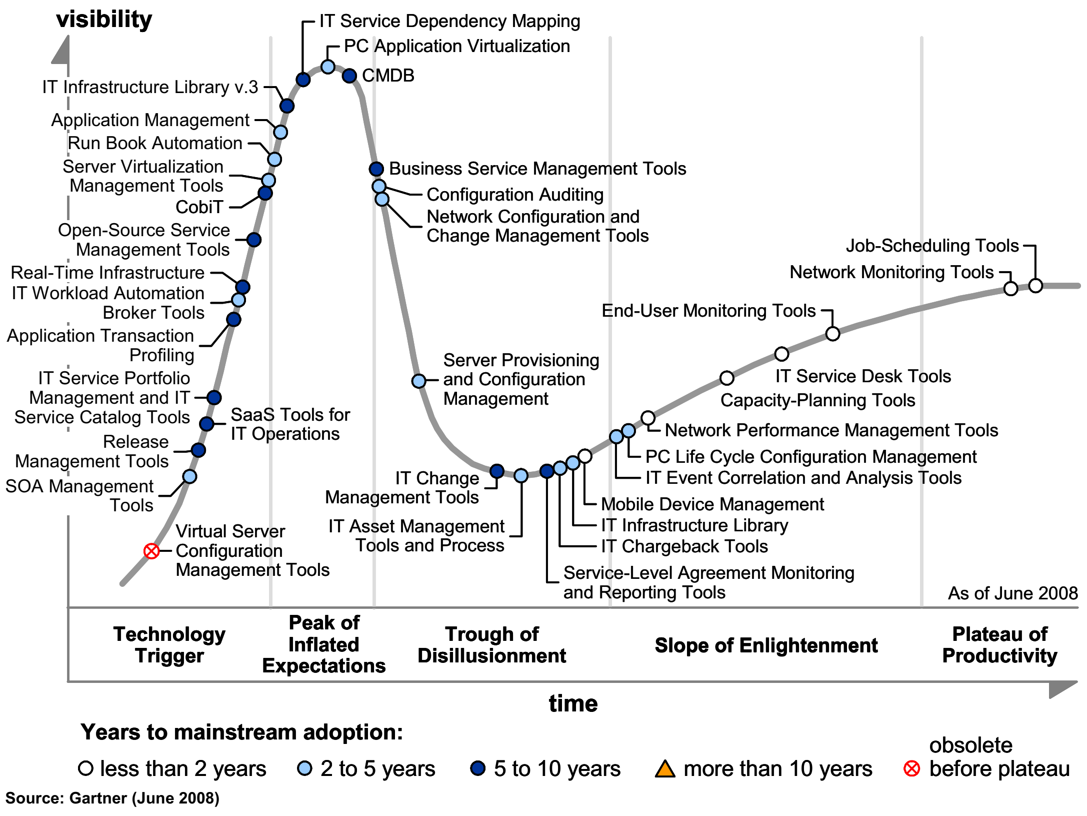


Figure 2.3: Gartner Hype Cycle for IT Operations Management 2008 (Source: Gartner Inc.)

IT management

IT management has come up with a number of related research topics to structure human processes in IT management (tasks) and technical implementation methods. The focus of this literature review was to find ideas how to translate resource-related IT management tasks to implementation methods. Therefore, literature was selected, that covers general ways to lead IT management tasks to an automated implementation. Regarding the specific question, we identified IBM's Autonomic Computing Initiative to well relate to the general problem statement.

IBM's Autonomic Computing Initiative (ACI), started in 2002, has been an important driver for research in management automation of IT resources, and therefore for this work. In its own vision, it took the autonomic nervous system as an example, which at the same time can take conscious control over certain body functions (such as breathing) but also be performed autonomically without direct awareness. This thesis uses a number of concepts of the ACI, namely being a network of autonomic managers each implementing the idea of a common feedback loop, the concept of interacting loops, as well as various levels of automation. Still, the automation approach itself described in the ACI remained fuzzy at best. This thesis blends these concepts with an approach on automation from systems engineering to create a link to human-performed tasks.

A number of additional publications from IT management contribute concepts to IT management automation: At the task/process level, the IT Infrastructure Library (ITIL) structures intra-organizational IT management tasks and provides best-practice process descriptions and roles in service design and service delivery. Business Process Modeling Notation (BPMN) is a standard to graphically layout processes, mostly being executed by humans and thus designed to be read by humans and used for documentation rather than automated execution.

Technology-wise, in IT management there is a whole range of management protocols for the diverse IT systems. Some of them standardized, like Internet management with SNMP, OSI management with CMIP, IETF standards like NetConf, and an uncountable number of proprietary management protocols and management tools at different levels of abstraction and automation.

All in all, while a number of "ingredients" exist, an approach to IT management automation that covers human processes to the actual implementation could not be identified in the endeavour to find related work. Later in the individual chapters, we will more closely investigate related work from IT management, where they better match the individual subtopics. However, in IT management work is missing that blends tasks to automated execution of formerly human-performed decisions.

Management science/Operations research

A valuable source of knowledge are decision support systems, generally applying computational methods to problems in business management. Instead of aiming at total automation of management operations, they apply algorithms to an underlying model of the problem, and support human deciders with computational analyses, plans, schedules, and forecasts, based on mathmatical models, in some cases originating from artificial intelligence.

Mora et al., "Toward a Comprehensive Framework for the Design and Evaluation of Intelligent Decision-making Support Systems (i-DMSS)" 2005 presents a framework for the design and evaluation of intelligent decision support systems, that takes a bottom up view to form decision support systems from machine capabilities. As can be seen in Fig, 2.4, the authors link machine capabilities with management decision tasks, but the direct link from the architectural capability level to the decisional service task level with its many links repeats the problem of the direct mapping stated above. In the large number of links of arrows between the architectural-capability level and the decisional service-task level the aforementioned complexity of a one step mapping can well be recognized.

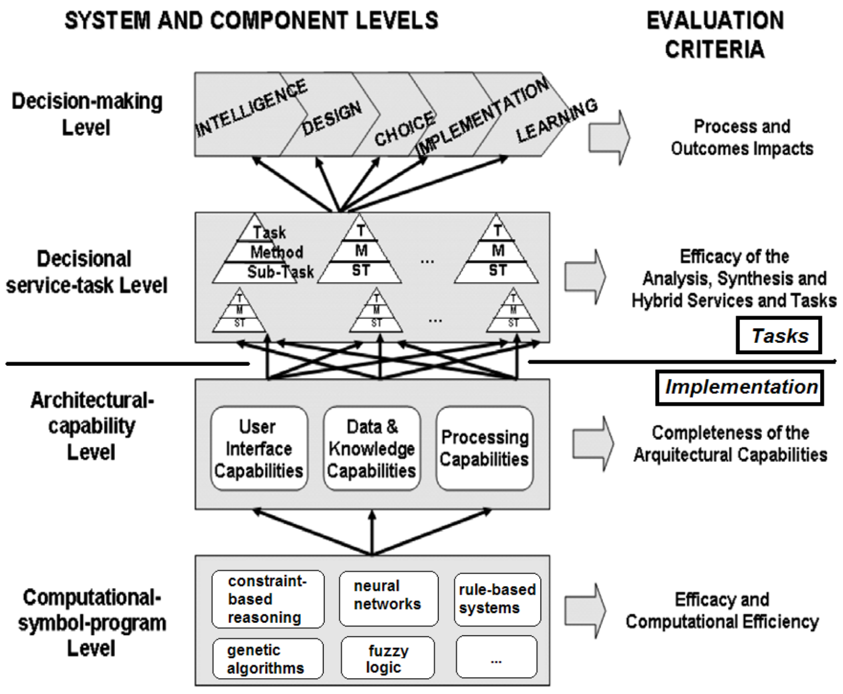

SYSTEM AND COMPONENT LEVELS
EVALUATION CRITERIA
Decision-making Level
INTELLIGENCE
DESIGN
CHOICE
IMPLEMENTATION
LEARNING
Process and Outcomes Impacts
Decisional service-task Level
Task
Method
Sub-Task
T
M
ST
...
Efficacy of the Analysis, Synthesis and Hybrid Services and Tasks
Tasks
Implementation
Architectural-capability Level
User Interface Capabilities
Data & Knowledge Capabilities
Processing Capabilities
Completeness of the Arquitectural Capabilities
Computational-symbol-program Level
constraint-based reasoning
neural networks
rule-based systems
genetic algorithms
fuzzy logic
...
Efficacy and Computational Efficiency

Figure 2.4: Framework for design and evaluation of IDSS (adapted from Mora et al, 2005)

Systems engineering and human factors design

A number of books covers systems engineering, which can be considered a discipline that covers the essential commonalities of systems analysis and systems design from multiple engineering domains. Thus, it is cross-domain by design.

Sheridan, "Humans and Automation" (2002) differs from other related publications, that he considers both the technical and human side of automation, and in particular their interaction and collaboration. He first lists a large number of application domains where humans interact with automation. These are the domains where point solutions in automation have been invented, among them: aircraft and air traffic control, automobiles and highway systems, trains, ships, spacecraft, teleoperators for hazardous environments, motion rescaling, and other nonroutine tasks, virtual reality for training and entertainment, nuclear power plants and process control, manufacturing of discrete products, hospital systems, command and control in defense and civilian emergencies, office systems, education, the smart home, and automation on the person.

After a brief introduction to these automation domains, Sheridan switches to the cross-domain view of systems engineering, and investigates a number of questions related to the interaction of humans and automation, among them an approach how to do automation: There he proposes the steps of:

- hierarchical task decomposition (see Fig. 2.5)
- tasks being aligned to a cognitive loop structure (see Fig. 2.6)
- a minimal task evaluation scheme (see Table 2.7)
- a scale of degrees of automation (see Fig. 2.8)
- and combines degrees of automation with the task steps (see Fig. 2.9)

Regarding the relevance for this thesis, the information presented in this book is described at the level of abstraction, which we hoped it to be described at for systems engineering. Sheridan really presents a scheme along which automation can be done in general from tasks to function allocation. In this book, there is also an appendix for implementation methods, even though they are not linked to the task steps before. Instead they appear without any particular order. In general, among the literature in this category, this book is the one which best applies to knowledge transfer due to its chosen cross-domain view.

In the following chapters of this thesis, Sheridan's automation structure is the basis for our approach, as it takes care of many of the requirements listed above.

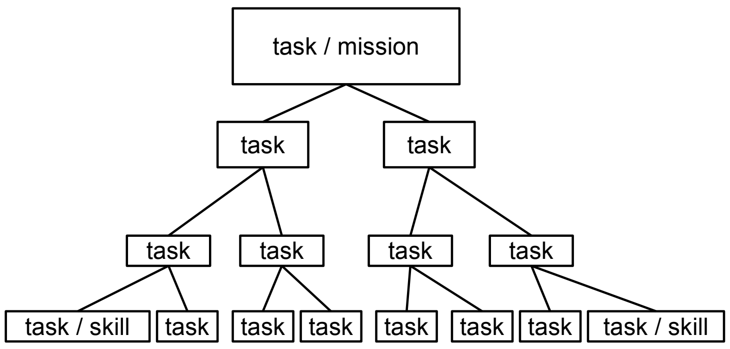


Figure 2.5: Tasks are typically decomposed along a hierarchical structure. Top-level tasks have been named "mission", bottom-level tasks "skill" in research on unmanned systems or "function" in system engineering. (Source: [She02])

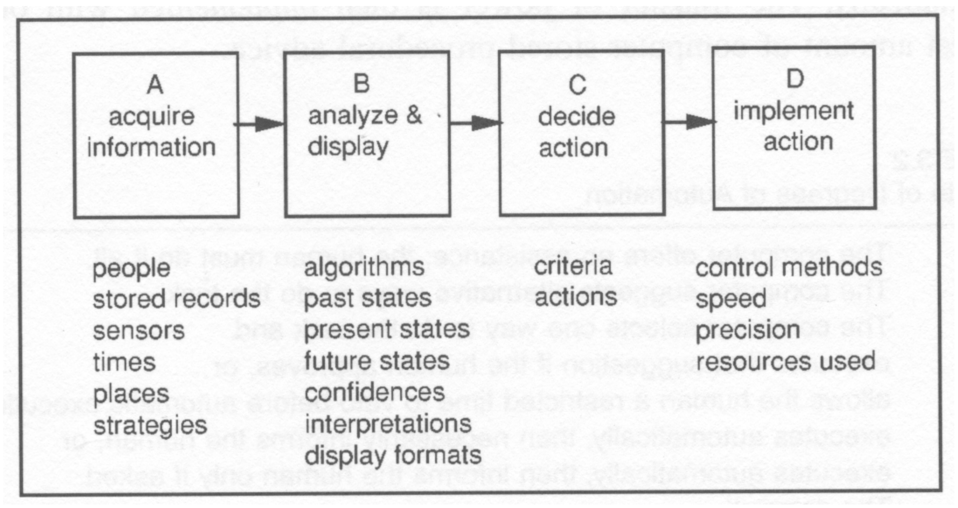
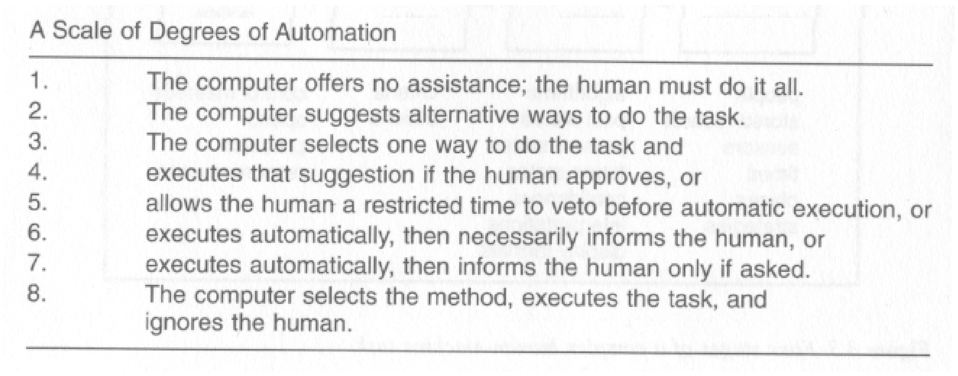


Figure 2.6: Each task is split into four stages that resemble the structure of a cognitive loop (Source: [She02])

Figure 2.7: This control loop scheme is also visible in an proposed structure for verbal task analysis (Source: [She02])

Task step	**Operator or machine identification**	**Information required**	**Decision(s) to be made**	**Control action required**	**Criterion of satisfactory completion**


A Scale of Degrees of Automation

1. The computer offers no assistance; the human must do it all.
2. The computer suggests alternative ways to do the task.
3. The computer selects one way to do the task and
4. executes that suggestion if the human approves, or
5. allows the human a restricted time to veto before automatic execution, or
6. executes automatically, then necessarily informs the human, or
7. executes automatically, then informs the human only if asked.
8. The computer selects the method, executes the task, and ignores the human.

Figure 2.8: In addition, automation levels are described, that span a fine grained scale from completely human work to completely computer work. (Source: [She02])

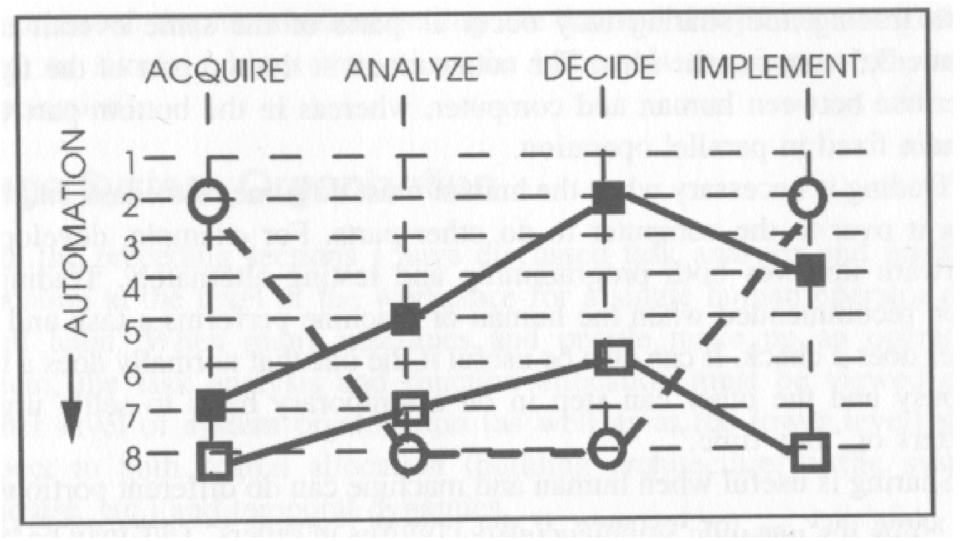


Figure 2.9: Finally, the task stages are combined with the automation degrees forming automation profiles. Sheridan imagines these profiles to be fixed at design time. (Source: [She02])

2.4 Idea: A four-step approach reduces complexity of mapping

The review of related work has shown, that in various domains there are a number of concepts to replace the direct mapping of tasks to implementation methods with a more deductive approach. Both tasks as well as implementation methods have an enormous diversity, therefore one important aspect in choosing these intermediate layers was that they should be simple, though versatile.

The following deduction shows, how tasks (layer 1) are step by step associated with implementation methods (layer 4), by the introduction of intermediate steps in EIMA.

2.4.1 Refinement of tasks

As said before, with a direct mapping of IT resource $\times$ management task $\rightarrow$ machine capabilities, automation engineers would have to map from e.g. “update management for a SAP solution” to “We need the following machine capabilities: software package self-description, version monitoring and comparison, compatibility knowledge base, compatibility constraint satisfaction solver, update utility function, update planning and scheduling, update plan recommendation and decision, transaction-based update execution, distribution scheme conscious of potential bandwidth restrictions”. In addition, unstructured task descriptions are hard to comprehend and thus to map to an automated execution. Even with the requirement of task descriptions to be given in workflow diagrams (they are the most common format to describe tasks in IT management today), the internal structure of the task descriptions in the swim lanes is enormously diverse.

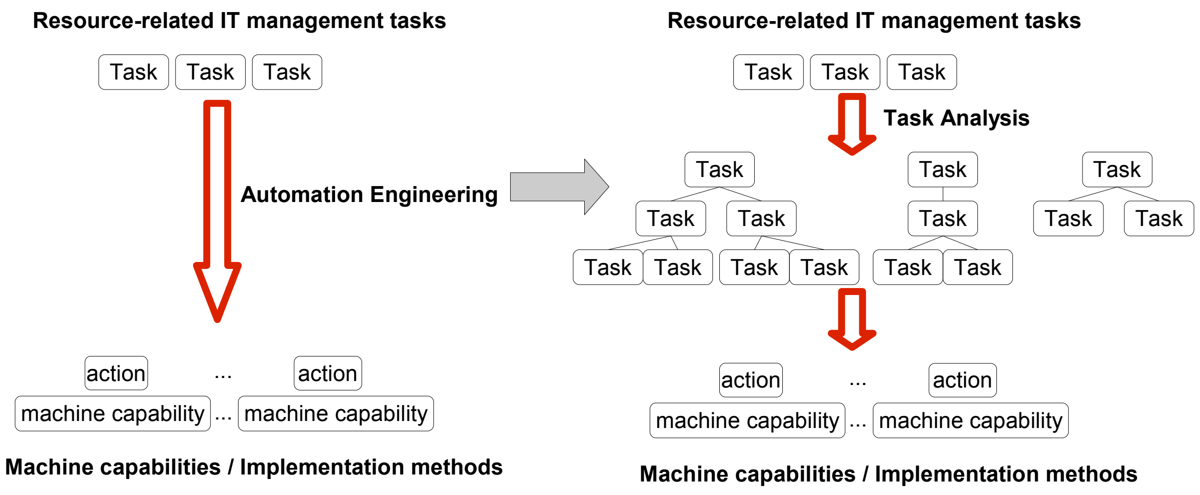

Resource-related IT management tasks
Task
Task
Task
Automation Engineering
action
...
action
machine capability
...
machine capability
Machine capabilities / Implementation methods
Resource-related IT management tasks
Task Analysis
Task
action
machine capability
Machine capabilities / Implementation methods

Figure 2.10: Tasks are structured into hierarchies. This way they are arranged in a way that reflects their inherent control hierarchies.

As a result of the literature review, we take over proposals from two sources (Sheridan 2002), and (Albus 1973) to split tasks into sub-tasks hierarchically (see Fig. 2.10), which is a standard approach e.g. in AI and robotics, as it nicely fits hierarchical task network planners. Now, with the requirement on the input in place, that all task descriptions must be given in workflows, trivial hierarchies could be created from them, as every workflow is a task or process, and all the individual steps are sub-tasks. However, such a trivial and lossy transformation would not help us. Instead, we need a task hierarchy created by an automation engineer, that understands the inherent control hierarchies in the workflows. See Chapter 3 for an in-depth investigation of task analysis in IT management automation.

2.4.2 Reformatting of tasks/sub-tasks to a standard loop

Now, automation engineers would have to map from sub-tasks to machine capabilities. This is still hard, as the (sub-)tasks are not sorted into any categories. For automation engineering, it is problematic that they are not arranged in a way, that the applicability of implementation methods can be directly derived. We therefore need a small number of "task categories" to associate sub-tasks with, so that they better sort the activities into the categories, that later can be mapped to a certain set of machine capabilities. These sub-task categories must also structure the sub-tasks in a way, that those sub-tasks/actions that interface to the managed resources (interfacing action) are divided from those that do not (management intelligence actions).

From the review of related work, we can justify that the majority of management tasks can be aligned to an inherent structure of one or more cognitive loops. As management tasks are typically repetitive, loops are an appropriate representation. Indeed, a number of publications describe remarkably similar loop concepts to be used for structuring tasks in management automation. Potential candidates include amongst others: Control loops as used in engineering, MAPEK loops as used in the ACI (Kephart 2002), AADI loop (Sheridan 2002), OODA loop (Boyd, 1970's), PDCA cycle for quality improvement (Deming, 1950's) .

As a result of a loop comparison (see Chapter 4 for details), we use the following cognitive loop as a superset of the described loops: Monitor, Analyze, Plan, Decide, Execute, based on Knowledge (MAPDEK loop), see Figure 2.11. For putting loops into hierarchies, we need to define a base loop with an upper and a lower interface. Therefore, these loops are coupled to resources or lower loops by touch-points (sensor/actor interface) as defined by the ACI. Loops also have an upper touch-point to be controlled/managed by other loops or humans.

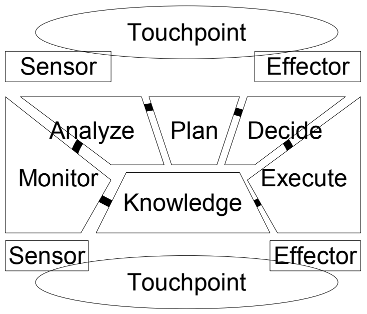


Figure 2.11: The MAPDEK loop forms a basic building block. It is the result of a combination of multiple cognitive loops described in the literature.

From MAPEK we take over the remarkable property, that we can quite easily hold the steps apart, that are either easy or hard to automate. All actions at the sensor/actor interface, "Monitor", and "Execute" seem to be relatively easy to automate, as this has been done many times. Indeed, many papers on monitoring have been published in the last decades. Relatively hard to automate are the steps that involve reasoning and deciding between a set of alternative choices, which covers the steps "Analyze", "Plan", "Decide", and capabilities involving the "Knowledge" repository.

To match the hierarchical task structure, loops can well be put in hierarchies recursively. When mapping tasks and sub-tasks to loops, we can expect that some of the lowest-level sub-tasks were actually just loop steps in a leaf-level MAPDEK loop and thus the overall number of loops is lower than the overall number of sub-tasks (see Figure 2.12). A structure with less components, which each have a uniform MAPDEK pattern, reduces the perceived complexity at the degrees of scale and heterogeneity.

The MAPDEK structure also nicely fits the one of decision support systems, where the highest ac-

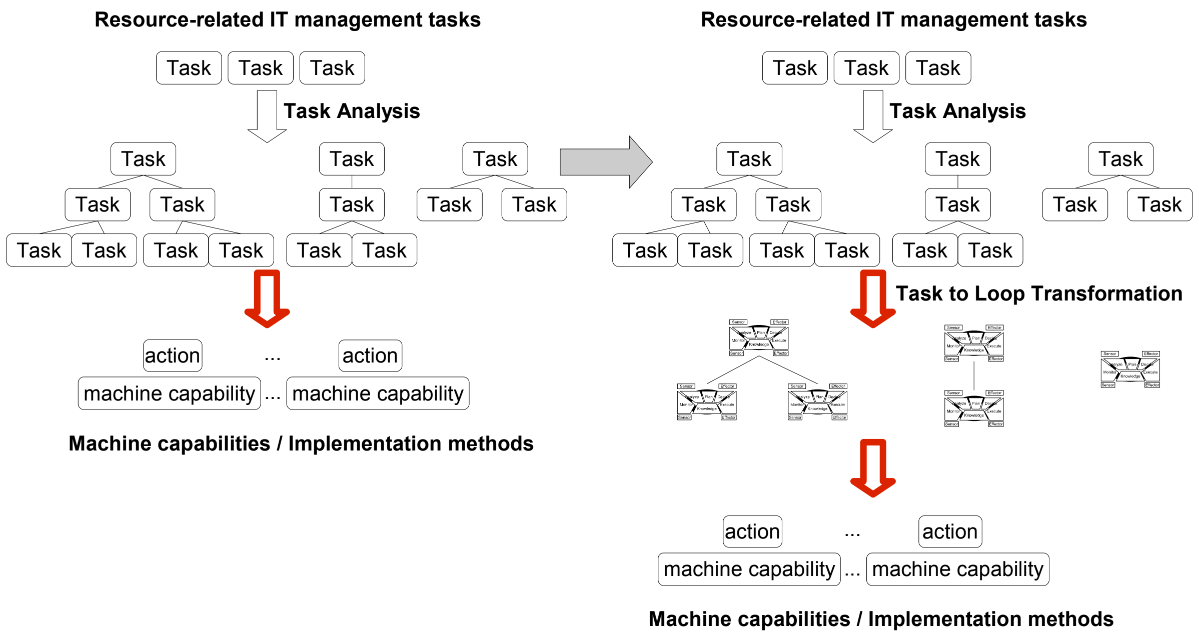

Resource-related IT management tasks
Task
Task Analysis
action
machine capability
Machine capabilities / Implementation methods
Resource-related IT management tasks
Task Analysis
Task to Loop Transformation
action
machine capability
Machine capabilities / Implementation methods

Figure 2.12: A transformation of tasks to cognitive loops with a common structure groups sub-tasks into rough categories. At the same time, this mapping reduces complexity in terms of scale and heterogeity.

ceptable level is, that a human decision is made from automatically processed input data, and then the resulting actions from this decision are automatically executed. By choosing the MAPDEK loop as our basic loop, while we loose some properties of other more engineering-style loop concepts (e.g. control theory accompanying control loops), we gain a universal framework to relate implementation methods to at a level of abstraction appropriate at this point.

Finally, loops have the welcome feature, that levels of automation can be attached to them intuitively. Terms like “human in the loop” or “human out of the loop” are known terms in research on human factors.

2.4.3 Allocation of the loop steps to humans/machines

Very few publications can be found that specifically deal with function allocation in IT management automation. Even manuals of management automation systems have shortcomings, as they may list the capabilities of the management automation system itself but do not describe which tasks are left to administrators. This is a problem because it leaves doubt and uncertainties with administrators about their opportunities to control the system.

To overcome this problem, function allocation forms an explicit own layer in our approach, where the individual steps inside a loop are assigned to either humans or machines, see Figure 2.13, being inspired by Sheridan’s work. As said in the beginning of this deduction, automation is mapping tasks to actions, performed by machines where feasible—with the rest of the tasks done by humans, either deliberately or because of non-feasibilty or predicted non-acceptance of automation.

We can consider a solution of an automation problem as more automatic, as more actions are given to machines instead of humans. Therefore, an automation engineer can assume the implicit desire, that all actions that are well automateable should be automated: “The more automated a system is, the less do operators have to manage the system.” But automation can also be seen from a different

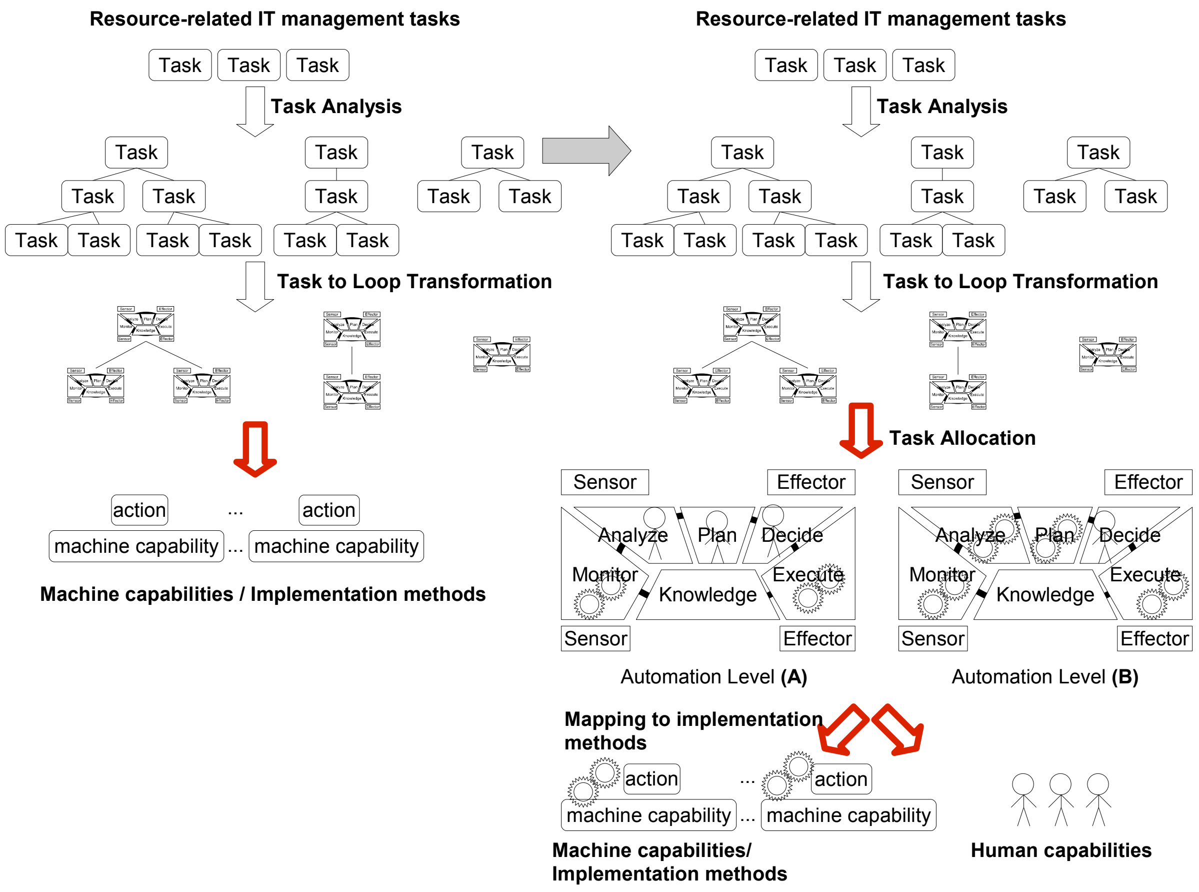

Resource-related IT management tasks
Task
Task Analysis
Task to Loop Transformation
action
machine capability
Machine capabilities / Implementation methods
Task Allocation
Sensor
Effector
Analyze
Plan
Decide
Monitor
Knowledge
Execute
Automation Level (A)
Automation Level (B)
Mapping to implementation methods
Machine capabilities/ Implementation methods
Human capabilities

Figure 2.13: An explicit function allocation step allows assignment of loop steps to either humans or machines. These levels of automation can be static or dynamic.

perspective: "The more automated a system is, the less do operators have the opportunity to manage the system." There are good reasons for not automating all steps which are feasible to automate: e.g. skepticism by administrators (caused by experience in the past), keeping a certain level of flexibility for administrators, especially in phases of fault analysis and managing incidents of the automation system.

In the past, function allocation was often done as a one-time decision (see Sheridan 2002), as automation engineers or systems designers respectively, fixed the function allocation at design time. This is not always welcome by operators, especially when automation fails. For this reason, this thesis argues in favor of not fixing the level of automation at design time, instead predicting multiple carefully chosen automation levels at design time taking into account the particular tasks of the management automation system, and having the administrator choose among them later. With the different efforts to automate certain loop steps, there result "levels of automation".

They are handled by function allocation, with "tasks" (or "functions" as Sheridan calls them here) being the loop steps. Such, the loop steps are assigned to either a human administrator roles, or a machine role, which will later be incorporated by a piece of code/software.

With automation levels expected skepticism can be predicted and reduced by shipping a system with a low level of automation (such as "list possible actions" mode, i.e. the system lists a number of available actions). In this phase administrators can learn about the system, they select an action themselves, then have this choice executed. Eventually they may switch to recommendation mode and see that the recommendation of the computer essentially matches their own chosen action most of the time or all the time. Then they may switch to confirm mode. When they recognize that they essentially always confirm they can go to a mode, where they only will be informed of the taken actions of the system.

Essentially, by switching to a higher level of automation, administrators can get "outside" of the loops and have certain actions performed automatically, gaining more time to care about other duties. In contrast, by switching to a lower level of automation administrators can get "inside" the loops and perform certain actions themselves. These levels of automation will also be welcome for fault analysis of the automation system and imperfections in automation that cannot be fixed on-site, but only worked around.

While in a perfect world, automation systems would run fine all the time and these activities would not be needed, we believe that offering multiple levels of automation will not regarded as a replacement for a carefully designed system but as a chance for operators to gain a certain level of information, influence and control when necessary.

Administrators typically only cover a portion of a 24h day at work. The automation levels can be particularly well accepted at night, as a kind of automatic operator (ideas and software on this function exist already) as a replacement for lower educated night-shift workers as in many scenarios "follow-the-sun" operation of IT resources (people in global timezones each being 8h apart take care of system management) will just prove to be too expensive.

2.4.4 Choosing machine capabilities from a catalog

Finally, after the tasks have been structured hierarchically, broken down into classified steps in loops, and many of them loosely assigned to machines, we need to associate the machine sub-tasks with actual machine capabilities in terms of running a certain machine activity to transform a certain input into a certain output.

Here, it becomes clearly visible, that IT management automation cannot work in a strict top-down manner because assigning sub-tasks to machines also requires that the required methods are actually available, feasible and accepted. New machine capabilities, such as new algorithms turned into software with better efficiency or effectivity in composition with fast, cheap processing will attract new sub-tasks. In contrast, poor software, or more demand on control, flexibility, or policy requirements may not allow automation up-front or hand back tasks to human execution. Not only the last two steps in the EIMA method interact with one another, it's essentially all steps that influence the neighboring ones.

To collect and structure the machine capabilities, we align the methods and capabilities to the steps in the MAPDEK loop that was also used in the formation of loops. We define two types of patterns: 1) capabilities, implementation methods, or typical sub-tasks in individual loop steps (e.g. analysis can be done by genetic algorithms, fuzzy logic, a constraint satisfaction solver or cased-based reasoning), and 2) to keep commonly used assembled loops, we also collect patterns that implement whole loops, e.g. a watchdog loop, a classifier loop, a load balancer loop, or a filter loop.

Regarding the touch-point (sensor/actor interface) to the actual resources, luckily in IT management automation we have few problems as most IT resources to be managed have a defined management interface already, even if it may be proprietary. At least, we do not have the problems to interface with the real world, as commonly experienced in robotics and many practical problems in automation engineering. Instead, many actions on "physical" items such as power cables, switches, network cables or servers have already been virtualized in recent years and therefore made manageable by automation. This development has shifted the focus from monitoring to actual closed-loop control.

Examples for machine capabilities are among others:

- performing monitoring actions via a monitoring interface speaking a monitoring protocol
- finding solutions to sets of equations with a constraint solver
- extrapolating likely future values for data series with regression analysis
- finding correct sequences of actions to reach a goal from a starting point with planning algorithms
- deciding how to commit resources to a variety of potential tasks with scheduling
- making sure, that actions are run in a defined manner using a transaction-based management execution protocol and interface

As has already been stated in Section 2.1 and shown in Figure 2.2, in our approach we separate the design of a management automation system from its implementation. Thus, an automation engineer translates task descriptions given in terms of workflows to a set of activities done by machine roles, and to a lesser extent by human roles. All tasks given to machine roles are supplemented by a set of feasible, appropriate, or promising machine capabilities, so that from the feasibility of the parts we can assume an implementation of the whole management automation system to be feasible. In fact, the overall idea of the catalog is, that a certain management automation system can be composed in an abstract manner out of the piece parts.

In this thesis, the tasks performed by software developers implementation-wise are not further examined. The assumption is, that once a system is defined in terms of tasks, loops, function allocation and corresponding, appropriate and implementable machine capabilities, system implementers will take care of software development, such as software architecture, information and data modeling in the actual problem domain, selection of libraries/middleware, coding work, unit testing. However, the automation engineer shall add non-functional requirements when necessary and stay in touch with system development to evaluate whether the system implementation actually follows the abstract systems design.

2.5 Conclusion

This chapter details the phases of IT management automation and the task of an automation engineer. In the scope of this thesis, the automation engineer combines knowledge from IT management and systems engineering and at the same time keeps an eye on human factors. Thus, he maps tasks to a cooperation of sub-tasks performed by machines (preferred where possible, feasible and accepted) and sub-tasks performed by humans.

If this mapping would have been performed in a casuistic style, it would result in an enormous complexity caused by a large number of cases. At the same time, it would contain much redundancy as the similarities among the tasks could not well be leveraged. Thus, in this thesis knowledge from different domains was applied to replace the direct mapping with a four-step mapping. The resulting four step approach called “Engineered IT Management Automation” (EIMA) significantly improves reuse of knowledge for all resource-related IT management automation problems.

The next chapters each cover one of the steps in detail, in a top-down order of the approach. They will include individual requirements derived from the general requirements listed in this chapter and refer to related work that is relevant only in the context of this particular step.

3 Step 1: Task analysis and task modeling

Contents

Among the four steps in the EIMA method to create a design concept for an IT management automation system, tasks are the entities that are closest to the mindset of IT administrators. The analysis of system administrators' either potential or real tasks is therefore a good start for automation. This chapter provides a more detailed investigation of task analysis and the nature of common tasks in IT management.

This chapter's content follows this line of reasoning: After two motivating examples and their drawbacks, we state three requirements on task analysis in EIMA. These requirements are general enough, so that essentially every IT management automation project would inherently include them. For this reason, the requirements are not many and they are formulated in a way to leave a certain room for solutions.

As can be seen from this chapter's table of contents, the general outline by design is a tabular one. The requirements form one axis of this table. Along these requirement, first the state of the art in task analysis in IT management is reviewed. As this review leaves a lot to be desired, we do a state of the art review in other domains more experienced in automation. The chapter finally concludes the findings and proposes a combination of previous approaches for EIMA. In the subsequent Chapters 3, 4, 5, 6, all four steps in EIMA will be dealt with in this form.

So here in Chapter 3, a descriptive section follows, where the domain of IT management is researched for current related knowledge matching these requirements. There it can be seen, that a lot of "ingredients" for task analysis in IT management do exist, but this domain seems to lack a certain methodology to do task analysis in a manner that better paves the way to automation. For this reason, then a cross-domain search for current knowledge is performed to be inspired by their findings, some of them being most recent, others of them being decades old. The "best of both worlds" is finally composed to task analysis in EIMA, written as a proposal to be most actionable for a potential user of this method. Each individual section is structured according to the requirements.

3.1 Motivation and current examples of task analysis

Examples Current task analysis in IT management automation is typically performed to document the state of current procedures in an enterprise or to design reference processes. A typical result that shows the interaction of an IT department with the HR department is shown in Fig. 3.1. Another one, for a change control workflow, is shown in Fig. 3.2. Both are real world examples.

EIMA is about engineered IT management automation, and more specifically *quickly* finding potential for automation in IT management operations and in aiding system designers and system administrators to jointly make better decisions around IT management automation. In this endeavour we need to concentrate on the relevant aspects for automation, otherwise we would be not able to propose automation routines unless we had the whole system figured out.

Criteria Along the EIMA method (see Chapter 2 for an overview), the following automation aspects will be most relevant:

Relevant tasks (Ch 3) The tasks need to be relevant for IT management automation. Ideally, they should also be relatively complete. It makes limited sense to include tasks where automation is not desired. This however, will depend on the specific scenario. To show their relevance to IT management, it should be possbible to associate the tasks with standard IT process frameworks.

Task control structure and task decomposition structure (Ch 3): Business management organization inside an enterprise is typically hierarchical as hierarchies have benefits of clear responsibilities and distribution of control, which enables separation of concerns and tracing faults. We can take this as a role model because IT management automation also includes the great risk that responsibilites are blurred by giving them to machines, while in fact only people can be held responsible for certain actions. It would be beneficial to see these control hierarchies in the task analysis results, because when taken over to systems design, then responsibilities would be clear and automation would lead to less confusion on responsibilities.

Tasks can be described at multiple levels of abstraction: in either very general terms or in great detail. It would be beneficial to have a mechanism to use variable levels of task detail, because then we could make shortcuts, where the assignment to either side is clear anyway. EIMA is not about specifying human work in detail, nor machine work. A "detailed enough" task decomposition reduces perceived complexity and saves time. The reader should keep in mind, that task decompostition is a different story than task control, so these two aspects should not be mixed up.

Standard task categories (Ch 4): Task descriptions/names have an enormous (semantic and syntactic) range but automation does not equally well apply to all of them. This means that in a set of e.g. hundred tasks it is hard to find the ones, that are more suitable for automation. In addition, the differences in task names (e.g. "sort incidents by ascending priority" or "classify problems into categories") make it hard

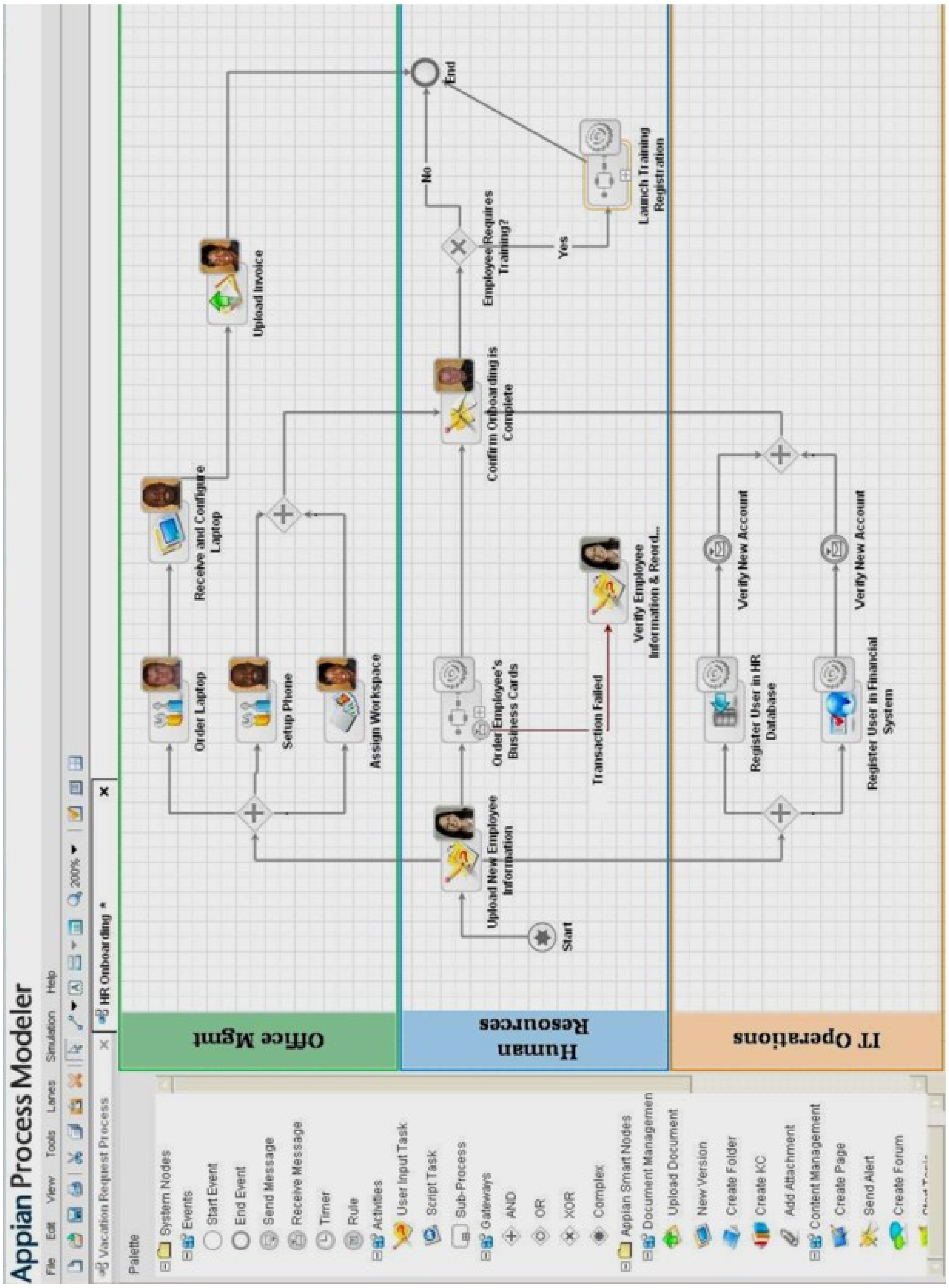


Figure 3.1: Workflow example 1: An HR process interfacing with IT operations, Source: `http://www.brsilver.com`

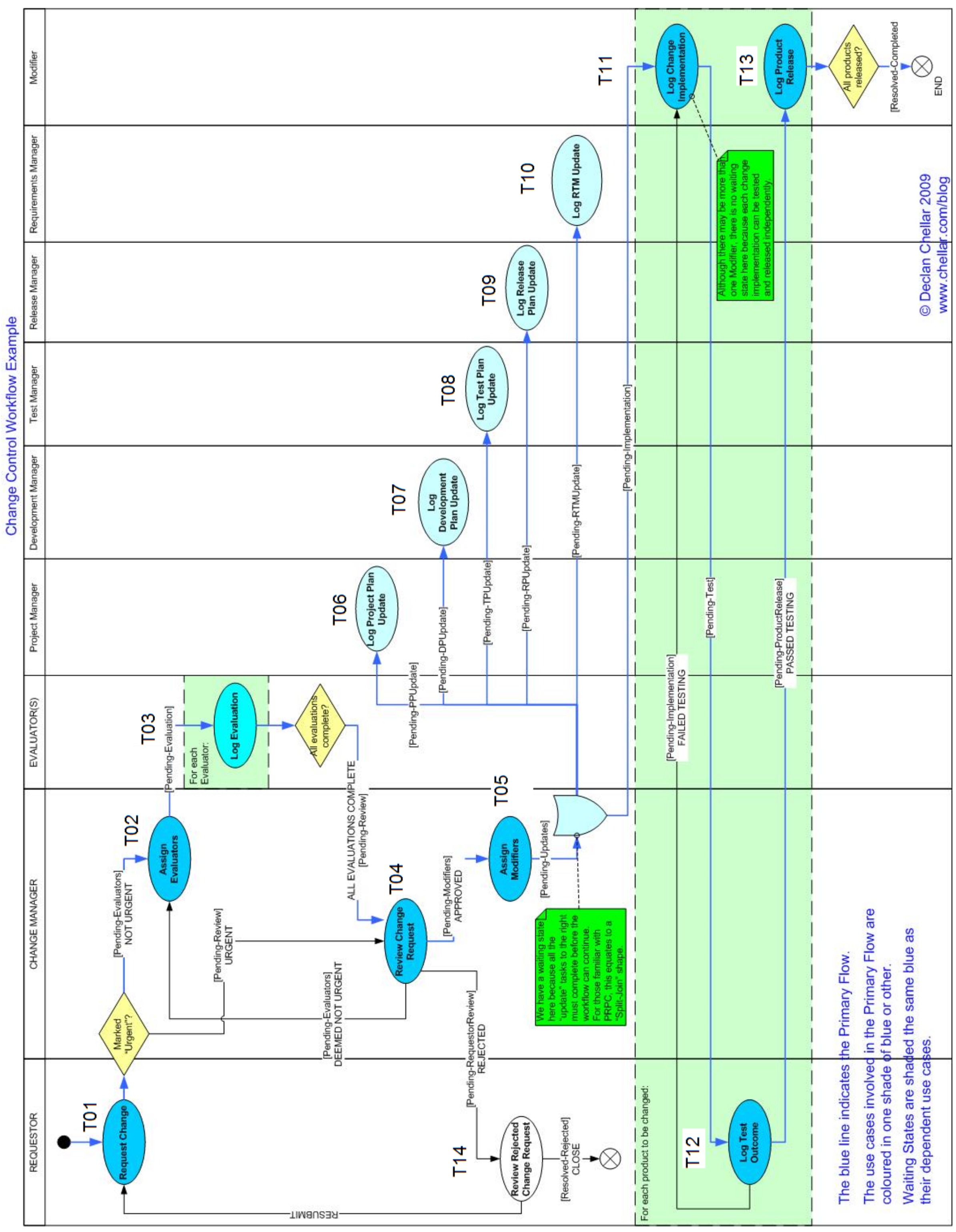

Change Control Workflow Example
REQUESTOR
CHANGE MANAGER
EVALUATOR(S)
Project Manager
Development Manager
Test Manager
Release Manager
Requirements Manager
Modifier
T01
Request Change
Marked "Urgent"?
[Pending-Evaluators] NOT URGENT
[Pending-Review] URGENT
T02
Assign Evaluators
[Pending-Evaluation]
T03
For each Evaluator:
Log Evaluation
All evaluations complete?
ALL EVALUATIONS COMPLETE [Pending-Review]
T04
Review Change Request
[Pending-Evaluators] DEEMED NOT URGENT
[Pending-Modifiers] APPROVED
[Pending-RequestorReview] REJECTED
T05
Assign Modifiers
[Pending-Updates]
We have a waiting state here because all the "update" tasks to the right must complete before the workflow can continue. For those familiar with PRPC, this equates to a "Split-Join" shape.
[Pending-PPUpdate]
T06
Log Project Plan Update
[Pending-DPUpdate]
T07
Log Development Plan Update
[Pending-TPUpdate]
T08
Log Test Plan Update
[Pending-RPUpdate]
T09
Log Release Plan Update
[Pending-RTMUpdate]
T10
Log RTM Update
[Pending-Implementation]
T11
Log Change Implementation
Although there may be more than one Modifier, there is no waiting state here because each change implementation can be tested and released independently.
[Pending-Test]
T12
Log Test Outcome
[Pending-Implementation] FAILED TESTING
[Pending-ProductRelease] PASSED TESTING
T13
Log Product Release
All products released?
[Resolved-Completed]
END
T14
Review Rejected Change Request
[Resolved-Rejected] CLOSE
RESUBMIT
For each product to be changed:
The blue line indicates the Primary Flow.
The use cases involved in the Primary Flow are coloured in one shade of blue or other.
Waiting States are shaded the same blue as their dependent use cases.
© Declan Chellar 2009
www.chellar.com/blog

Figure 3.2: Workflow example 2: A change control workflow example, Source: `http://www.chellar.com`. The task IDs T01-T13 have been added by the author of this thesis to later refer to the individual tasks.

to find similar tasks that could be covered by similar machine capabilities, here a sorting algorithm. We would therefore benefit from (a small set of) task categories to improve quick assessment of automation potential and knowledge reuse. As we will later see, cognitive task analysis already created such schemes.

Feedback (Ch 4) In technical IT management, in the end everything is about the managed objects: they are the primary information source, and they are the main entities to apply actions upon. In fact, from a technical perspective everything that does not involve the resources, and the services built on top of them, will not really matter to the customers who later run their services on the managed IT infrastructure. It's important to recognize that all actions performed on the managed entities will feed back to the administrators and management systems in some subsequent task. For example, if a certain resource has been procured, it is instantly clear, that the very same resource will have to be decommissioned at some later point in time. Or in a different example: If there are two administrator groups, and one configures the servers to maximize security, while a different one configures servers to maximize performance, then these two groups will interfere at some point in time. While the tasks may not interact, the configuration changes done by one group will feed back to the other group via the same managed infrastructure entitiy: the servers. It would be beneficial, if this feedback can be recognized in the task analysis results, because then we could more easily identify management collisions, such as multiple managers managing different aspects of the same managed entities but changing the same parameters.

Flexible human/machine task allocation (Ch 5): When IT management automation involves humans and management automation systems, then the tasks need to be assigned to either side. Members of both sides need to show up in the task analysis results. The results however need to be flexible regarding task allocation to ease the shift of tasks from the human side to the machine side, enabling automation or in the opposite direction to gain human control in case of automation failures.

Relevant machine capabilities (Ch 6): Automation needs machine capabilities to do tasks, that previously have been done by humans, or new tasks that arise and are instantly assigned to machines instead of bothering system administrators with yet another task. Machine capabilities will need to show up in systems designs to have a good clue on how the machine task is going to be implemented.

Modeling and tools (an issue across all chapters) Modeling an IT management automation concept will get complex despite all efforts to keep complexity low. If the results of the initial automation assessment were to be used as a foundation for later system design, the reuse of the models is beneficial. In all chapters we will demand the use of existing or upcoming standards and tools.

Results When we apply automation-oriented task analysis to the process descriptions in Fig. 3.1 and Fig. 3.2, we can see from the list of criteria, that the workflows, at least as stated there, have a couple of disadvantages. The shortcomings are detailed in Table 3.1.

Such workflows are quite detailed in what they show and require a lot of work to create, in fact, regarding some properties they do require more work than would be necessary for automation assessment. But concluding from that table and the criteria listed before, they miss the point of automation. They may be suitable for the documentation of human-performed processes, but they do not have desired automation-relevant properties like task categories, explicitly shown feedback, or variable human/machine allocation, and links to machine capabilities, such as AI methods. EIMA will use four steps to achieve a better result. This chapter covers the first one.

Table 3.1: Automation-oriented analysis of the two workflow examples

	Example 1: HR Onboarding process with link to IT operations	**Example 2: Change Control workflow in IT management**
Task control structure	(-) No control hierarchies stated. Workflow-like.	(-) No control hierarchies stated. Workflow-like.
Task decomposition structure	(+) Shows only the main control flow (with one exception). (+) Links to sub-items of work recursively.	(-) Primary control flow explicitly stated but cluttered by exceptions. (-) References to recursively defined tasks cannot be recognized.
Standard task categories	(-) Tasks are not grouped according to similarity.	(-) Tasks are not grouped according to similarity.
Feedback	(-) The feedback via IT resources can not be seen.	(-) The feedback via IT resources can not be seen.
Human/machine task allocation	(-) Fix humans (instead of roles) are assigned to most tasks, even with a photo, which eases human recognition but makes automation abstraction harder. (-) Some tasks are without assignment. Here, responsibilities are only included by department name for the swim lanes.	(+) All tasks are assigned to roles via swim lanes. (-) It is unclear, whether these are human or machine roles. The semantics of capitalization is unclear. Responsibilities are unclear.
Relevant machine capabilities	(-) Shows some necessary IT actions, but not machine capabilities. At least, all machine actions do have a simple gearwheel icon, but not a machine component name.	(-) Does not link machine capabilities to the tasks.
Modeling and tools	(o) With Appian BPM Suite, a BPM tool is used, which does offer support for workflow simulation and workflow automation via a set of predefined actions.	(-) The concrete used tool is not visible. But the graphical complexity allows to assume that a drawing tool was used.

3.2 Requirements on task analysis in EIMA

The following three requirements will be the revelant aspects for task analysis in this thesis.

3.2.1 R1.1 Common automation-oriented IT management task structure

IT solution builders and system administrators spend too much effort on automation. Often, each person or company builds their own automation solution despite significant similarities of his scenario with other scenarios. This starts in task analysis, that can result in different sets of tasks with different names but essentially much overlap.

To overcome this situation, we need to have a general structure of *common* resource-related tasks in IT management. While of course each set of managed resources is different, we need to come up with a list of common tasks. We could then refer to the individual processes and tasks using standard names that have clear semantics to a number of people. This is necessary for communication between system administrators and systems designers and developers. Later, IT administrators can understand, which person will be supported by automation or responsible in what way. By doing so, referring to these tasks in system documentation eases the adoption by a data center operator using the management automation system. At the same time, if automation relates to formerly human-performed tasks, we can assume them to be relevant.

Main criteria can directly be derived from the terms: the task structure needs to be made for resource-related IT management, and needs to be commonly, accepted especially at the side of IT administrators.

3.2.2 R1.2 Task decomposition and control hierarchies

The goal of task analysis in EIMA is to find the opportunities for automation in resource-related IT management. Aspects in task analysis relevant for automation are:

1. Role *control* structure: Automation is about delegating control between human roles and machine roles. We need to remain a grip on responsibilities and hierarchies to know who controls what. Therefore, when analyzing tasks, we need information on hierarchical control.
2. Task *decomposition* structure: It is clear upfront, that tasks can be described at different levels of detail. With a good approach, this should result in a structure that uses one of the higher levels of abstraction to reduce specification and analysis overhead. To go into more detail, it should allow a recursive decomposition to refine all of the tasks and show their internals.

To find a good approach, we first need an overview on automation-oriented methods for task analysis. Probably, we can then build upon existing knowledge and schemes. Main criteria for a good approach in automation-oriented task analysis is a background, where many automation projects have been successfully performed along the lines of the named approach, in terms of task decomposition and control hierarchies.

3.2.3 R1.3 Automation-oriented task descriptions

Adopting the engineering approach, it is not sufficient to just describe tasks in long verbal specifications and numerous drawings, both made to be read and interpreted by humans. Instead, tasks will need to be represented in tools for task analysis and process description in a formal language whenever possible. For this purpose, we need a model for tasks and processes. It must contain enough metadata to capture the essential results of automation-oriented task analysis for each task, as well as the relationships of tasks. In the later stages, this information on task anaylsis results will need to be linked to the loops, allocations, and implementation methods. Main criteria is an appropriate level of fomality matches the needs of automation assessment: It needs to be formal enough to not waste time in semantic and syntactic differences, but it needs to be informal enough so not become a collection of mathmatical formulae but instead an actual means for communication and modeling.

3.3 Terms

As this thesis involves cross-domain research, vocabulary is both less important and more important. Less important, because we need to recognize related concepts or entities in different domains, even when they are named differently. More important, because we need to state what we mean when using certain words to prevent that readers with a different domain background might associate words with other meanings. For this purpose, we provide a description list of commonly used words in this chapter to explain their implicit meaning within this thesis:

task A task is a (usually assigned) piece of work that either needs to be finished within a certain time or is stated as an on-going one to keep or reach a certain state. By performing a task, a certain output is created from a certain input. Tasks can be performed by human administrators or management software. In SysML, the tasks are called "actions".

We only deal with tasks involving the processing of information. Tasks requiring physical work are out of the scope of this thesis as they are relatively uncommon in IT management, even if there are a few exceptions such as cabling work, replacing removable media, disassembling, fixing and reassembling hardware, installing components into racks. Still, the majority of the tasks is of a cognitive nature and purely involves information processing: editing files, general file operations, editing databases, using IT management systems with a CLI or GUI, communicating via e-mail or phone or other messaging systems, starting/stopping services.

task analysis The goal of task analysis is to capture a model of IT management tasks that shows their relationships and their properties relevant for IT management automation.

Task analysis can basically be performed at two points in time: 1) predictively when a new IT system is planned and future IT management tasks are predicted for this particular set of resources or 2) reactively at systems operation time analyzing the automation potential in currently performed IT management for a certain set of IT resources.

process A process refers to a certain defined sequence of tasks. We use processes for a rough categorization of work in IT management, and tasks and sub-tasks for refinement. In SysML, processes are called "activities".

resource-related IT management IT management combines features of technical systems management and features of business management. The scope of this thesis is on the technical systems management part, interfacing to the actual IT systems. The explanations are less applicable to the business side of IT management. Still, we keep the links to this upper management level.

human administrator A human IT administrator is a person that manages a certain set of IT resources by monitoring them and eventually by reconfiguring them to better achieve a certain goal.

management automation system A management automation system is a technical system that can either completely take over IT management tasks or it can support a human administrator by taking over a couple of sub-tasks.

3.4 Existing knowledge and concepts from current IT management automation

This section presents the findings in IT management with respect to the requirements stated before. It serves as an overview of the state of the art and current practice in this domain.

3.4.1 R1.1 Common task structure of resource-related tasks in IT management.

IT management is an application domain with at least three viewpoints: a technical one (at systems level by resource, or at systems level by functional area), an organizational one (IT process management), and an economic one (not considered here in detail with the exception of CobiT). Thus, several approaches have been taken to structure tasks in IT management, which often take one of these viewpoints. The phrase task structure will be abbreviated as "TS" in some places in this section.

Task structure at the systems management level by resource

As we are looking for an analysis of *resource-related* tasks, we could of course start with the actual resources. The resource-bound view is a traditional one, where system administrators are associated with resources, and then are assigned all tasks that are associated with a certain mostly physical resource. As an example, a server administrator would care about procurement, setup, configuration, performance, faults, security and so on, all for servers.

This way, systems management in IT management has been strucutured into processes/tasks in such a way:

- server management: management of servers (typically OS and basic services)
- storage management: disk arrays, tape libraries, archives, partitioning, backup and restore
- network management: network components such as switches, routers, gateways, firewalls, network services like DNS
- data center management: data-center equipment like power supply, cooling, racks, cable management
- printer management
- applications management: typically broken down into the applications, such as database, groupware, web server, ERP, etc.
- software license management
- and many more

About each of these topics, there is much documentation, most of it vendor-specific. A treatment of tasks at this level would be very resource-specific. Its value for a general method such as EIMA would be limited, so this way is not taken here.

The following two books show well this classical view in IT management.

TS in Lyke, IT automation - The quest for lights-out management, 2000 [Lyk00] tackles "lights out management", describing methods for IT management automation including project and migration management as well as organizational aspects. The term "lights out" is typically used to emphasize, that all IT resources are essentially managed either by humans remotely or by automated procedures. As it's humans, which need light in a server room or data center room, the light can be switched off as soon as no administrators are inside the actual housing rooms anymore. During the design of such data centers, it must be preplanned that as many management interactions as possible can be performed remotely instead of walking to the resources. This shall create a more human-friendly workplace for administrators, who no longer are exposed to noise, heat/cold by machines/cooling systems, or the danger of fire retardant systems. At the same time, however, lights out management is a major step towards automation or remote management as the physical interactions are reduced to a minimum, which can make a certain number of local administrators redundant.

The book describes in a project-oriented manner how automation in data centers can be tackled. Less emphasis is put on the methods but instead the author proposes a number of hands-on step-by-step guides to organize the automation of frequent activities in data centers. The approach as human-oriented automation project has a main focus on considerations about staffing, project planning, project budget and deployment, instead of a main focus on actual enabling technologies.

Instead of a pure resource-related point of view, Lyke mixes resource-related, function-related and service-related management tasks. Thus, he recognizes that in addition to the resource-centric view a view on functional grouping and service managemnet is important in IT management.

TS in Limoncelli, The practice of system administration, 2002 This contribution [Lim02] covers the daily work of system administrators. It is written by system administrators, who document a number of (their) best practices and general knowledge gained during (their) professional work. Most interestingly, this book describes how system administrators at the same time like and dislike their work being automated. The table of contents is organized in mostly resource-specific chapters.

Task structure at the systems management level by functional area

A number of intiatives recognized that many of the tasks on individual resources can be aggregated into functional groups. These functional groups then serve as main categories for management tasks.

TS in OSI Systems Management Functional Areas: FCAPS, 1980's As the first standardization body, the International Organization for Standardization (ISO) came up with a model for the management plane in IT systems management. To achieve this, the committee had to implicitly do task analysis. They grouped their findings into functional groups documented in ISO 10040, the Open Systems Interconnection (OSI) Systems Management Overview (SMO) standard, defining five systems management functional areas (SMFA's), originally adopted mainly for network management:

- fault management: fault detection, fault analysis, fault correction, fault prevention
- configuration management: configuration, versioning and documentation, change management, asset management, provisioning
- accounting management: IT resource use metering and accounting
- performance management: performance measurements, performance analysis, performance tuning

- security management: incident detection, incident analysis, incident correction, incident prevention

As a group, these five areas are commonly referred to as "FCAPS". They are all covered by the Common Management Innformation Protocol (CMIP), which today is still used in the telecommunications industry, while the upcoming Internet had developed the Simple Network Management Protocol (SNMP) as a light-weight alternative.

As an addition to the actual standard documents, see [HAN99] to describe concepts of IT management and management architectures, with a special focus on integrated management. The aspect of IT management automation is only briefly mentioned in this book in terms of management delegation between a management station and management agents that is also part of OSI systems management. Still, it is an excellent source for knowledge of IT management tasks grouped along this function-related view.

TS in IBM Autonomic Computing Initiative: Self-CHOP, 2001 The Autonomic Computing Initiative had a major impact on research with its idea of self-managing IT resources. For this idea, its visionaries, see [GC03, IBM06], also needed a rough grouping of self-management functions as a replacement or support for human IT management tasks. They came up with the task structure:

- self-configuration: reduce configuration interactions performed by humans to the absolute minimum
- self-healing: mask and handle internal faults so that they do not lead to externally observable malfunctions
- self-optimization: approach an implicitly or explicitly given performance goal by tuning parameters
- self-protection: protect oneself from unwanted change

These four areas are abbreviatedly called self-CHOP properties according to their initial letters. Interestingly, these four can be associated with four out of the five FCAPS management functional areas: self-healing being automated fault management, self-configuration being automated configuration management, self-optimization being automated performance management, and self-protection as automated (preventive) security management. Already at this point we can recognize, that the multitude of task names is probably larger than the the diversity of actual management tasks.

TS in Organic Computing: ACI+Self-Organization, 2004 Organic Computing (OC) is a mainly German initiative inspired by the ACI, but with an emphasis on biological schemes as a source of inspiration for multi-agent systems and the inclusion of different application areas than just IT management. Researchers include for example applications in embedded systems in cars, traffic control, and the power grid. OC adds to the four self-properties from the ACI the function of self-organization of the potentially large number of agents with the aim of decentralized knowledge, decentralized control, emergent structures and adaptive behavior [HWM07]. This way, Organic Computing includes the task to deal with the growing scale of IT resources. Instead of centralization and human-defined hierarchical resource structures, which are known for their good control, low resource consumption, policy enforcement and fault diagnosis, they argue that self-organizing structures may add overhead, but will increase availability, adaptability and scalability.

Task structure at the process management level

The traditional resource-centric and functional area-oriented views are more and more superimposed and supplemented by a more customer-oriented *service* view. The need for this service view arised, when customers were less able and willing to specify their needs in terms of actual IT resources, but instead wanted to specify quality levels, as well as feature requests and incidents, at their level of understanding of the actual services.

In order to create this more business-oriented view on IT management, IT resources are abstracted away behind IT services. IT service management took over the idea of functional grouping of tasks, and created IT service management processes to bring the service-oriented view into the organization of an IT service provider. ITSM deliberately gives up the tasks being split by resource and instead creates all processes to indirectly cover all IT resources. This way, the tasks in IT service management more and more veer away from the actual resources and build a link to business management of an IT service provider. As such, a different break-down of IT management tasks is created. To give an example: Now, a service desk agent and an incident manager together need to deal with all incidents regardless of the resource. They will take care of the incident to be classified, eventually solved with a solution from a knowledge base, otherwise assigned to specialists in the operational departments, and resolved or worked around quickly. Some of the ITSM processes have a good potential to be automated in management automation systems closely associated with the actual resources.

The following paragraphs will give an overview on task analysis according to a selection of the most relevant standards in ITSM.

TS in ISO 20000 (2005) and ITIL V3 (2007) ISO 20.000 and the IT Infrastructure Library (ITIL) V3 are probably the most prominent and relevant documents to structure tasks in IT service management today. ISO 20.000 is an internationally standardized version of the processes defined in ITIL V2 and condenses the lengthy descriptions in the ITIL best practice descriptions to a short certifiable standard. It is a welcome foundation for the definition of terms in IT service management, common processes and many of their properties.

ITIL V3 as a best practice framework reorganizes the overall content of the previous ITIL publications along a slightly different structure and includes recommendations on service strategy and service design. Overall, the global structure looks like this:

- Strategy Generation: Financial Management, Service Portfolio Management, Demand Management, Return on Investment
- Service Design: Service Level Management, Service Catalogue Management, Information Security Management, Supplier Management, IT Service Continuity Management, Availability Management, Capacity Management
- Service Transition: Transition Planning and Support, Change Management, Release and Deployment Management, Service Asset and Configuration Management, Service Validation and Testing, Evaluation, Knowledge Management
- Service Operation: Request Fulfillment, Incident Management, Problem Management, Event Management, Access Management, and four Functions: Service Desk, Technical Management, IT Operations Management, Application Management
- Continual Service Improvement: The 7-Step Improvement Process, Service Reporting, Measurement, Business Questions for CSI, Return on Investment for CSI

From this small overview, we can see that the main focus within ITIL V3 for this thesis will likely be service design and service operation. Therefore, the transition from ITIL V2 to V3 has not brought many new automation-relevant features and we will concentrate on a certain subset of ITIL V3, where automation is well applicable. This set will later be introduced in this thesis and called AutoITIL, which is closer to ISO 20.000 (and therefore ITIL V2) than ITIL V3.

TS in CobiT 4.1 (2007) Control Objectives for Information and Related Technology (CobiT), currently in version 4.1, is an IT governance framework. It is designed to serve as an interface between an ITSM framework such as ISO 20000 or ITIL and the actual business objectives management.

Regarding task analysis, CobiT comes up with a list of 34 IT-controlling related processes in four groups:

- Plan and Organize: PO1 Define a strategic IT plan. PO2 Define the information architecture. PO3 Determine technological direction. PO4 Define the IT processes, organization and relationships. PO5 Manage the IT investment. PO6 Communicate management aims and direction. PO7 Manage IT human resources. PO8 Manage quality. PO9 Assess and manage IT risks. PO10 Manage projects.
- Monitor and Evaluate: ME1 Monitor and evaluate IT performance. ME2 Monitor and evaluate internal control. ME3 Ensure compliance with external requirements. ME4 Provide IT governance.
- Acquire and Implement: AI1 Identify automated solutions. AI2 Acquire and maintain application software. AI3 Acquire and maintain technology infrastructure. AI4 Enable operation and use. AI5 Procure IT resources. AI6 Manage changes. AI7 Install and accredit solutions and changes
- Deliver and Support: DS1 Define and manage service levels. DS2 Manage third-party services. DS3 Manage performance and capacity. DS4 Ensure continuous service. DS5 Ensure systems security. DS6 Identify and allocate costs. DS7 Educate and train users. DS8 Manage service desk and incidents. DS9 Manage the configuration. DS10 Manage problems. DS11 Manage data. DS12 Manage the physical environment. DS13 Manage operations.

For each process, CobiT does not define what to do at the resource or process level, but what to achieve, i.e. control objectives regarding effectiveness, efficiency, confidentiality, integrity, availability, compliance, and reliability, and how to measure the current maturity level of a process.

With its scope on business, CobiT is a good orientation for general business-level goals given to management automation systems, but by design it lacks a link to technology. Automation is simply assumed to exist as can be seen explicitly in process “A1: Identify automated solutions”. However we think, that the role of automation in CobiT can well get much more important. As advanced IT management automation systems will need to be goal-driven and in need of utility functions for the control objectives listed above, CobiT is a good input as the automation matures. Once automation gets capable to translate goals at this high level of abstraction into machine actions, it is very welcome to have a widely accepted document to specify such high-level goals and metrics.

TS in other standards A few other standards are less important within this thesis but added here for the reason of completeness and a broad coverage.

The eSourcing Capability Model for Service Providers (eSCM-SP) is a standard for sourcing of sub-services, created with strong influence by the Capability Maturity Model (CMM). Regarding task analysis, it consists of 84 best practices, grouped in the following catagories: Knowledge Management, People Management, Performance Management, Relationship Management, Technology Management, Threat Management, Contracting Management, Service Design and Development, Service Transfer (initiation or transition), Service Delivery.

The Enhanced Telecom Operations Map (eTOM) defines processes and tasks in a telecommunication operator's enviroment. Together with the definition of the Shared Information and Data Model (SID) an Operations Support System framework named NGOSS was derived. As a telecommunications provider has a slightly different task and process landscape, despite some overlap though, the eTOM approach has not widely been taken over in IT management.

The upcoming ISO 27.000 series standards will cover information security management in-depth.

Consolidation of knowledge

Structure is the main answer to problems of scale and heterogeneity, both if which contribute to complexity. Therefore, a common overall structure of IT management tasks to apply automation to would help to better understand the groups of tasks in IT management. As was shown before, IT management has already been analyzed and structured in a number of ways.

We now need to sort the tasks, align the different frameworks, and to recognize where tasks in the different domains overlap or complement each other. The result will make system analysts or designers of IT management automation systems aware of the wide range of activities. As we believe that ITIL is among the most influential process frameworks, we will create a subset of ITIL processes which seems to be particularly well suited to be supported by automation.

Along this subset, called AutoITIL processes, a task list for a certain IT solution can then be checked for completeness to a certain extent. By aligning automation design to ITIL processes, it will clearly improve system administrator acceptance, as the management automation system is built on the vocabulary and overall spirit of ITIL and therefore easier to understand than some totally new, machine-focused protocol or task set.

To subsume the findings, we need to align four understandings of tasks in IT management: ad hoc IT management, FCAPS and Self-CHOP along functional areas, and ITSM process-related frameworks like ITIL. The different task/process frameworks match quite well, as can be seen in Table 3.2.

Selection of one naming scheme: AutoITIL A common problem of IT management automation is the variety in syntax and semantics of names of tasks. As stated in the "Terms" section of this chapter, cross-domain work can suffer from many words meaning similar things. Thus, in Table 3.3 we have aligned task names/process names along their semantics.

It would much help, if IT management automation would have a certain reference set with standard names given to typical IT management tasks, that are to be automated. It provides a common structure to align human IT management processes/tasks to. It is important however, that this reference set is not yet another process framework, but instead is understood with the current knowledge of system administrators. Thus, in order to reduce the variety of syntax, now we need to choose one column from Table 3.3.

They are close, so let's take the newer standard, and, within this standard, choose the tasks that are close to resources and promise a good potential for automation

Table 3.2: Alignment of abstract tasks in IT management with Autonomic Computing added.

Ad hoc management	**OSI managent SMFA's**	**ITIL V2**	**Autonomic Computing**
ad hoc fault analysis and correction	Fault Mgmt	Incident Mgmt, Problem Mgmt, IT Service Continuity Mgmt	Self-Healing
ad hoc configuration	Configuration Mgmt	Configuration Mgmt, Change Mgmt	Self-Configuration
flat pricing	Accounting Mgmt	Service Level Mgmt	none
best effort performance	Performance Mgmt	Service Level Mgmt, Capacity Mgmt, Availability Mgmt	Self-Optimization
ad hoc reaction to security incidents	Security Mgmt	Security Mgmt	Self-Protection

A quick evaluation, that adds the resource-oriented view and CobiT to the columns of Table 3.3, results in:

- The names along ad hoc management are not suitable, of course, for engineered IT management automation. It is adhoc management and adhoc automation, that EIMA wants to avoid and replace.
- Data centers are not organized along the FCAPS model, e.g there is no entity whose role is connected to fault management in general. So these terms are irrelevant in common IT management practice.
- System administrators can be assumed to be familiar with ITIL V2/ISO 20.000. A certain obstacle to automation is the large change and enhancements brought by ITIL V3, as it adds a lot more mainly on the side of service design and overall business management, such as detailed improvement procedures. So, while ITIL itself keeps growing and adding more and more processes, the set of ITIL processes listed in Table 3.3 focuses on a subset directly relevant for automation, and is therefore more accessible and regarded as less complex.
- The self-CHOP terms are mis-leading and their binary semantics makes them appear like magic. For this reason this thesis avoids them altogether, if they are not needed to refer to other documents. The names were potentially used in advertising but never in data centers. In fact, "self"-management in terms of autonomic managers embedded into the resources does hardly matter to administrators. What matters is the automation itself and an off-load of work. The actual physical distribution of the automation routines is of minor concern.
- Names of sub-domains of systems engineering would essentially be a good candidate. We have refrained from using them as they put their emphasis on systems design instead of systems operation time. This could defer system administrators from accepting the results of the EIMA method.
- CobiT terms and process names would be too high-level for the automation purpose. Nevertheless, it provides a nice list of goals to achieve, even if stated in an abstract manner.
- Resource-oriented split-up of tasks: This would also be possible, but as we deal with resource

pools, we can assume that task analysis for each individual resource has been performed before. In addition, a split of tasks by resource can be derived without much work from a given IT solution. However, the solution was built for an overall service/services that no single component alone can achieve.

As a result of this quick evaluation, we will adopt a task/process scheme related to ITIL/ISO 20.000. Among the ones presented above, it seems to be the only scheme, that is both resource-related in part, and relevant to actual system administrators in their practice. To constrain the number of processes, the other domains have helped to find the most relevant processes for automation within ITIL. Also now (with help of the table) we can relate ITIL tasks to more technical matters, and most essential to the systems engineering sub-domains.

To reduce the heterogeneity of terms, in the rest of this thesis we will only use the process names from the ITIL column of Table 3.3. We will call this subset the "AutoITIL processes", or in short just "AutoITIL".

In the rest of this thesis, we will now assume a cooperation of AutoITIL processes (run essentially by management automation systems) and ITIL processes (run essentially by humans). In most of the cases, it will be clear, whether with e.g. "incident management" we mean the AutoITIL one, or the human one. Nevertheless, to make things explicit, we will add the prefix "auto" to process names, when we mean an AutoITIL process, such as in: "auto incident management". When we mean the human ITIL processes, we will add no prefix.

Intended use of AutoITIL The AutoITIL processes will be the basis for IT management automation in the rest of this thesis. However, one important aspect will change when compared to ITIL processes. ITIL processes have often been turned into or understood to be transformed into workflows. For people, this may be a good description. For automation, though, we will change this workflow-related thinking to a control-loop structure, and the whole automation to a hierarchical control system. In such a system, all "workflows" are modeled as feedback control loops, related to cognitive loops. See the next chapter for a closer investigation of loops.

As an outlook, and not covered in this thesis, the AutoITIL process set could of course be extended with AutoITIL protocols. The idea is that these protocols are easy to understand and debug for a system administrator later, as the whole underlying concept is based on ITIL, and therefore his understanding of work. If a management protocol was aligned to ITIL, all messages, events, parameters, control structures, roles and so on do have an immediate meaning to him. The major benefit is, that this protocol is easy to understand for administrators. This is important, as system administrators may have to investigate, troubleshoot or even work around AutoITIL automation routines themselves.

3.4.2 R1.2 Task decomposition and control hierarchies

As many others, ACI or organic computing do not refer to any task analysis methods in particular. Regarding control hierarchies, at least the ACI includes a graphics about a imaginary overall control system hierarchy, which is called AC reference archicture, see Fig, 3.3.

The most interesting aspect here is, that for the self-configuration, -healing, -optimzation, -protection properties they use equally structured "intelligent control loops". We will investigate the loops themselves in Ch. 4, the details on the links between these elements remain vague however.

OSI management and FCAPS do have a management by delegation concept, but task control or task decomposition are not covered. The ITSM frameworks provide long verbal descriptions on many

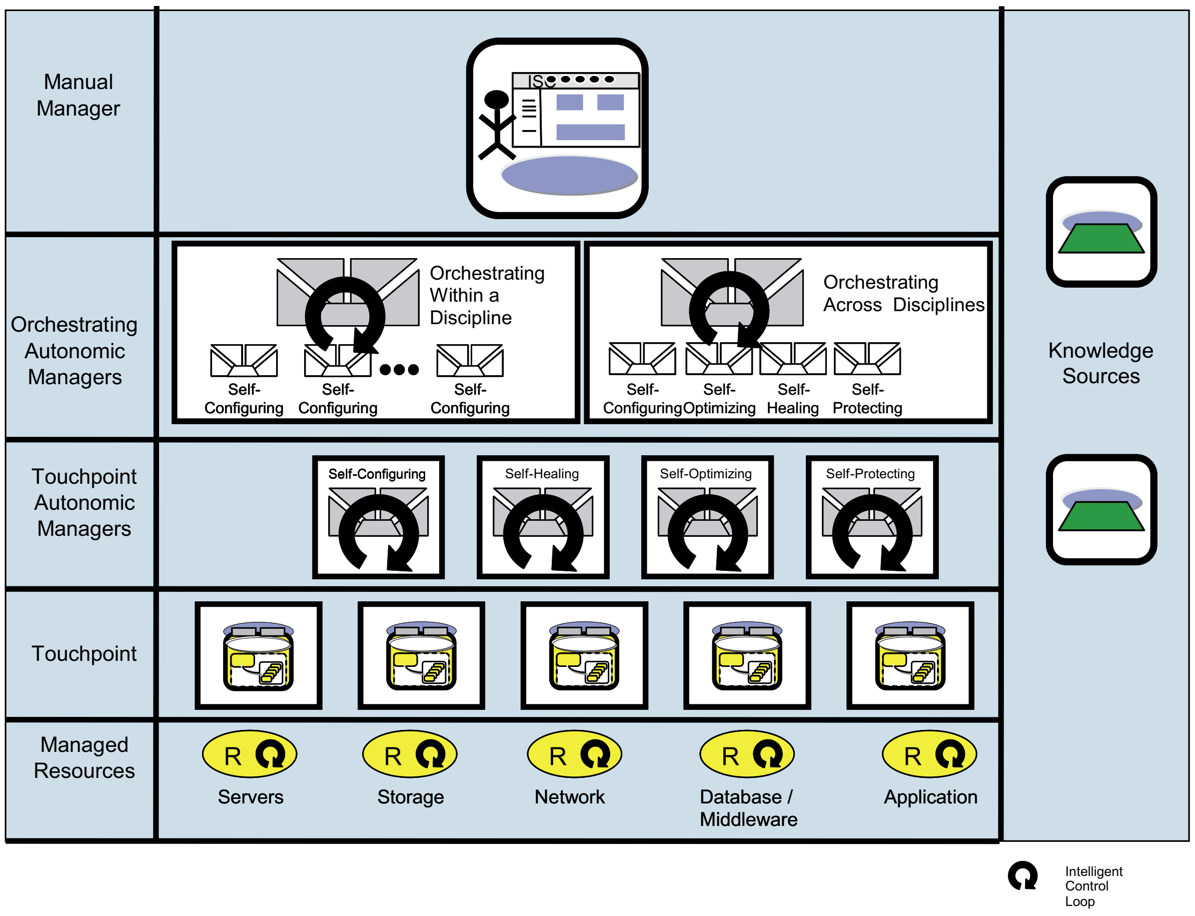


Figure 3.3: This illustration on the “Autonomic Computing reference architecture” (see [IBM06]) shows ACI’s understanding of task control hierarchy and task decomposition.

particular aspects of the processes, but lack a true process refinement. As a control instance across ITSM processes, there is no concept besides general upper business management.

In general, task analysis as an actual method in terms of task decomposition and control hierarchies is not well investigated in IT management. As an example, we show the deficits along two publications, that each takes a methodological approach, but are either too generic or they mix multiple of EIMA's concepts.

In their publication "A systemic approach to autonomic systems engineering" [BSTB+05] Bustard et al. propose an approach to autonomic systems engineering by applying Beer's viable systems model, that builds upon abstract entities such as operations, coordination, control, audit, intelligence, and policy. This model has been applied to a whole range of social systems. While the aforementioned entities are a part of most organizations, this task analysis does not seem to be very constructive for IT management automation. During the course of their argumentation, they derive a hierarchical conceptual model for enabling the technology perspective of an organization, see Fig. 3.4. As can be seen from the figure, this structure is not of immediate utility either.

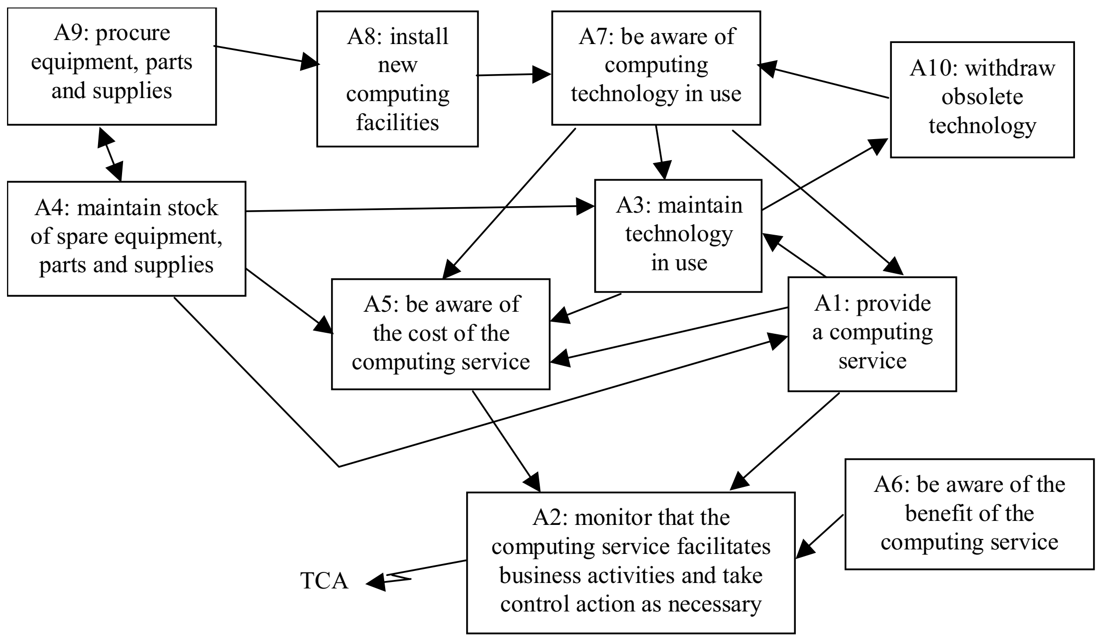

A9: procure equipment, parts and supplies
A8: install new computing facilities
A7: be aware of computing technology in use
A10: withdraw obsolete technology
A4: maintain stock of spare equipment, parts and supplies
A3: maintain technology in use
A5: be aware of the cost of the computing service
A1: provide a computing service
A6: be aware of the benefit of the computing service
A2: monitor that the computing service facilitates business activities and take control action as necessary
TCA

Figure 3.4: Conceptual Model for Enabling Technology Perspective of an Organization (see [BSTB+05])

In "Towards requirements driven autonomic systems design" [LLMY05] Lapouchnian et al. work towards requirements-driven autonomic systems design. They use a method from software engineering to first create a goal model as a means of communication and then to derive a feature model (see Fig. 3.5). In both of them, they include variation points (VP) to later come up with product variations. As can be seen from these figures, they create a structure of goals, tasks, and tasks allocation. With variations points, they also create a rough equivalent to a differing degree of automation in realizations of the modeled system. In essence, we can see though, that their approach mixes a number of steps, that EIMA wants to keep separate from each other for the reason of separation of concerns.

Even though these two publications only provide a glimpse at this field, we can conclude that in IT management task analysis remains more of an intuitionally performed activity, than some standard task. We dare to conclude, that we will have to rely on other domains to have come up with a good method for task analysis.

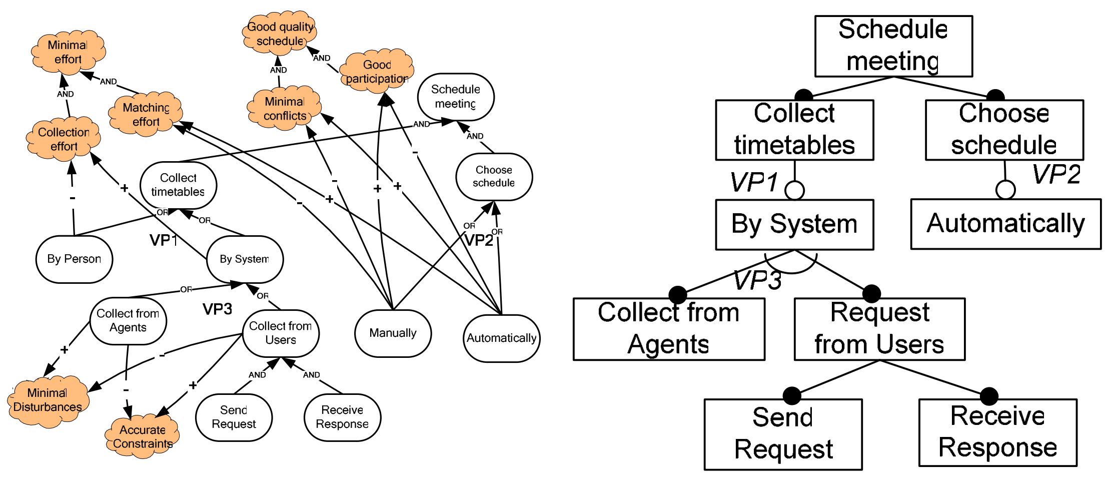


Figure 3.5: Goal model and derived feature model for a meeting scheduler (see [LLMY05]). The authors mix task analysis and function allocation here.

3.4.3 R1.3 Automation-oriented task descriptions

Even if task analysis is not well established as a common method in IT management, there are some (semi-)formal representations of processes, workflows, and tasks in IT management. So, based on different methods different results have been achieved with a differing information and data model to represent information and save results persistently.

In contrast, *resource* modeling formats, such as Management Information Base (MIB), Common Information Model (CIM) and the Resource Description Framework (RDF) are already common in IT management and have achieved much more acceptance as of today.

Task models/descriptions in ACI For task descriptions, the Autonomic Computing Initiative did not propose or endorse any particular standard.

Task models/descriptions in BPMN + BPEL Mainly in the scope of IT *service* management (ITSM), research includes task decomposition/modeling of those parts of ITSM processes that are performed by humans.

Taking over evolving standards in business processes, verbal descriptions in ITSM reference processes in frameworks like ITIL and ISO 20.000 have been the basis for modeling. As an example, [Cla06] came up with models ITIL change management in the Business Process Modeling Notation (BPMN), a graphical representation of workflows.

The following standards are close representations of data formats for the process content in BPMN: BPEL, XPDL. Nevertheless, BPMN tools typically use an additional proprietary format to capture all graphics specifics. In essence, the current BPEL/BPMN or XPDL/BPMN converters often have difficulties with the full syntax and semantics, as BPMN and BPEL/XPDL have been designed for different purposes.

Also industry has made efforts to transform ITIL into a machine-interpretable format, see for example IBM's software and tools around the IBM Tivoli Unified Process, with visual workflow diagrams modeled in IBM's Rational Method Composer. It allows data exchange with other IBM products, but limited support for other standards. We can assume, that other ITSM tool vendors have created similar process templates, based on their interpretation of the reference framework. That is not an obstacle, as a customer is expected to customize the process templates to fit his own needs anyway, but the individual interpretations may differ widely.

Task models/descriptions by Petri nets In academia, there are proposals to use modeling techniques, such as modeling IT management as actions in a state space. Then, activities can be modeled in Petri nets, and a controller can be used to minimize the difference of the observed state to the desired state, as proposed by Graupner in [GCC07]. Petri nets are a good formal model with a number of tools for validation, dead lock analysis etc., but they seem inappropriate as a communication scheme between system administrators and system developers, especially once extensions like coloured Petri nets, Petri nets with many tokens, stateful tokens, or probabilistics are added. In essence, this approch makes automation generic and given enough models would enable automation. It does not seem to be a good approach for task analysis in the scope of this thesis though, due to its necessarily large amount of modeling at a low level of abstraction.

Task models/descriptions in proprietary formats According to a Gartner report on run-book automation tools [Wil07], there are already a range of tools targeted at data-center automation. With the tools named as *run-book automation* tools, they are relatively simple and based on schedules that run external scripts. As such, they can be understood as a feature-enhanced version of *cron* jobs, adding execution monitoring and logging, replay, simple control structures, and action blocks for the most common tasks on resources, all in a common graphical interface. They are mostly tools with support for visual scripting or workflows to replace glue code for other scripts. Regarding task modeling and task decription, if such automation tools are used in IT management, they rely on their own proprietary representation and visualization of information and data.

In essence, run-book automation tools take a different view than (human) task analysis, as they mainly capture the tasks that are scripted already and provide an integration interface for them. To ease process composition, the automated scripts are often presented with a work-flow related visualization. Eventually, users will recognize that tasks for humans are not an entirely different thing than tasks for machines.

From the point of view of the author of this thesis, a common short-coming of run-book automation tools is this workflow view instead of a control loop view or the breakdown into the cognitive phases. Another concern is their lack of feedback and lack of integration with lower management systems. Thus, the value of runbook automation tools will much enhance, when they are integrated with monitoring and control tools for the actual IT services and IT resources within their topology. Only such feedback makes the system administrator aware of the real effects of the scheduled automation tasks. The next step (at a certain future point in time) would then be to simulate the effects of new jobs on the IT resource landscape or to predict states from the current one into the near future like weather forecasts.

3.4.4 Summary

There have been multiple approaches to structure tasks in IT management or its sub-domains. Today it seems that ISO 20.000 and ITIL V2 and V3 are the most prominent process frameworks and that CobiT can provide an additional controlling-oriented view. However, regarding the methodology of their task analysis and task descriptions, all current standards rely on textual descriptions. The standards on processes have not come up with more formal representations of processes in workflow languages or any other machine-interpretable format, even though some companies and academia have started to model them. All in all, a general automation-oriented task analysis has so far not been performed, with the combination of eTOM, SID and NGOSS being the exception to this rule, but in the scope of a slightly different domain.

3.5 Existing knowledge and concepts from other domains

Automation-oriented task analysis has been weak in IT management and made little use of knowledge in other domains. Instead, other domains have come up with remarkable automation results, such as autopilots, flight control systems or factory automation systems, built up enormous expertise in designing and operating these systems and have overcome many of the problems raised in the use of automation systems. This section looks at what other domains can contribute and offer us in terms of concepts for task analysis transferable to IT management automation.

3.5.1 Introduction to the domains

From the name of the EIMA method,“Engineered IT management automation”, the most relevant inspiration domains can be derived.

Human factors engineering All artificially created systems, and therefore IT systems, have to meet the needs of their human operators, users, and administrators. Therefore, designers include human factors into their work, which created the domain of human factors engineering or ergonomics. It is a domain with influences from medicine, technology, psychology.

Systems engineering Organizations like space agencies and the military, as well as governments have ordered systems with firm requirements on critical systems, which has lead to the development of systems analysis, systems design, and systems engineering especially for control systems.

Systems engineering techniques are used in many complex projects: from spacecrafts to chip design, from robotics to creating large software products to building bridges. Systems engineering uses a host of tools that include modeling and simulation, requirements analysis, and scheduling to manage complexity. The NASA Systems engineering handbook defined systems engineering as:

> Systems engineering is a robust approach to the design, creation, and operation of systems. In simple terms, the approach consists of identification and quantification of system goals, creation of alternative systems design concepts, performance of design trades, selection and implementation of the best design, verification that the design is properly built and integrated, and post-implementation assessment of how well the system meets (or met) the goals. (NASA Systems engineering handbook)

For analysis-related matters, systems engineering is typically refined to systems analysis. Systems analysis is the sub-domain of systems engineering dealing with tasks like the description and decomposition of systems, often prior to their automation as computer systems, and the interactions within those systems.

Klaus Krippendorff defines in an unpublished Dictionary of Cybernetics (1986):

> Systems analysis is the diagnosis formulation, and solution of problems that arise out of the complex forms of interaction in systems, from hardware to corporations, that exist or are conceived to accomplish one or more specific objectives. Systems analysis provides

a variety of analytical tools, design methods and evaluative techniques to aid in decision making regarding such systems. (Krippendorff)

In a later stage of this work we will also include knowledge from systems design. Systems design is the process or art of defining the hardware and software architecture, components, modules, interfaces, and data for a computer system to satisfy specified requirements.

Systems analysis and systems design of IT management automation systems has not yet become a matter of science such as control systems in other scientific domains which in turn have faced automation problems. Therefore, it seems desirable to profit from existing knowledge that has been gathered in these domains, as parts of their knowledge address parts of the vision stated above. In the design of automation systems in IT management, much more emphasis should be put on building management automation systems based on known development processes and tested standard modules instead of re-implementing software and re-finding answers to many issues, that arise during implementation. See Figure 3.6 for a graphical illustration of this kind of inspiration.

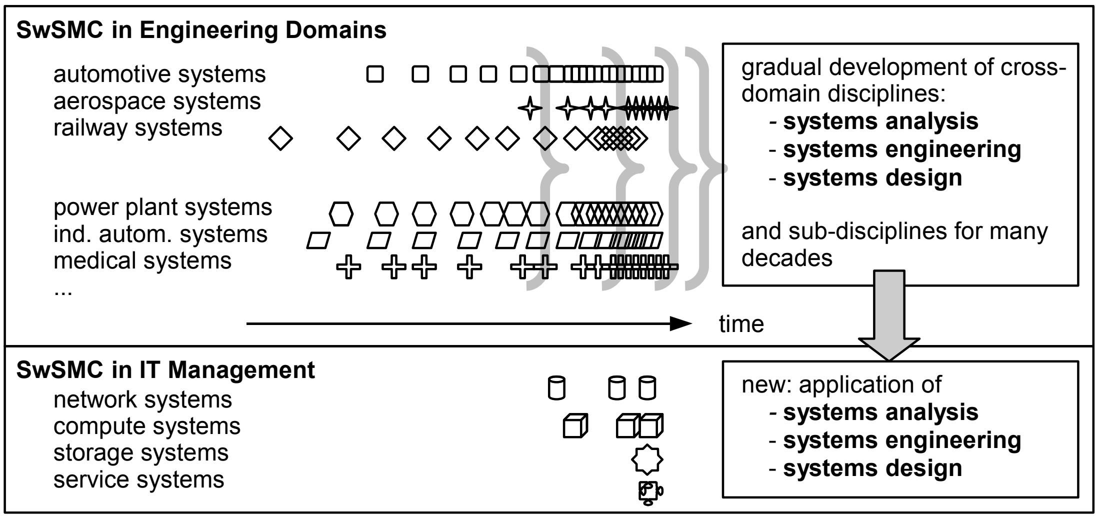


Figure 3.6: The engineering domains have a long experience in systems analysis and systems design. They have started to create more and more complex artifacts about 200 years ago, a few in the beginning and then with increasing frequency. Over time, scientists in the different engineering domains noticed the similarities in their domains regarding aspects like safety, engineering methods (systems analysis and design), theory and founded cross-domain disciplines like systems engineering, roughly about in the 1940s. Which had a positive impact on standardization and maturity of systems, as well as handling growing sets of demanding requirements and therefore increasing complexity. About this time we can also witness the first computers roughly comparable to our modern ones and related systems. Instead of re-inventing these methods, we should investigate their findings in methods and results, and take over relevant parts of this work to IT management systems, as they are just yet another class of systems.

Business process management Business process management (BPM) is a field of management focused on aligning organizations with the wants and needs of clients. It is a management approach that promotes business effectiveness and efficiency while having aiming at innovation, flexibility, and integration with technology. Business process management attempts to improve processes continuously. In turn, business process *automation* is the process a business uses to contain costs. It consists of integrating applications, restructuring labor resources, and using software applications throughout the organization.

We can regard IT management tasks and processes to be the business processes of an IT service provider. This way, we can take over knowledge from business process management and automation.

Automation engineering Automation is the use of control systems, in concert with other applications of information technology, to control industrial machinery and processes, reducing the need for human intervention. In the scope of industrialization, automation is a step beyond mechanization. Whereas mechanization provided human operators with machinery to assist them with the muscular requirements of work, automation greatly reduces the need for human sensory and mental requirements as well. Processes and systems can also be automated. This domain is mainly interesting because of its achievements.

Still, we have to keep in mind that the mechanization part of automation engineering typically deals with physical goods and machines in factory processes. Therefore, artifacts in this domain differ considerably from IT management automation systems, which essentially process information. In IT management, mechanization does not play any role. Instead we can assume, that virtualization (host virtualization, network virtualization, ...) is one of the key elements to reduce dealing with physical resources such as cables in switch ports, or installation media for servers.

3.5.2 R1.1 Common task structure of resource-related tasks in their domain.

EIMA will mainly interface with two domains: systems engineering of composite IT systems from individual IT systems, and software engineering of management auomation systems. Therefore, only these two domains are investigated here for their typical task structure for the sake of brevity and direct relevance. In addition, product lifecycle management / application lifecycle management were recently established as an addition to these two engineering domains. To the phases of requirements analysis, and design, they add considerations on the actual usage time, feedback to development, maintenance, and product/application disposal.

Task structure in Systems Engineering

Due to the application of common methods to a range of different artifacts (airplanes, ships, cars, industrial production plants, power plants, ...), systems engineering itself has been split into special task groups, which each are applicable to many artifacts. This way, among others, the following sub-disciplines have arised:

- supportability / maintainability engineering, reliability engineering, availability engineering
- configuration mgmt, configuration engineering
- cost engineering, operational cost planning, (return on investment analyses)
- performance engineering, scalability engineering, operations research
- security engineering, safety engineering

The order of the sub-disciplines here has been chosen so that they align to the FCAPS grouping of IT management activities. All these domains are mainly focused on the analysis and design of systems. As such, they can also be applied when designing or analyzing IT management automation systems.

We can conclude that for example reliability engineering and availability engineering are pre-thought fault management, availability and capacity management. Because it is highly unlikely that a system despite all excellence in the former two will work as promised at all times, supportability / maintainability engineering are performed at design time to improve later incident management and problem management processes. Here we can already see the relevance of these engineering domains to IT management automation.

Especially later in Chapter 6, the catalog of machine capabilities, we will remember these systems engineering sub-domains and refer to their most relevant methods and techniques that are also well applicable in IT management automation.

Task structure in Software Engineering

There are many publications that give an extensive overview on the tasks in software engineeering, such as [Bal00]. Again, for the sake of relevance and brevity, we will only describe CMMI here, as it has been refined into three documents and one in particular for service providers. Here, we see the most potential for overlap and fruitful inspiration.

Capability Maturity Model Integration (CMMI), 1.2, 2006 CMMI is a general approach to measure and document the improvement of development processes and organizational processes. With a background in software and systems engineering, CMMI consolidated a few earlier standards. Recently, it has been broken down into parts for the following three main application areas:

- CMMI for Development (CMMI-DEV), v1.2 was released in August 2006. It addresses product and service development processes.
- CMMI for Acquisition (CMMI-ACQ), v1.2 was released in November 2007. It addresses supply chain management, acquisition, and outsourcing processes in government and industry.
- CMMI for Services (CMMI-SVC), v1.2 was released in February 2009. It addresses guidance for delivering services within an organization and to external customers.

We can see, that EIMA may eventually come in touch with with all three of them:
CMMI-DEV will be relevant at design and creation time of IT management automation systems. In fact, only a well implemented IT management automation system will stand up to the expectations of its later users. Yet, this thesis assumes excellence in software engineering with the implementers anyway.

CMMI-ACQ will be relevant for the phase, when the appropriate IT resources, which may be deliverd by suppliers, are selected to assemble a resource pool implementing an IT solution.

CMMI-SVC is the most relevant part for systems design in this thesis as in fact an IT solution can be seen in abstract manner as an IT service provider itself. CMMI-SVC lists 24 CMMI-SVC process areas. Essentially 50% of them roughly overlap with ITIL V3.

Consolidation of knowledge

To the previous Table 3.2 on page 59 on task analysis in IT management, we can now add results of systems engineering sub-domains. While these activities may seem unrelated to each other, they in fact match quite well, as can be seen in Table 3.3. They even match subdisciplines of systems engineering, which are named after different kinds of system requirements.

Table 3.3: Task scheme alignment of Ad hoc management, OSI managent SMFA's, ITIL V2, Autonomic Computing, Systems Engineering. Having added the findings in systems engineering, it can be seen, that there is a good match in the alignment of abstract tasks in 1) IT management and 2) engineering sub-domains respectively. Both take care of similar concerns to overcome ad hoc management, but in different phases of the system life-cycle. (The table is reproduced in total to enable comparisons.)

Ad hoc management	**OSI managent SMFA's**	**ITIL V2**	**Autonomic Computing**	**Systems Engineering**
ad hoc fault analysis and correction	Fault Mgmt	Incident Mgmt, Problem Mgmt, IT Service Continuity Mgmt	Self-Healing	Supportability / Maintainability Eng., Reliability Eng.
ad hoc configuration	Configuration Mgmt	Configuration Mgmt, Change Mgmt	Self-Configuration	Configuration mgmt, configuration eng.
flat pricing	Accounting Mgmt	Service Level Mgmt	none	Cost Eng., Operational cost planning
best effort performance	Performance Mgmt	Service Level Mgmt, Capacity Mgmt, Availability Mgmt	Self-Optimization	Performance Eng., Scalability Eng., Operations Research
ad hoc reaction to security incidents	Security Mgmt	Security Mgmt	Self-Protection	Security Eng., Safety Eng.

3.5.3 R1.2 Task decomposition and control hierarchies

Task analysis in Human Factors Engineering/Ergonomics

Task analysis is a common undertaking performed in ergonomics/human factors engineering. It includes the analysis of how a task is accomplished, including a detailed description of both manual and mental activities, task and element durations, task frequency, function allocation, task complexity, environmental conditions, necessary clothing and equipment, and any other unique factors involved in or required for one or more people to perform a given task. Task analysis may be of manual tasks and be analyzed as time and motion studies using concepts from industrial engineering. For an in-depth coverage, see the Chapter 6 “Task analysis” and Chapter 38 “The analysis of organizational processes” in Wilson, “Evaluation of human work” [WC05]. As a more general approach, *work domain analysis* covers task analysis for a number of related tasks in a certain enviroment, such as a helicopter cockpit.

Hierarchical task analysis (HTA) Task analysis often results in a hierarchical representation of what steps it takes to perform a task for which there is a goal and for which there are some lowest-level ”actions” that are to be performed. In such hierarchical trees, goals and plans are inherently included.

For the typically analyzed mechanical tasks, e.g. for automating handling physical items by the use of robots, a number of tasks like grabbing, lifting , and moving items, pulling and pushing them, releasing them is a natural fit. However, task analysis also includes mental activities in additon to physical ones. See Fig. 3.7 for an example from a guide on task analysis, that displays a process control task and is therefore quite similar to a corresponding IT management task.

Cognitive task analysis (CTA) Cognitive task analysis is applied to work environments such as supervisory control where little physical works occurs, but the tasks are more related to situation assessment, decision making, and response planning and execution. This applies well to IT management automation, too.

In cognitive task analysis, GOMS (an acronym that stands for Goals, Operators, Methods, and Selection Rules) is a family of techniques proposed by Card, Moran, and Newell (1983), for modeling and describing human task performance. The components are used as the building blocks for a GOMS model. Goals represent the goals that a user is trying to accomplish, usually specified in a hierarchical manner. Operators are the set of atomic-level operations with which a user composes a solution to a goal. Methods represent sequences of operators, grouped together to accomplish a single goal. Selection Rules are used to decide which method to use for solving a goal when several are applicable. As the author found out after creating the EIMA method, this thesis roughly follows the GOMS idea.

Task analysis in Robotics and AI

Judging from work in robotics and unmanned systems, tasks can be described at multiple levels of abstraction, e.g. in hierarchies (see Figure 3.8), layers or groups. Task decomposition splits tasks to a manageable complexity that does not any further explanations. When tasks are decomposed, the sub-tasks get more and more domain-specific. In advanced management automation systems in engineering, the lowest levels of task decomposition are automated themselves, e.g. in robots inferring actions from (precondition, action, postcondition) rules together with a planning algorithm.

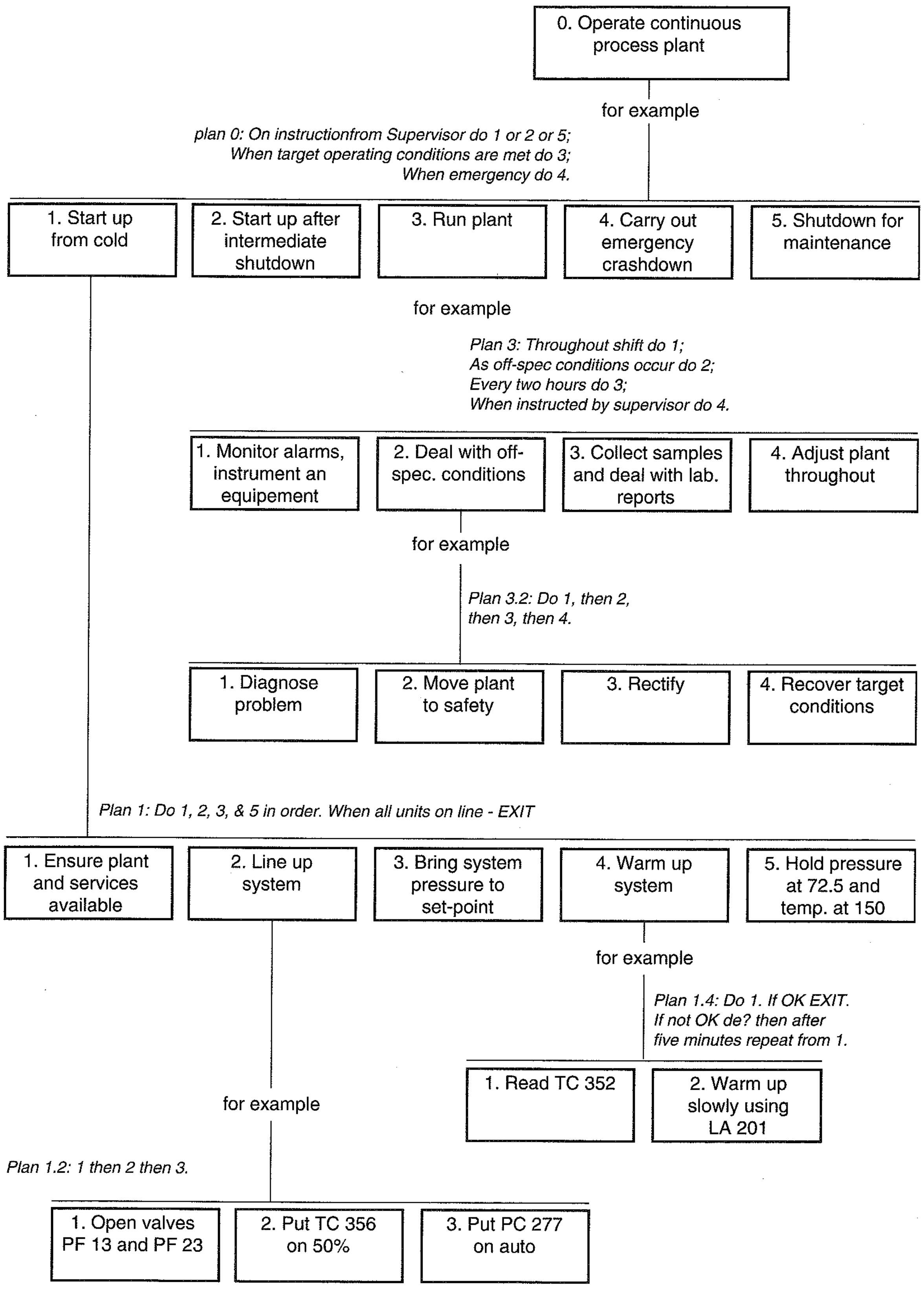

0. Operate continuous process plant
for example
plan 0: On instructionfrom Supervisor do 1 or 2 or 5; When target operating conditions are met do 3; When emergency do 4.
1. Start up from cold
2. Start up after intermediate shutdown
3. Run plant
4. Carry out emergency crashdown
5. Shutdown for maintenance
for example
Plan 3: Throughout shift do 1; As off-spec conditions occur do 2; Every two hours do 3; When instructed by supervisor do 4.
1. Monitor alarms, instrument an equipement
2. Deal with off-spec. conditions
3. Collect samples and deal with lab. reports
4. Adjust plant throughout
for example
Plan 3.2: Do 1, then 2, then 3, then 4.
1. Diagnose problem
2. Move plant to safety
3. Rectify
4. Recover target conditions
Plan 1: Do 1, 2, 3, & 5 in order. When all units on line - EXIT
1. Ensure plant and services available
2. Line up system
3. Bring system pressure to set-point
4. Warm up system
5. Hold pressure at 72.5 and temp. at 150
for example
Plan 1.4: Do 1. If OK EXIT. If not OK de? then after five minutes repeat from 1.
1. Read TC 352
2. Warm up slowly using LA 201
for example
Plan 1.2: 1 then 2 then 3.
1. Open valves PF 13 and PF 23
2. Put TC 356 on 50%
3. Put PC 277 on auto

Figure 3.7: Hierarchical task analysis for a continuous process control task (see [KA92])

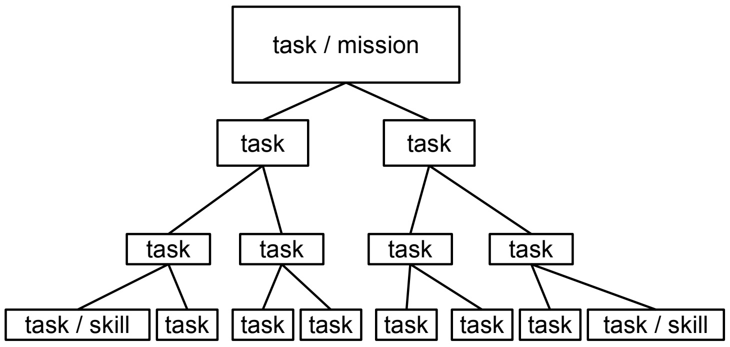


Figure 3.8: Tasks are typically decomposed along a hierarchical structure. Top-level tasks have been named "mission", bottom-level tasks "skill" in research on unmanned systems or "function" in system engineering.

TA in Albus/Barbera, Reference Model for Intelligent systems, 1980's Albus, now at National Institute of Standards and Technology (NIST), and Barbera created a reference model for intelligent systems, later called RCS for real-time control system. It has been applied to a number of major projects at US Navy and DARPA was was continually enhanced with new features. In this model, also a hierarchical task tree is used as the overall goal is being refined into smaller chunks. See Fig. 3.9 for a graphical summary of this approach from on of their more recent papers.

TA in Hierarchical Task Network planning HTN planning is a standard approach in AI to automated planning in which the dependency among actions can be given in the form of networks. Such a structure provides appropriate input for efficient planning algorithms, for which quite a number of applications exist. A planner application then creates detailed plans to fulfill a certain overall mission.

Task analysis in Systems Engineering

Sheridan, Humans and Automation, 2002 This publication [She02] mixes aspects of human factors engineering and systems engineering. Tasks are identified by a specific tabular task analysis scheme. With this scheme, he captures the essential properties of each task without requiring too much modeling at this point in time. At the same time, by using this scheme, Sheridan advises against task analysis being too close to the process to be automated. As the other sources in systems engineering, he suggests, that tasks should be organized in a hierarchical task tree. This tree breaks up an overall goal (mission) into tasks, and resolves each task to machine functions and human skills. Even if its main target application areas are ground and air vehicles, as well as industrial automation, IT management (potentially refered to as "office systems" in the book) fits well with his ideas.

Fault tree analysis (FTA) FTA is a special method in systems engineering. Its purpose is not mainly task analysis, but the origin, probability and effects of faults. Once more, a hierarchical structure is used for tasks to reduce complexity, even if it may not fit all applications.

Task analysis in Industrial Automation

As industrial automation mainly deals with physical tasks, and not cognitive tasks, the literature review was clearly reduced.

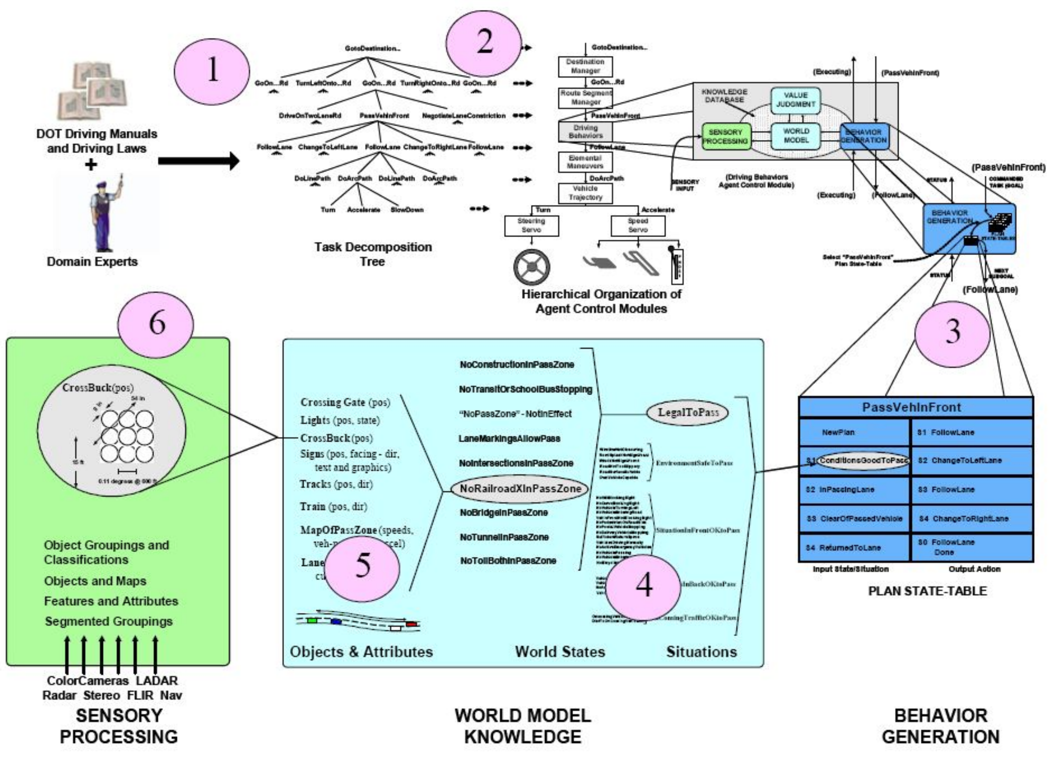


Figure 3.9: The six steps of the RCS methodology for knowledge acquisition and representation. From [AB05]

Takeda, Low-Cost Intelligent Automation, 2006 Takeda [Tak06a] describes very simple changes to improve mostly manual work in assembly lines. While the actual content of this book is not relevant in IT management, he brings up a number of concerns, that are important side notes for task analysis in IT management automation: 1) As an important remark, Takeda advises to not automate all the inefficiencies of a human process. He recommends to first identify and use the possible improvements in the human processes. 2) He shows, that automation is not always robots and more sophisticated robots (comparable to big new management machines and more sophisticated IT management automation systems in our domain). Within assembly lines, he lists about 40 simple changes to reduce common human errors in typical assembly steps. Getting inspired from his ideas, similar simple changes can probably be found in certain IT management tasks. We accept this critique and will remember it at the appropriate points later in this thesis.

In a related publication, [Tak06b] Takeda also advises for synchronized just-in-time production and lean inventories. We can probably translate this into the “On demand Computing” (IBM) campaign, which applies similar concepts in IT management. Synchronized production here compares to on-demand service creation and use, lean inventories translate to less overprovisioning of computing capacity and leasing remote resources from e.g. remote service providers at peak times for short periods.

Froeschle, IT industrialization, 2007 It was already generally found, that the application of parallel principles from industrial automation to IT management, coined as “IT industrialization” makes perfect sense, see [FS07] for a more in-depth coverage.

Figure 3.10: An example format for task analysis (Source: [She02])

Task step	**Operator or machine identification**	**Information required**	**Decision(s) to be made**	**Control action required**	**Criterion of satisfactory completion**

Consolidation of knowledge

Not much cross-domain harmonization of knowledge is necessary regarding the task analysis method, as there is so little existing work in IT management.

The knowledge from the non-IT-management domains suggests, that an IT management automation system may have the form of a hierarchical control system (see [AB05, She02, WC05]). With this overall design structure in mind, but kept separate for the two types of hierarchies (task control and task decomposition), we create the EIMA method to do task analysis.

3.5.4 R1.3 Automation-oriented task descriptions

Quite a number of information models and data formats have been developed in the different domains to represent the results of task analysis. However, they create a heterogeneous landscape with few compatibilities or direct transformations.

Fundamental process properties that can be recognized across domains are the data flow and the control flow. The data flow expresses the dependencies of final and intermediate results/information during a certain process. Control flow refers to the temporal order in which the individual statements, instructions, or function calls of an imperative or functional program are executed or evaluated. One or both of them can can be found in many of the following standards.

Task models/descriptions in Human factors engineering

The typical output of task analysis in human factors engineering is a hierarchical task representation , which can also be presented in a tabular format [WC05]. For example, Sheridan suggests an assessment scheme shown in Table 3.10, that evaluates the essential information which matters in automation. However, up to now there is no advice in HFE on the suitable level of abstraction of the tasks, nor a common model or information or data.

Darwin Information Typing Architecture (DITA) Originally created by IBM for their product documentation, DITA is an OASIS standard and available in version 1.1 since in Aug 2007.

It provides an abstract, task-oriented documentation in terms of topic maps built from chunks of information plus navigational elements. DITA specifies four basic topic types: Topic, Task, Concept and Reference. It allow a task organization in sequences, hierarchies, groups. In DITA, tasks are analyzed into steps, with a main goal of identifying steps that are reusable in multiple tasks. It provides a technology-independent major format for technical publishing and technical documentation, and typically DITA input is transformed to online help applications, web pages, or operator manuals.

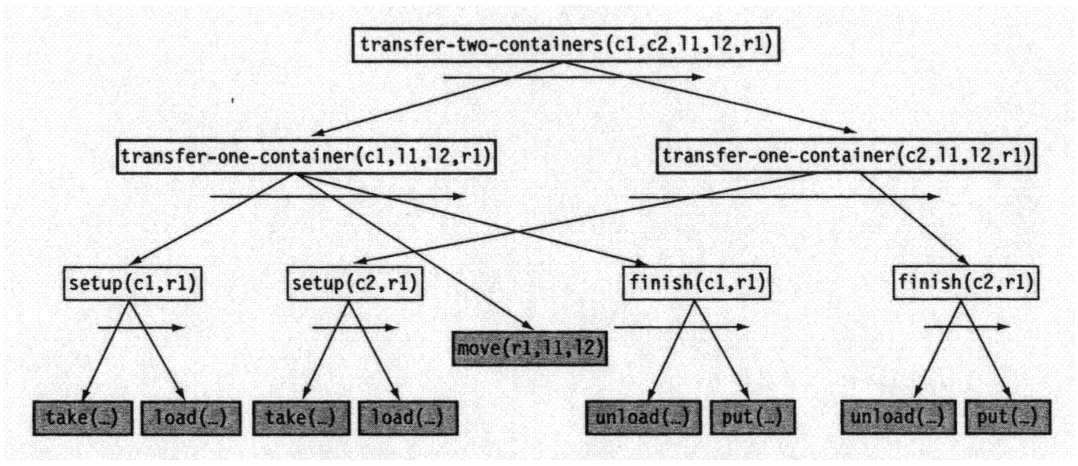


Figure 3.11: Example HTN tree, (from [GNT04])

When compared to DocBook, another standard with a similar application area, DITA is more concept-based, while DocBook is more output-oriented. With the IBM Task Modeler, there is an available software tool to model DITA content.

Concept-wise, DITA content for available IT items is very interesting as a source for IT management automation related task analysis. For one reason, help files/product manuals describe, what users still have to do. They will hardly however, describe the systems internally handled tasks. Thus, if we find DITA content for a product, in which to increase automation, we instantly have a good model of its human-assigned tasks. For another reason, because IT management automation could use DITA for its modeling of task analysis results.

Task models/descriptions in Artificial intelligence

Hierarchical task network (HTN) As stated before, a HTN is a typical format in AI. See Fig. 3.11 for an example. As one can see, the lowest planning levels are the concrete function calls with their appropriate parameters.

Regarding a common data format , in 1998 the Planning Domain Definition Language (PDDL) [MGH$^+$98] was defined, originally to align the input formats for contests of different planning applications (International Planning Competition) . Planning tasks specified in PDDL are separated into two files: 1) A domain file for predicates and actions and 2) A problem file for objects, initial states and goal specifications. In 2008, the latest version 3.1 appeared. As an alternative, the Action Notation Modeling Language (ANML) [SFC08] was presented more recently.

Task models/descriptions in Software Engineering

Unified Modeling Language (UML) Diagrams in UML are rooted in software engineering. UML was designed to include and succeeded to replace a whole range of other modeling languages that were created in software engineering before. UML diagrams can be serialized and saved persistently in the XML Metadata Interchange (XMI) format.

They sometimes have been applied in task analysis in IT management in terms of activity diagrams or sequence diagrams to illustrate certain organizational workflows or technical sequences of actions (protocols), see e.g. [Bre07, Sch07a].

Fundamental Modeling Concepts (FMC) FMC was created from former ideas at the Hasso-Plattner-Institut Potsdam, a SAP-founded research entity, to model and visualize system-related structures in enterprise applications. The main three constructs in FMC were compositional structures in block diagrams, dynamic structures in Petri nets, and value range structures in entity relationship diagrams. For FMC, stencils were available for flow-charting applications, but no immediate persistent format. Therefore its use in automation projects was restricted to human interpretation.

In 2009, SAP revised the relationship of FMC content and UML content into the Technical Architecture Modeling (TAM) format. On the conceptual level, TAM is FMC with UML notation. For compositional structure, block diagrams are used in the known FMC notation, since the UML does not offer an equivalent diagram type. For dynamic structures, it now uses activity diagrams instead of Petri nets (and sequence diagrams for examples). For entity-relationship structures, UML class diagrams are used. Still, its main use is communication about systems among people.

Task models/descriptions in Systems Engineering

In systems engineering, there is no standard format for task analysis results. However, two more prominent formats are IDEF and since recently SysML.

Integrated Definition (IDEF), 1980's IDEF, a standard composed of many sub-standards and still being enhanced more and more, is based on the Structured Analysis and Design Technique. Its origin is in US airforce and US defense applications, where it has been used in the communication with contractors for a number of systems. Due to its variable level of detail, recursive application leads to more and more fine-grained models. An example IDEF diagram is shown in Fig. 3.12.

The basic building block of IDEF0 (IDEF zero) "Function modeling", is an activity, which is a functional block and can also be seen as a task. This block is defined by its input variables, control input from upper functions, and mechanisms enabling the actual function, and output variables. Such an activity quite perfectly matches the idea of a common base loop in EIMA, as will be shown in the next chapter.

Applications for IDEF0 typically support data export of IDEF0 content to XMI (for UML applications), textual formats like HTML, graphical representations in SVG and proprietary drawing formats. We do not know, whether there is also a standardized serialization of the different IDEF sub-standards themselves. For a comparison of IDEF and UML, see [Nor04].

SysML 1.1, 2008 UML by tradition has a focus on software engineering for rather large and portable software applications. This does not always apply to the systems engineering divisions of e.g. the automotive and aerospace sector with software that is more linked to the hardware and needs to fulfill real-time requirements. Therefore, to restrain to the relevant pieces in UML, SysML was recently created by an industry consortium with a background in Systems Engineering as a subset of UML, but extended by features for requirements specification and special diagrams for embedded systems. As its name implies, SysML is mainly used for technical systems, rather than human tasks, therefore human systems are typically not covered. As block description diagrams, sequence and activity diagrams are part of SysML, tasks can be modeled in this language as well.

Fault and reliability analysis Safety engineering uses fault trees for a proper analysis. Fault tree analysis (FTA) as a top-down method has been standardized in national industrial standards. In a

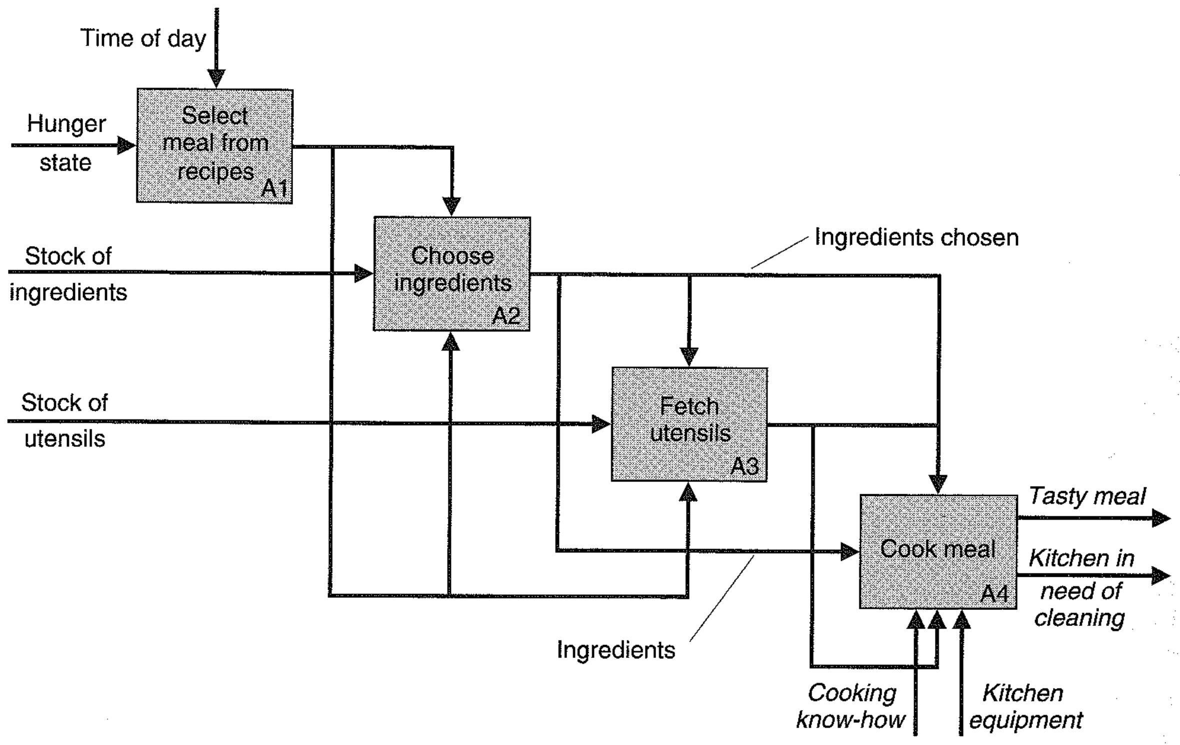


Figure 3.12: An example IDEF diagram. This is a high-level diagram, the boxes of which could be decomposed to other IDEF diagrams. The diagonal arrangement of activities is part of the technique. Demonstrated here is the emphasis on inputs, outputs, controls and resources. Also evident are the interfaces to the rest of the world. (see [WC05])

similar way, failure modes and effect analysis (FMEA) has been standardized and used with top-down and bottom-up analyses. We do now know whether there is a standard data exchange format besides the proprietary individual formats of software for reliability analysis along the FTA or FMEA method.

Task models/descriptions in Industrial Automation

Industrial automation in the sense of Computer Integrated Manufacturing (CIM) can be broken down into a whole range of sub-domains: CAD (computer-aided design), CAE (computer-aided engineering), CAM (computer-aided manufacturing), CAPP (computer-aided process planning), CAQ (computer-aided quality assurance), PPC (production planning and control), ERP (enterprise resource planning). As we can even see from this impressive list, this domain has well matured, which is also documented by a whole number of commercial application (for each domain, there is 2 or 3 widely adopted one, and many more special applications) and standard data exchange formats.

Task analysis will likely be defined in CAPP oder PPC. We do not know of any standards in this domain that is especially targetted at task analysis artifacts. However it is likely, that either the standards from systems engineering are used, or proprietary information and data models.

Task models/descriptions in Computing Grids

In computing grids, there is the need to describe complex compute jobs, which are then expected to be run without further human intervention. The compute jobs run grid applications that process input data available within the grid and produce results saved to the grid.

While in principle make files with dependencies among intermediate and final results would suffice, the data flow can now often be specified either in terms of results or in terms of a workflow to flexibly assemble processing "pipelines" from existing compute services and to express parallelization. Manually created workflows can be checked and verified for certain properties. The dataflow representation allows a flexible mapping to actual processing units by a scheduler. Currently, there is no standard format to describe these workflows, but a whole range of proposed formats. See the section on workflow description languages at `http://www.gridworkflow.org` for a list and further remarks.

Task models/descriptions in Business process management

Business process modeling mainly deals with workflow descriptions for business processes. It has been mainly used to analyze and model common business administration processes, such as human resource management, enterprise resource planning, company cost accounting and planning, customer relationship management, that appear in a similar manner in many enterprises, once they have grown to a certain size. The same applies to municipal and government administration, law, and medical processes, that are to be analyzed, modeled and supported by workflow support tools.

Also in business process modeling, there was a certain consolidation of standards into fewer and more general ones. Here, we only list a relevant subset of standards: EPC, XPDL, BPMN.

Event-driven process chains (EPC), 1992 EPCs are a semiformal graphical modeling language for task description in business processes. They are used in software like the ARIS Toolset and SAP. Originally, proprietary data formats with limited data compatibility were used. Lately, as an academic result, the EPC markup language (EPML) [MN06a] has been created as an interchange format. For an overview on XML languages for data exchange in BPM, see [MN06b].

XML Process Definition Language (XPDL) 2.1, 2008 XPDL was defined by the Workflow Management Coalition (WfMC) to provide a common standard and exchange format for workflow descriptions. XPDL V2.1 is one of the available serialization formats for BPMN 1.1.

Business Process Modeling Notation (BPMN) 1.2, 2009 For improved human readability and expressiveness, descriptions of tasks and processes need an appealing and useful graphical representation. For this purpose, BPMN was created. In addition to be used for illustrations, the actual logic in a BPMN diagram can be saved to XPDL and BPEL, among other formats. BPMN allows relatively high-level task descriptions where a direct automation is not directly visible as humans need to add quite an amount of semantics to the actual content.

Within the scope of EIMA, BPMN has the disadvantage, that the tasks are not grouped according to functional groups. Instead, with its swim lanes, BPMN diagrams use the y-axis to add information on task allocation, while the x-axis roughly represents the sequence of tasks over time.

In EIMA, we intentionally keep two levels of task analysis (tasks first, then break-down into loops), and a separate function allocation step linked to existing machines capabilities. Therefore BPMN content will be reorganized and transformed to a different structure (MAPDEK loops), but we will try to find a good representation within the bounds of the BPMN specification.

Consolidation of knowledge

As we have seen, there is a whole range of task description formats, and they differ in many properties such as formality, graphical representation, available tools, information and data model, and standardization. An in-depth comparison is beyond the scope of this work. Each of the listed modeling formats is capable to model very abstractly and to hide lots of details, or in contrast by recursively nesting diagrams to model tasks in a very fine-grained manner.

However, at this point in time, we do not know yet, which of the functions will likely be assigned to machines and which to humans. Therefore, it does not make much sense to model tasks in a very fine-grained fashion here. In EIMA, we want to achieve a good systems design for the machine functions interacting with human processes, such as in a demonstrator that shows a system simulation or preview, and recommendations to fill the system functional blocks, even if they will later have to be implemented by programmers. EIMA does not replace in-detail function and system design but shall by an aid in quickly finding automation potential.

At this point, we would restrain from workflow models and instead use functional hierarchical models. The reasons for doing so have been described along the two introductory examples at the beginning of this chapter.

Regarding task detail, up to this point, the functions and tasks only have function/task names, while in a refinement step they could be completed to IDEF0 like structures with input and output parameters. The link to methods will be dealt with in Chapter 6, “Catalog of machine capabilities”.

3.6 Proposal on task analysis in EIMA

The last two sections showed a selection of related work on task analysis in both IT management as well as other domains. Even though only a fraction of the available literature was considered, we believe that it gives a good impression on the state of the art.

Now, this section concludes these findings along the three requirements: a task analysis method, a task model, and a good representation of tasks in the domain of resource-related IT management. For each of these, we first compare and normalize the findings to then blend the knowledge to form recommendations to do automation-oriented task analysis of resource-related IT management tasks. To better illustrate the proposal, a chosen example scenario will be used in this section.

3.6.1 Example Scenario

All following recommendations for EIMA task analysis will be applied on an example. As example scenario, the workflow in Fig. 3.2 on change control is used. A certain depth of task analysis is not dictated by EIMA, as it was designed to be flexible in this regard due to its hierarchical structure for task decomposition. Whenever a leaf node is not clear there, it can be "unfolded" and therefore expanded. We will use a depth that is enough to serve as an example here. When an EIMA automation assessment scenario is performed in a real business case, this way the depth of modeling can be chosen in line with available personnel, time, budget and task expertise.

3.6.2 R1.1 Automation-oriented IT management task structure in EIMA

1. Use AutoITIL as a top-level structure for tasks!

Instructions

- Decide on which AutoITIL processes to support with automation!
- Then, use these AutoITIL processes as a sorting scheme for the workflows! Hint: AutoITIL is a subset of ITIL processes/tasks, where automation knowledge is already very well applicable. See Section 3.4.1 for details.
- When there are no workflows to work with, collect a set of unique tasks that relate to the AutoITIL processes and resources!

Reason By an IT management domain analysis, we have aligned a subset of ITIL processes with the FCAPS functional areas for resource management. The AutoITIL set, a set of ITIL processes to which automation can be applied especially well. EIMA performed along the AutoITIL set is supposed to off-load human ITIL processes to machines.

We use AutoITIL as the top level tasks, as ITIL is understood by administrators, and data centers will likely be organized along the ITIL recommendations. At least, every worker will know, which ITIL roles are essentially assigned to him. This way, everyone knowledgeable in ITIL can associate the automation with the known general processes.

Example In the example, we only deal with one workflow, which is named to be a change control workflow, as the workflow title states. Choosing an appropriate superordinate task from the AutoITIL set, we can definitely associate it with change management. This is taken over to the task decomposition diagram in Fig. 3.14 on page 85. When processed by EIMA, it will be broken up into a human part of change management, and a machine-driven AutoChangeManagement part.

To keep the example small, we do not add more top-level AutoITIL tasks, as would be the case if a real automation scenario would be considered with many workflows to work with. In such as case, all the other AutoITIL tasks would have to be examined, if there were more workflows that well go along with some AutoITIL task. As it is just one workflow here, this is not the case. The general idea is to align all the matching workflows with the AutoITIL tasks and to create necessary input if it does not exist.

3.6.3 R1.2 Automation-oriented task decomposition and control hierarchies in EIMA

2. Structure roles in role control hierarchies!

Instructions

- Recognize inherent control hierarchies among the roles in the swim lanes in the workflow, and structure the roles into one or more hierarchies!

Reason In EIMA task analysis, based on the experiences and results in other automation domains, we have chosen to rely on control hierarchies. Hierarchical systems are often favoured in IT management and other domains (the military, government, business administration) because of their controllability, known responsibilities, low overhead, and good fault analysis, even if they might contain single points of failure, which need to be worked around where necessary. Non-hierarchical control structures do have their own advantages, but we esteem that their risks and overhead outweigh their benefits in this particular application.

First, from a control hierarchy we can infer matters on responsibility, overruling of decisions made, and general control. Whenever tasks like authorization tasks or control tasks are found, they should be treated to belong to superordinate roles to the ones, that they control. Ideally, there are not more than two or three control layers though.

Control hierarchies are also used in hierarchical technical control systems, and are easy to understand in every domain due to their resemblance with structures in business organization. Thus, they provide a good mental mapping to a potential former non-automated scenario.

It is important to recognize the difference between these two: 1) task control and 2) task decomposition into sub-tasks. Therefore, when modeling tasks, these two types must be kept distinguishable from each other. We believe that both of them will form a good match, when done well.

Example In the example workflow, we can see nine swim lanes, and therefore nine roles, written in two ways of capitalization, with no clue on semantics. When just looking at these role names, we have difficulties to know about the general organization of these roles. Putting these roles into a hierarchy will remove, or at least reduce, this unclarity.

To create a role hierarchy, an in-depth investigation by a human is necessary, who can understand the tasks and general workflow and who may have background knowledge on the implicit details that are not stated in the workflow. This human may add, remove or rename roles if necessary. In this respect, EIMA task analysis is not a mere formal transformation, but includes interpretation. This is done to improve the results for its suitability for automation.

For the given example "change control workflow", refer to Fig. 3.13 for the result.

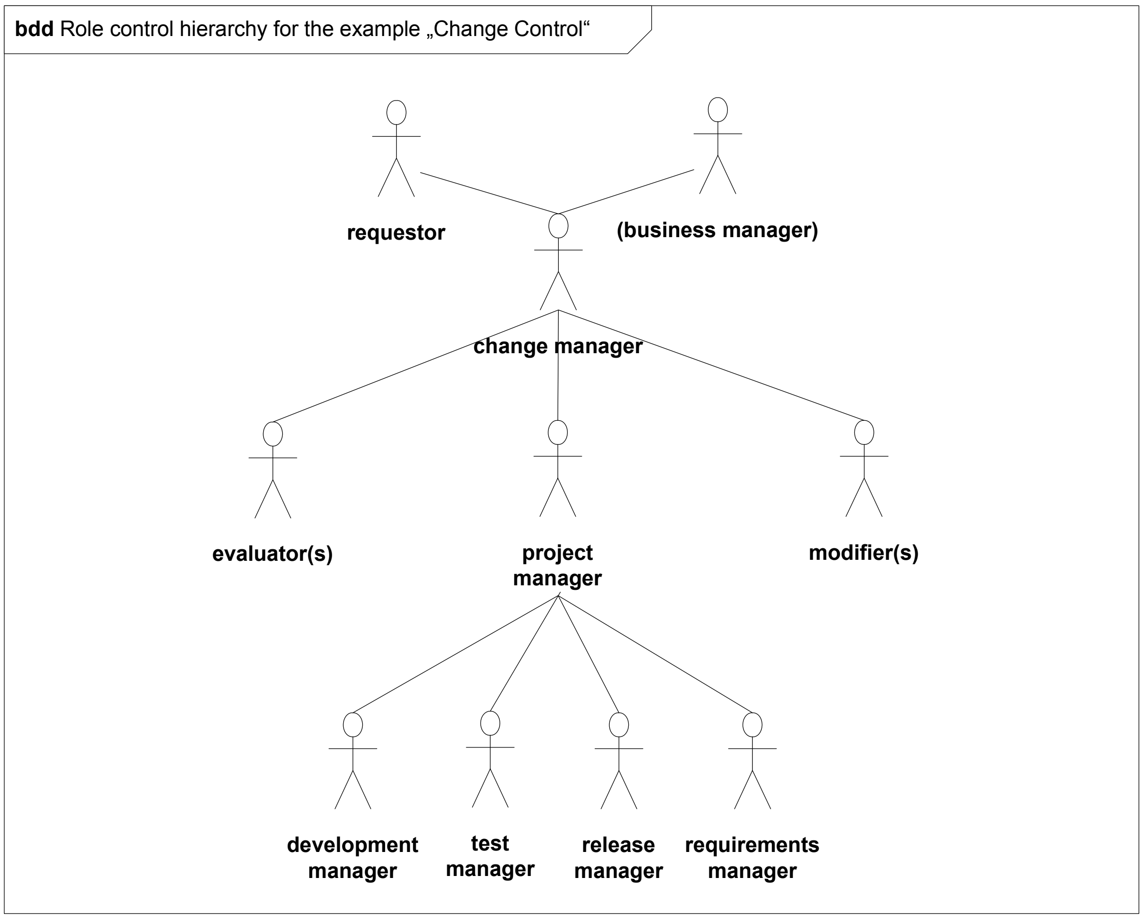


Figure 3.13: SysML block definition diagram to define the role control hierarchy in the example

The central role is the change manager role as this role does the key tasks in change control and also decides on change approval. This role is driven by input from change requestors, who is also the target in case of rejections.

At the same time, the change manager has to balance all his decisions with the business policies directed to him by business management. While this role is not in the original workflow, it is important, because his approval decision must be based on some policies, that come from a certain entity.

All other roles are subordinate to the change manager in this workflow, because they either report to him or get their trigger from the change manager. In principle, the change evaluators could also be upper in the hierarchy (asking his bosses for change request evaluations), but then we see no sense in doing such escalations all the time. There is a reason, that such matters are important in the scope of automation: The higher a task is in the control hierarchy, the less likely it gets that we can apply automation, because more and more strategic considerations influence these tasks. In contrast, if the evaluations were only a few syntactic checks, then they would be lower and easy to automate.

As we can guess from the workflow, the project manager probably leads the change implementation project. This role was therefore put to be superordinate to the other roles that deal with change planning.

All in all, the reader should recognize, that matters of role control hierarchies are important and the new diagram is more expressive *in this regard* than the original workflow. In addition, from a hierarchy many answers on further questions can be given intuitively.

3. Decompose tasks in task decomposition diagrams!

Instructions

- Break up the workflow input into tasks (in fact, the tasks are already visible in workflows)! Then recognize task to sub-task hierarchies, where a task is broken down into steps! Take hierarchical task analysis as an example! (Here, the role control hierarchy is helpful, and the allocation of same tasks to the same role.)
- While creating the task decomposition hierarchy, do also analyze the tasks for similarities with the MAPDEK loop structure of Chapter 4 “Loops”, and insert intermediate tasks to show the categorization! The lowest entities in these task decomposition trees are the skills or functions, that will later be performed by humans or machines.
- Eventually, change the level of detail of tasks (subsume tasks to a common one, or break tasks down)!
- If it exists, use DITA content (see paragraph 3.5.4) from former administrator manuals to take over instructions for human administrators as task descriptions of future automated management tasks!
- If there are no existing workflows as input, derive tasks from the chosen AutoITIL processes to create a set of IT management tasks that evolve around the IT solution!

Reason Creating the task decomposition tree (one or multple of them) is the major activity in EIMA task analysis. Its purpose is to restructure the tasks to show their functional breakdown and to prepare them to be associated with the MAPDEK (Monitor, Analyze, Plan, Decide, Execute, based on Knowledge) categories. The workflows, that may exist as input to EIMA, are often complex and therefore hinder restructuring.

Regarding simplification, task trees contain a lot less information than workflows. This can be compared along the following list:

- Roles and swim lanes: Workflows typically include both, task trees neither of them. We make this up by the role hierarchy presented before.
- Links between tasks: Worfkflows contain links which are either signals or tokens, such as documents, both of which describe the order of tasks. Task trees only have the “is composed of” relationship, that links a composite task with its individual sub-tasks.
- Ordering of tasks: In workflows, task order is given by a start point, then via a path of links, logical functions (e.g. “and”, “or” gateways) and conditionals (if) to a termination point. This can be used for complex, sophisticated control structures. In a task tree, there is a natural depth-first tree order, the child nodes are visited from left to right. Tasks (and therefore sub-trees) can be skipped by the use of text annotations. This is much less powerful, but also much less complex. For simple changes in the level of detail, sub-trees can be collapsed or expanded.
- Input and output parameters: Workflows may or may not include input and output parameters for each task. See the two examples at the beginning of this chapter for reference: Example 1

does have no input and output parameters, the links between tasks simply show a control flow, not a data flow. Example 2 has almost all links annotated by names of events or states, which also illustrates the control flow. None of them includes the data flow. But a data-flow driven workflow illustration would also be possible, such as done in workflows that specify grid tasks.

Task trees do not contain input and output parameters, which again reduces complexity at the expense of detailedness. In this regard, even if details on in- and output parameters are not directly available, the tree shows enough information on the tasks, matching the approach of EIMA to not model more than is really needed. To not loose too much information, but at the same time to get the most relevant information regarding automation (at least according to [She02]) we add task description tables (see next proposal).

We have seen from other automation domains, that the inherent complexity of workflows may not even be necessary. Instead, task decomposition trees promise to be a good concept for automation engineers and system administrators alike in the design phase of systems. Their simple structure allows easy modifications, additions or removal of tree parts, as well as a flexible level of task detail. Of course it must be checked by domain experts, that the new task structure is not contradictive to the old one in such a way, that automation along the new structure would not likely to fulfill the overall effectiveness of the automated routines.

The task decomposition tree is a suitable structure for task description, refinement and is appropriate input for action planning. Regarding task description, EIMA task decomposition trees are essentially hierarchical task analysis (HTA) diagrams, which are intuitively understood on their own. Regarding task refinement, in a decomposition hierarchy it is also intuitive how to refine tasks: leaf nodes in the tree can simply be expanded with children nodes. Inner nodes can be collapsed or replaced with a small sub-tree. With respect to planning, the task decomposition tree also has the advantage, that it can be extended with more data (state space, start state, goal state, task-related action set), and then be transformed into a hierarchical task network (HTN) problem, which is typical input for automated planning. Successful applications for action planning originate mainly in robotics, examples can be seen in the RCS-1 to RCS-4 examples on page 107.

It is important to recognize the difference between these two: 1) role control and 2) task decomposition into sub-tasks. Therefore, when modeling tasks, these two types must be kept distinguishable from each other. We believe that both of them will form a good match, when done well. The joining of both will come later in Ch. 5, where tasks will be allocated to human or machine roles.

Example First, in the original workflow (see Fig. 3.2 on page 48) also tasks were tagged with task IDs T01 to T13 in roughly consecutive order along the primary flow. This already shows that the workflow's layout is not very intuitively presented in the drawing. Then, finding task to sub-task hierarchies is a matter of grouping the existing tasks if they belong together, and at the same time, associating the individual steps with the MAPDEK steps. After doing so, we get the task decomposition tree in Fig. 3.14. There, additional nodes have been included to group the tasks.

We can see, that this task tree is much simpler than the original workflow, and here color coding helps to hold the MAPDEK categories apart. The tasks themselves appear at the leaf nodes in the same order than in the original workflow. The general structure was composed in a way, so that it does already nicely fit the MAPDEK structure. It should now be visible, that this task tree is essentially equivalent to the workflow, but a lot simpler, more accessible, and the new structure allows to look for automation-related properties more easily without having ones mind set on the previously existing task allocation. The specific advantages of this structure will get even more clear, when the loop steps themselves will be investigated in the example part of of Ch. 4.

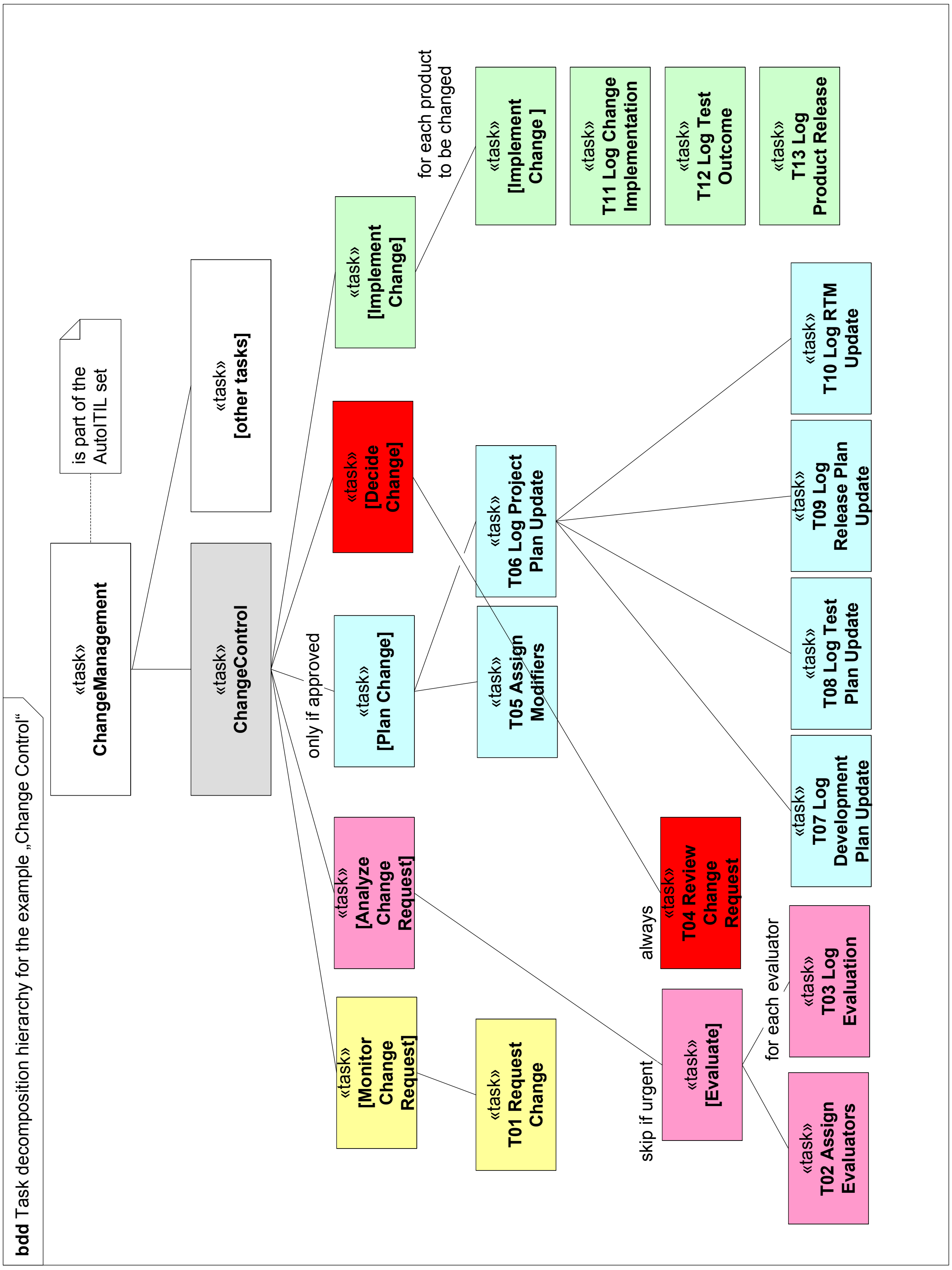

bdd Task decomposition hierarchy for the example „Change Control“
is part of the AutoITIL set
«task» ChangeManagement
«task» ChangeControl
«task» [other tasks]
«task» [Monitor Change Request]
«task» [Analyze Change Request]
only if approved
«task» [Plan Change]
«task» [Decide Change]
«task» [Implement Change]
for each product to be changed
«task» T01 Request Change
«task» T05 Assign Modifiers
«task» T06 Log Project Plan Update
«task» [Implement Change]
«task» T11 Log Change Implementation
«task» T12 Log Test Outcome
«task» T13 Log Product Release
skip if urgent
«task» [Evaluate]
always
«task» T04 Review Change Request
for each evaluator
«task» T02 Assign Evaluators
«task» T03 Log Evaluation
«task» T07 Log Development Plan Update
«task» T08 Log Test Plan Update
«task» T09 Log Release Plan Update
«task» T10 Log RTM Update

Figure 3.14: SysML block definition diagram to define the task decomposition hierarchy in the example. The tasks in brackets are new composite or leaf tasks, that did not appear in the workflow. They are added for clarity.

4. For each task complete Sheridan's automation information table!

Instructions

- In Sheridan's example format for task analysis (see Table 3.10), fill in a line for each task!
- Keep a reference to the task hierarchy, e.g. by task IDs!

Reason Takeda [Tak06a] advises to not automate current inefficiencies in tasks/processes. Likewise, Sheridan [She02] suggests to not "document" all the details the currently performed single steps, but instead to specify information that is required, the decisions to be made, the control actions to be taken (at the level of the controlled process), and the criterion for satisfactory completion of that task. This keeps enough flexibility for modification for better automation and more freedom in later function allocation. We take over Sheridan's approach as this scheme nicely captures control-relevant properties, and is both generic and specific enough to be applied to every task without causing too much in detail modeling or text before the later task allocation step. It also well fits the MAPDEK loops that we will introduce in detail in Ch. 4 "Loops".

Example The task table is a central source for automation related knowledge. It contains: the allocation of tasks to roles prior to automation design (taken over from the original workflows), relevant required input information, made decision, relevant output in terms of control actions, and a criterion of satisfactory completion, which sums up the most relevant data. While it is still verbal and informal, it is at least structured in a strict and comprehensive manner. The result for this example is shown in Fig. 3.15.

It can be seen here, that the original workflow also does not contain all of this information. For this reason, and due to the necessary creativity in composing the other EIMA task analysis artifacts, EIMA task analysis should not be seen as a "machine transformation" of existing workflows. It involves a lot of background knowledge, potentially interviews on unclear matters, educated guessing, because the input information may be incomplete (even if workflows existed, they do not specify enough relevant items), and it may be ambiguous, not up to date, or the IT management practice in a certain enterprise may differ from the documented process. The author of this thesis is confident that the mentioned flexibility of EIMA proves to be an advantage, instead of a disadvantage.

3.6.4 R1.3 Automation-oriented task descriptions in EIMA

As stated as a prerequisite in Chapter 1 we expect workflows as input that include the IT-resource related tasks to be automated. These workflows are documented procedures which can be given in terms of some human-understandable description format, they already include all the tasks. The following recommendations all deal with restructuring this knowledge on the processes and tasks, no hints are given on the actual learning about the processes via interviews or questionnaires. Many techniques for this last matter (getting processes out of people's minds) is described in books on task analysis in human factors engineering/ergonomics, e.g. [WC05].

5. Use a modeling application to model tasks, with support for data exchange formats!

Instructions Tasks should be modeled instead of only simply be described verbally or drawn: Identify and use an application for modeling task analysis!

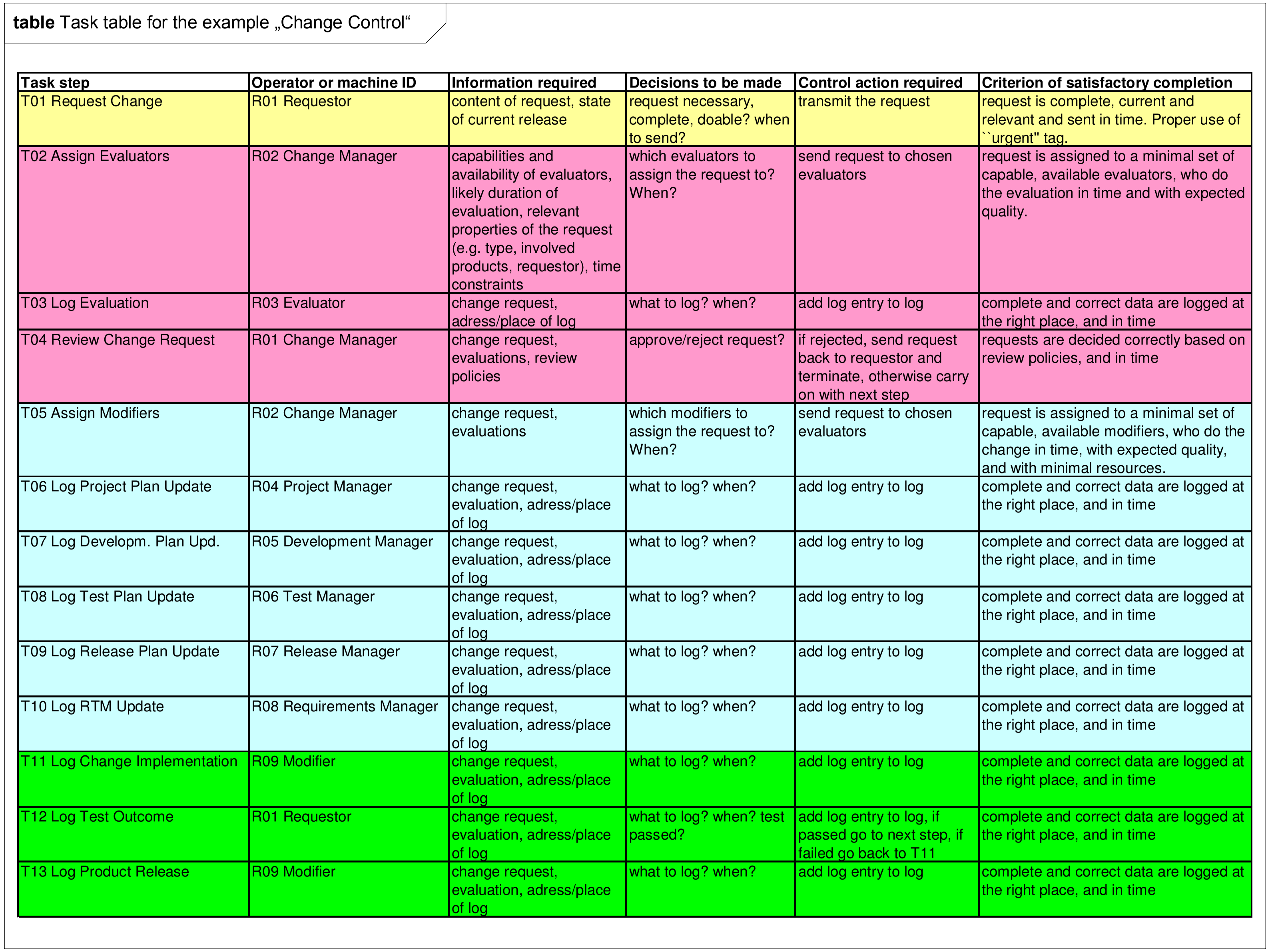


table Task table for the example „Change Control“

Task step	Operator or machine ID	Information required	Decisions to be made	Control action required	Criterion of satisfactory completion
T01 Request Change	R01 Requestor	content of request, state of current release	request necessary, complete, doable? when to send?	transmit the request	request is complete, current and relevant and sent in time. Proper use of ``urgent" tag.
T02 Assign Evaluators	R02 Change Manager	capabilities and availability of evaluators, likely duration of evaluation, relevant properties of the request (e.g. type, involved products, requestor), time constraints	which evaluators to assign the request to? When?	send request to chosen evaluators	request is assigned to a minimal set of capable, available evaluators, who do the evaluation in time and with expected quality.
T03 Log Evaluation	R03 Evaluator	change request, adress/place of log	what to log? when?	add log entry to log	complete and correct data are logged at the right place, and in time
T04 Review Change Request	R01 Change Manager	change request, evaluations, review policies	approve/reject request?	if rejected, send request back to requestor and terminate, otherwise carry on with next step	requests are decided correctly based on review policies, and in time
T05 Assign Modifiers	R02 Change Manager	change request, evaluations	which modifiers to assign the request to? When?	send request to chosen evaluators	request is assigned to a minimal set of capable, available modifiers, who do the change in time, with expected quality, and with minimal resources.
T06 Log Project Plan Update	R04 Project Manager	change request, evaluation, adress/place of log	what to log? when?	add log entry to log	complete and correct data are logged at the right place, and in time
T07 Log Developm. Plan Upd.	R05 Development Manager	change request, evaluation, adress/place of log	what to log? when?	add log entry to log	complete and correct data are logged at the right place, and in time
T08 Log Test Plan Update	R06 Test Manager	change request, evaluation, adress/place of log	what to log? when?	add log entry to log	complete and correct data are logged at the right place, and in time
T09 Log Release Plan Update	R07 Release Manager	change request, evaluation, adress/place of log	what to log? when?	add log entry to log	complete and correct data are logged at the right place, and in time
T10 Log RTM Update	R08 Requirements Manager	change request, evaluation, adress/place of log	what to log? when?	add log entry to log	complete and correct data are logged at the right place, and in time
T11 Log Change Implementation	R09 Modifier	change request, evaluation, adress/place of log	what to log? when?	add log entry to log	complete and correct data are logged at the right place, and in time
T12 Log Test Outcome	R01 Requestor	change request, evaluation, adress/place of log	what to log? when? test passed?	add log entry to log, if passed go to next step, if failed go back to T11	complete and correct data are logged at the right place, and in time
T13 Log Product Release	R09 Modifier	change request, evaluation, adress/place of log	what to log? when?	add log entry to log	complete and correct data are logged at the right place, and in time

Figure 3.15: SysML table to define the tasks along Sheridan's example format, see [She02]

Select the data format for task analysis results, so that it fits with other application software for systems analysis and systems design!

Reasons Due to its size, the task hierarchy (control hierarchy, and task refinement hierarchy) should be a model in one or more machine-interpretable files. While pen and paper will suffice (and be the more prominent choice) for sketches and in brainstorming sessions, engineering domains have shown with CAD/CAM systems, SPICE applications, and similar standard tools, that appropriate development tools are a key factor to cope with comlexity, especially with distributed teams for systems design and development.

A pure text editor to create model files is an opponent to productivity and consistence, in this case. To edit the files, we need a suitable editing application that allows easy and versatile task modeling in a graphical view, eventually extended by a text view. In addition, features for versioning and change control are necessary. In addition, it should allow, (among other operations): queries, bulk operations, filtering, linking, grouping, all with machine support.

It is very important, that the results of task analysis are modeled and persistently saved in a format that other applications can import and build upon. Therefore, pure drawings are not more than a first start to have a handdrewn sketch in a more presentable format. What really matters is to deal with the actual meaning and content of each task, as well as its relationship to other tasks. In fact, the task analysis results have limited value on their own. The task model or function tree will later be needed to be linked with actual system components. The format must be supported there, in a systems design tool, as well.

To actually represent the hierarchical model in a file, it is hard to decide in favour of or against a certain task model. From the point of view of the author of this thesis, it is not so much relevant, what specific application or model to use as long as they support trees and tables. This can be e.g. SysML (functions tree/task tree as a block definitions diagram or SysML tree structure), or a home-grown XML format plus the necessary transformations to translate the data to a format, that other applications for systems analysis and design can import.

Example In the example, we used SysML block diagrams for the role hierarchy and task decomposition tree. We could also have used the special SysML “tree” diagram, if there had been support for it. The task table is a SysML table, which is also a predefined element of this language. For SysML, appropriate modeling applications exist, that can save these models in files.

In the example, all three models: the role hierarchy, the task decomposition tree and the task table, were created with Visio and Excel. A SysML modelling application, such as Artisan Studio would have been more appropriate in terms of modelling but was not available to the author of this thesis.

Regarding data exchange: SysML has standard serialization forms XMI and MOF. These formats are intended for model data interchange between modeling applications. While there are still difficulties in practice, the OMG Model Interchange Working Group works on interoperability issues, see e.g. `http://www.oose.de/blog/2009/07/22/modelle-austauschen-miwg.html` for the state of July 2009.

3.7 Conclusion

This chapter has shown a proposal for task analysis in EIMA, that was derived from existing knowledge in IT management as well as systems engineering and other domains with a profound background in automation. Regarding the EIMA task analysis method, we presented four decisive steps to translate workflow input into a hierarchical set of tasks along selected processes, named AutoITIL, which are the presumably best automatable ITIL processes. For this matter, we presented a table to align multiple proposed task structuring schemes. Knowing about this alignment will definitely improve later cross-domain inspiration and at the same time reduce the set of names to a core set to reduce name heterogeneity.

To model the task analysis result, we gave hints on choosing a good representation format, that is both machine-interpretable (for data interachange) and human-readable. Here, SysML looks like a good candidate to model role hierarchies, task decomposition trees, and task description tables. Apart from task analysis, SysML will later also well cover function allocation, the actual system, and even the requirements in a way that is standardized. SysML has a graphical representation, a standard persistent format (XMI or MOF), and is well supported by modeling applications, even though with room for improvements on data exchange.

The following chapter will continue the method and show, how we give up a workflow-related presentation of tasks, and instead model tasks with a common base loop with a MAPDEK structure. The overall idea is to represent an IT management automation systems as a hierarchical control system which still keeps ITIL's spirit to be most comprehensible to system administrators.

4 Step 2: Loop composition from tasks

Contents

The previous chapter described the results of task analysis in engineered IT management automation (EIMA), this one will now show how loops will serve as a structuring element in the task decomposition tree. The chapter opens with a motivating example, which will be used to derive three general requirements on these loops. Then, as in the previous chapter, relevant existing knowledge on loops will be presented as an overview on the state of the art in the domain of current IT management. In the same manner, and intended as a source of cross-domain inspiration, we will investigate the state of the art in other research disciplines, that can contribute to the use of cognitive loops for automation. The findings in all of these domains will then be checked against the requirements listed at the beginning to spearate the more promising items of related work from the ones being less applicable. The chapter closes with proposals on structuring tasks with loops and modeling these loops in EIMA.

4.1 Motivation and current examples

In Chapter 3 we used workflows as a starting point for task analysis, two example workflows were shown in Fig. 3.1 and 3.2 near page 47. From an automation point of view, we had identified two disadvantages associated with such workflow structures: heterogeneity of the workflow and heterogeneity of tasks, and unclear reduction of complexity, as well as much necessary modeling. Then, in the course of task analysis in Chapter 3, the process description structures were changed from a workflow-like representation to a hierarchical one. This resulted in a task decomposition tree with tasks and relationships between them. See Fig. 3.7 for an example, which already looks complex enough, yet it can be assumed that it was already drastically reduced by the authors to fit on a printed page. We can expect that average real world examples to contain about 10, 100 or 1000 times more tasks in one or more task decomposition trees for a subset of the AutoITIL processes. So, even when arranged in a task decomposition tree, the number and variety of tasks in the trees are an expression of heterogeneity. They may all nicely fit into the tree structure, but the tasks are very many and very different. In the next three subsections, the requirements on structuring elements in the tree will be motivated. A preview on the intended result can be seen in Fig. 3.14 on page 85, where a single loop was introduced. Later, loop structures can also be nested and cascaded. A more complex example will be shown in the example at the end of this chapter.

4.1.1 Necessity of a structuring element in the task decomposition tree

We need to take into account, that automation is the more applicable, the more structured a problem is and the more heterogeneity is reduced (see section 1.2.1) . We had addressed number and heterogeneity of tasks with unfolding or folding the lower levels of the task decomposition tree, but there is a limit to this type of (easy) simplification. There is a certain level of complexity in tasks, that cannot be further reduced without severely veering away from the original automation task. As we need to keep this minimal required detail in tasks, there is no way around keeping the “minimal” necessary size of the task decomposition tree (being the lower limit in terms of the number of tasks, which will still result in many tasks).

Then, within a tree of this least scale, more structure needs to be added in these task decomposition trees than the simple hierarchical links with “is sub-task of” semantics. By design, we have also set the precondition, that the majority of the tasks is repetitive. Repetition is one of the necessary preconditions for automation, because if the tasks were not repetitive then we would unlikely apply automation to them. Repetition is a common pattern to fight complexity in terms of scale. Repetitive actions can be trivially simplified by specifying one cycle, and then specifying a number of iterations of that cycle.

This is called a loop in IT. Repetition well applies to all AutoITIL processes, for many of them we can assume a periodicity of multiple instances per day or month. Considering the repetition, we can translate the tasks to loops.

Therefore we want to combine tasks in the task decomposition tree into loop composites, by adding loops as an additional structure overlay. As the task tree covers multiple levels of abstraction, the loops need to be applicable at multiple levels of abstraction as well.

4.1.2 Necessity of a common structure

When looking at the examples, what still hinders automation is the high heterogeneity of tasks. For example to name just a few imaginary ones, that are typical in IT management: “restart HTTP server”, “plan resource allocation on computing cluster”, “compare configuration files x and y for semantic differences”, “find similar incident cases in the knowledge base”, “create back-out plan”, “watch service level compliance”, “copy `$HOME/.ssh/id_dsa.pub` to remote server”, “set up HA cluster”, “obey to company security policies when configuring email archival and remote access”. We would profit, if we could at least associate them with categories.

Theoretically, even the use of loops at multiple levels of abstraction leaves open, whether these loops are structured in the same way (all loops have common task categories), of whether these are loops with differing structure (different categories dependent on the loop instance). Differing loop models however would not help in grouping tasks according to similarities across the tasks and sub-tasks. If we knew a person or assistant system that did a certain task very well, we would not see the other related tasks, where this person’s skills or this system’s capabilities had a good chance for similarly good performance, unless the target task would have the same loop structure.

We therefore need loops that associate the tasks with *common* categories. It is clear, that a *commonly structured* loop model would also reduce heterogeneity and therefore improve automatibility. At this point, it remains open whether a common loop model for these widely varying tasks exists at all. But it is very important to remember, that the tasks that EIMA deals with are not just any tasks. They are all cognitive tasks, as we have set this as a precondition in the previous chapter (the tasks requiring physical work are few in comparison to cognitive tasks in IT management).

It is now interesting, that the structure of a cognitive loop can be found in many domains like human factors, systems engineering, business management and artificial intelligence. It is a known concept, even though interpreted a little differently in each research domain or application domain. Again, this is good because of the potential knowledge transfer and bad because of syntactic and semantic differences. As done in Chapter 3, we once more need to combine the different domains’ knowledge without at the same time increasing heterogeneity. Now, with this knowledge from other domains on cognitive loops, we have achieved two welcome properties: 1) a clear confirmation that cognitive loops are applicable to cognitive tasks in general and 2) there is related work that we can build on during the state of the art analysis.

4.1.3 Necessity of machine-interpretable loop modeling

As with the result in task analysis, *engineered* IT management automation relies on models. EIMA task analysis was proposed to be performed in a modeling applications with a data exchange format. As loops are an overlay on the task decomposition tree(s), we need to keep references between loops and tasks, and so loops need to be modeled as well in the same or a related modeling tool. The profits out of such an approach are easy data exchange, better consistency, and availability of models to distributed groups and linking to the other EIMA modeling artifacts.

4.2 Requirements on Loops in EIMA

With the motivation of the requirements just stated, we can now briefly list the key aspects of the individual requirements.

4.2.1 R2.1 Loops must be usable at multiple levels of abstraction

As stated above, tasks and therefore loops will be stated at multiple levels of abstraction during initial systems analysis and early systems design, such as business level of abstraction, resource business level of abstraction, or levels in between. This heterogeneity clearly hinders automation, but homogenization comes at the price of either cutting down the tree, or more modeling. In essence, we have two choices: 1) to live with the heterogeneity and cope with it in the loop model. Or 2) to refine or subsume the loops so that all tasks are described at and broken down to the same level of abstraction.

As a complete refinement would require much work for an automation engineer, we require that the EIMA loop model must be applicable at multiple levels of abstraction. This allows to model the automation loops in a high level of abstraction or quite in detail. The loop should be generally applicable to business control or to resource control. This way, it is applicable for communication between system administrators and systems designers. When applied to resource control, it should be adaptable to resource-specific properties.

Also regarding the level of abstraction, loops must be informal enough to be used as a means for human communications, but at the same time formal enough for use in computer-based modeling systems.

The most relevant criteria regarding loops at multiple levels of abstraction are:

- a variable level of proximity to the business and resource abstraction layer in IT management
- a low level of necessary mathematical formality, but enough options to formalize the model

4.2.2 R2.2 Commonly structured base loop

As said before, automation profits from homogeneity. If we cannot achieve a common level of abstraction, then we should at least require a common loop model, and its steps should express a cognitive loop. This common loop model must overcome the differences in the cognitive loops of the different domains. Thus, we should either choose one of the existing ones, or reasonably merge the existing ones to form an “umbrella” loop model. Clearly, in EIMA loops will supplement or replace the known workflows. The loop steps must be general enough, so that essentially for each task there is a proper loop step to be associated with.

In addition to these properties, this loop model must be well applicable to the remaining steps of the EIMA method. By choosing a certain loop structure as overlay on the task decomposition tree, that at least gives a hint on the automatability of its steps, we can prepare the task decomposition tree for function allocation and have a first indicator, whether certain functions will be more likely be assigned to either machine capabilities or human skills. It must allow a variable level of automation (see Chapter 5), as later we will decide for the individual loop steps, how human administrator and the management automation system collaborate. This implicit additional requirement will be taken into account in this chapter, details will then be described in Chapter 5.

And the alignment of tasks to loop steps needs to add a real benefit. In addition to function allocation, this benefit must be in finding appropriate automation solutions quickly and taking over knowledge from one task associated with a certain loop step to a different task that is associated with the same loop step. See Chapter 6 for an implementation pattern catalog of available machine capabilities aligned to the loop composites in general, and along the MAPDEK categories for individual tasks.

The most relevant criteria regarding a commonly structured base loop are:

- a fair number of loop steps: Too many task categories inside a loop cause too much confusion because of the diminishing differences, in contrast too few categories would have little added value in terms of a classification.
- evidence that the commonly applicable steps are really applicable to the majority of IT management tasks: When proposing task categories for IT management tasks in general, it is impossible to prove a that all tasks can really be split up along the classification scheme. We should identify a base set of categories, that has already been regarded by many as being a commonly applicable one.

4.2.3 R2.3 Machine-interpretable loop modeling and linking back to the tasks in task analysis

It is a common practice in engineering, that the exchange of knowledge about systems analysis, system design, and systems construction in terms of models is not only welcome, but required once either systems get more complex or when teams of engineers collaborate. As loops are one intermediate result in the progression of this process, we need to model the loops. And as we can predict a large number of loops for an average system, we will profit from machine support when designing the conceptual loops. Moreover, we need to link the loop steps to the tasks in the task model created in Chapter 3 and to the additional information presented later. All taken together, also for the loops we need an information and data model, as well as ideally existing applications that allow such modeling and data exchange, which at the same time have a useful modeling interface for humans.

However, to keep EIMA overhead low, loop modeling in EIMA is bound to an approach, in which with relatively low effort we can achieve relatively high utility. For this reason, loop modeling is reduced to first associating each task with a task category. Then optionally, task analysis and loop modeling can be reiterated to refine loops (e.g. add additional tasks for other categories) and re-arrange tasks. For better comprehension, the loops may be visualized. At all times, the task data in the task decomposition tree needs to be kept synchronized to the loop models.

If early at design time features like simulation, verification and optimization would be possible along the models even at this high level of abstraction, which is tasks and loops so far, then such "computer-aided systems engineering" has even more value.

The most relevant criteria regarding loop modeling with machine support are:

- a standard modeling scheme with application support
- a data format for data interchange between applications

4.3 Existing knowledge about loops in current IT Management automation

Loops at different levels of abstraction have been used in IT management. They are presented here roughly in an order from more business-level improvement cycles down to control loops embedded into the control logic of management applications. It should be noted though, that essentially many of the loops can be applied at multiple levels of abstraction (close to resource management, close to business management) if simply the level of abstraction of the information, events, and actions is changed as well.

4.3.1 Improvement cycles in ITSM frameworks

ITIL describes the work of IT management along two kinds of layers: 1) the actual daily tasks which are described verbally for each individual ITIL process, even though not in a loop fashion, and 2) above the previous layer, a set of continuous improvement tasks to improve the working steps in the lower layer and even the improvement cycle itself (see Fig. 4.1).

As with similar standards, such as CobiT and CMMI, the improvement cycles are described in a separate volume "Continual Service Improvement" in ITIL V3, and they all roughly align to the Plan-Do-Check-Act cycle (see Fig. 4.2) known from quality management, which was originally described by Deming.

The level of abstraction of these improvement cycles is close to strategic management and business controlling, and not close to resource-related IT management. If we wanted to automate at this level, it would be the next step after tackling the lower loops that do the actual resource reconfiguration actions. And it would involve first capturing all the necessary input such as costs, risks and benefits, including their probabilities, and overall business strategy. The standards give hints to help in modeling these parameters mathematically, but they need to be adapted to the individual situation.

4.3.2 Cognitive loop in FOCALE architecture

Other IT companies (e.g. HP, Microsoft) have started similar research initiatives, but not achieved a similar impact on the worldwide research community. Their loops (see Fig. 4.3) essentially match the IBM idea. "Observe" is monitoring, "compare" is analysis, "actions" matches execution. The learning phase is new, the policy server was outside of the loop in the MAPEK loop. The "model-based transformation layer" is the touch-point. DEN-ng models and ontologies are part of the knowledge base in the MAPEK loop. Finite state machines are either part of the program logic itself, or would be content of the knowledge base as well in MAPEK.

The FOCALE architecture [SDBD06] by the Autonomics Lab inside Motorola's Network Research Labs stands out from these initiatives, because it includes learning and discovery for cognitive networks, both fixed and mobile. The central idea of the FOCALE architecture is to include even more AI techniques, such as machine learning, decision making, context handling. This can be seen as an expression of the fact, that operations support systems and management automation already have a good standing in the telecommunications industry.

4.3.3 MAPEK loop of the ACI

In IBM's ACI, the basic building block is called an autonomic manager. One or multiple managed resources and an autonomic manager form an autonomic element. Inside an autonomic manager, there is

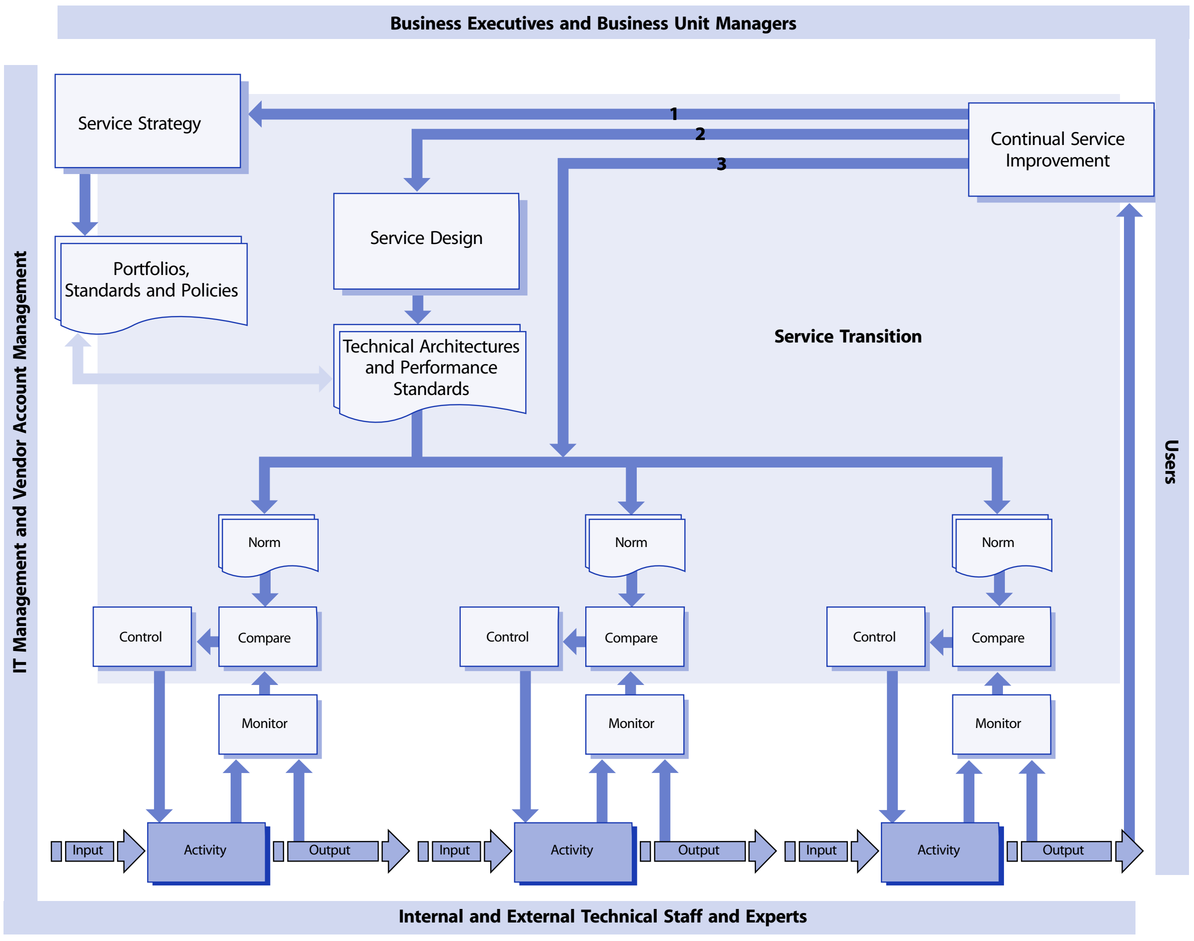

Business Executives and Business Unit Managers
IT Management and Vendor Account Management
Users
Internal and External Technical Staff and Experts
Service Strategy
Portfolios, Standards and Policies
Service Design
Technical Architectures and Performance Standards
Continual Service Improvement
1
2
3
Service Transition
Norm
Control
Compare
Monitor
Input
Activity
Output

Figure 4.1: ITSM monitor control loop in ITIL V3 Continual Service Improvement (see [Spa07])

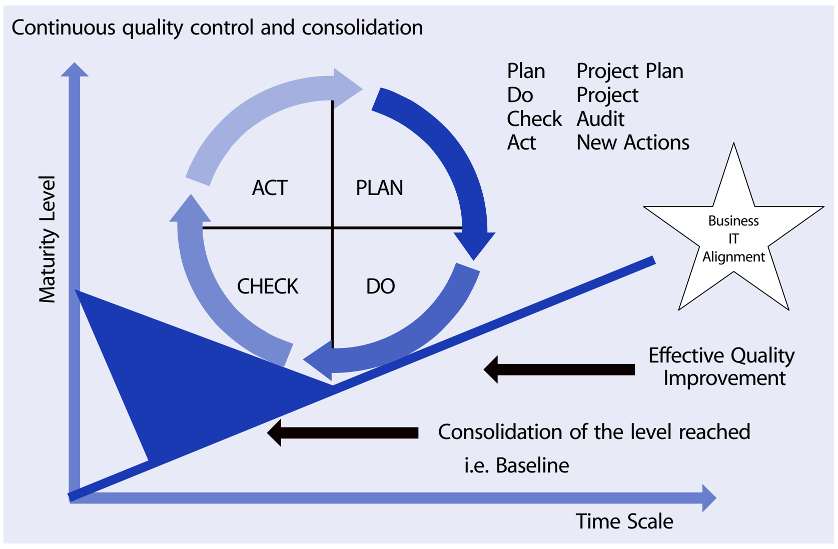


Figure 4.2: The Deming cycle (see [Spa07])

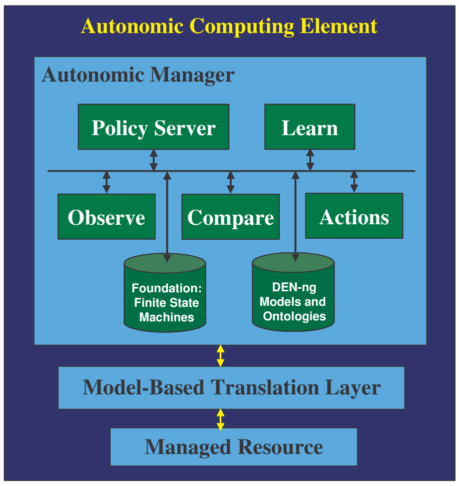

Autonomic Computing Element
Autonomic Manager
Policy Server
Learn
Observe
Compare
Actions
Foundation: Finite State Machines
DEN-ng Models and Ontologies
Model-Based Translation Layer
Managed Resource

Figure 4.3: Autonomic Computing Element in Motorola's FOCALE architecture (see [SDBD06]).

a MAPE-K loop (see Fig. 4.4), or less frequently MAPE loop. In the vision of Autonomic Computing, Kephart describes the MAPE-K loop as inspired by the human autonomic nervous system.

The structure of an autonomic manager is basically defined by parts for the following: monitoring, analysis, planning, execution, a local knowledge base, sensors and effectors to the lower autonomic manager or resource, and an interface to the upper autonomic manager or human. The steps are connected to each other and the outside interfaces by a communication bus, so that they do not have to be executed in strict sequential order.

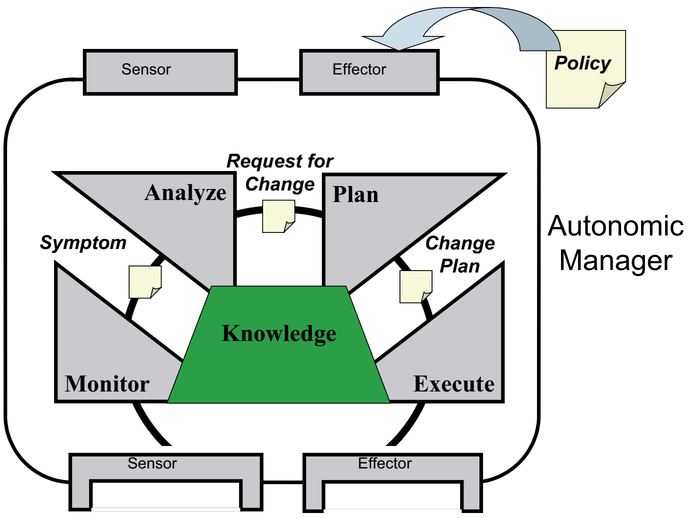


Figure 4.4: A MAPEK feedback control loop (see [IBM06]). We can see, that in this recent version ITIL change management already had a certain impact.

Conceptually, an autonomic manager senses the current state of the outside world via its sensor interface, then the data is processed in the monitoring component, then analyzed and with prospective actions being planned, and finally executed via the actor interface. All components are based on available knowledge. This is performed in a loop-like fashion, either continuously or based on triggering events and conditions. We can associate this "behavior" with a cognitive procedure. In relation to the workflows, here the individual tasks inside the loop are attributed to one or more of the MAPEK-steps.

The lower interface connects to either a resource, then this autonomic manager is called a touchpoint manager, or to other autonomic managers. The touchpoint encapsulates all data transformations that are necessary to adapt to the resource, while the MAPE-K components are the "intelligence" of the autonomic manager.

The upper interface allows other autonomic managers or humans to exercise a certain amount of control, such as communicating constraints, goals, target states, domain knowledge, rules and other policies. Of course, for an autonomic manager to function properly, the target state must be reachable by the locally performable actions and effectors. The analysis and planning routines have to figure out, how to get to the target state from the current state. Also via the upper interface an autonomic manager can send data such as alerts, reports, feedback, delegations to the upper entity.

Autonomic Computing has a flat model for each single autonomic element (AE) with single autonomic manager (AM). Then, a hierarchy (layered architecture of AEs) is formed of the layers: Resource Elements, Composite Resources, Business solutions (see Autonomic Computing Concepts). Still, the ACI autonomic element is not described in a very explicit way. It is used from time to time in research papers, but often the paper content is not even aligned to this MAPE-K step model.

4.3.4 Triggered application of event, condition, action (ECA) rules

Event, condition, action rules are actually conditional actions: on $< triggerevent >$ check if $< condition >$ holds, if so do $< action >$. In comparison to MAPE-K, in ECA rules monitoring/information input for the trigger event and variables in the condition is implicitly assumed, the analysis is the condition, the action matches the execution step. Further analysis, planning, and knowledge is either not performed, or inherent to the action, or paid attention to by the rule writer by a masterfully composed set of ECA rules.

Event Condition Action rules in addition with other types of policies (such as goal policies, obligation policies and constraints) are used in policy-based management. A specialty of ECA rules is that—provided that one can catch most of the semantics of the business rules—they allow to map business-level guidelines to rules that are interpretable and actionable for machines. When applied periodically, ECA rules form loops.

The simplicity of ECA rules (which fit the "input, processing, output" paradigm of computing in general) is good and bad for automation. It is good, because it can be well implemented on machines using three components: 1) an event listener (polling based or interrupt based) for the trigger event, 2) a condition evaluation component, as the condition is expressed mostly in a machine-computable way, and 3) an action handler. The bad side about the simplicity of ECA rules is their non-adaptivity. Once a more complicated behavior is desired, this either means an explosion of the number of ECA rules (with essentially overall behavior resulting from the individual actions) or much complexity in the "action" part of the ECA rule, because in these cases, the action parts needs to include all necessary additional MAPE-K steps. Additionally, for the selection of the applicable ECA rules in a certain situation, there's a variety of ways to go (first match, best match, all matching). Eventually, here a tie-breaker is needed.

Researchers at Imperial College have created a basic building block for policy-based management called self-managing cell. In addition to network management and ubiquitous computing, they applied their work to scenarios in the military and in medical care. For more information see [SFLS09] and cited previous work.

For policy-based IT management in a more general sense see [Dan07, Kem04].

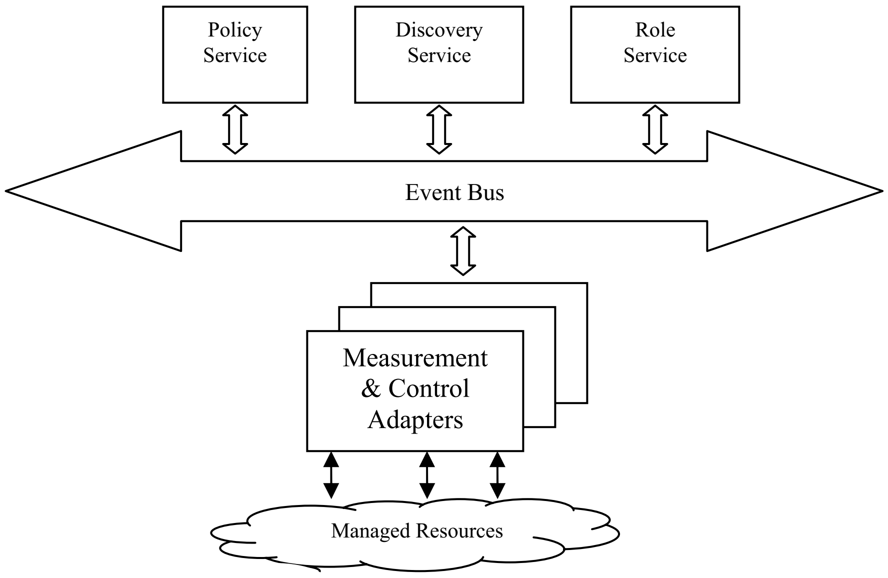


Figure 4.5: Basic self-managed cell as an example for a policy-based loop. [SBD+05]

4.3.5 Cyclic application of fix-point operators

In contrast to the periodic application of ECA rules, which could lead anywhere in the state space, fix-point operators imply that they change the system state to a known (fix) point state, which of course should coincide with the desired state, regardless of the current state. If these operators are imagined as "repair actions" that fix normal life's growing entropy and deviations, then periodically applying them means to manage a set of resources. Fix-point operators very much support the loop idea. In contrast to improvement loops however, fix-point operators do not really lead to improvement beyond the fix point, besides administrators reconfiguring the fix-point operators and the schedule, to change at what points in time they are to be triggered. Only by added human improvement, fix-point operators become similar to cognitive loops. The cognitive part is in the staff writing the rules and run schedules, the loop part and periodic application is performed by the system.

Burgess et al. [BC06] have transformed their ideas and concepts into an open-source management software called `cfengine` (for configuration engine). This is more than a simple `cron`-based task planner, the real value is the expressive and useful set of actions that can be used inside `cfengine` scripts. cfengine is based on the concept of promise theory.

In `cfengine`, essentially MAPEK is reduced to a constant action for very simple management tasks, such as editing files, deleting files, checking access control on files. The strength is in that administrators do not need to define, what normal is. Instead they write down the actions to come closer to the normal state. This is natural to them because fixing things and bringing them back to a known good state is their every-day work. What changes is, that they now need to write the rules in a way, that no harm is done if the actions are run in a state where everything was perfect before anyways.

4.3.6 Schedule-based automated procedures

Schedule-based automation is the basis for many other of the more advanced and sophisticated types for automation. Its basic function is to execute certain actions at pre-known points in time, in most cases periodically. So they form loops as well. In comparison to ECA rules, the triggering event is a timer event, the condition is always assumed to evaluate to true and the action is executed.

The simplest form is the `at` command in common operating systems that execeute an action once at a later point in time. This way, an administrator does not have to check the wall clock time over and over but instead can schedule the actions for later execution.

An advanced application is the `cron` system available mainly on operating systems with a UNIX background. Here, a table-like `crontab` file (or set of files) (see Fig. 4.6) specifies the trigger points (with explicit periodicity) in time and the actions to be executed.

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || \
                                    ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || \
                                    ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || \
                                    ( cd / && run-parts --report /etc/cron.monthly )

# TAR backup
0 4 1,15 * *    root    /usr/local/mnm/sbin/mkbackup root boot dev| 2>&1

# Create links in soft/bin
0 6 * * *       root    /soft/IFI/org/soft-adm >/dev/null 2>/dev/null
```

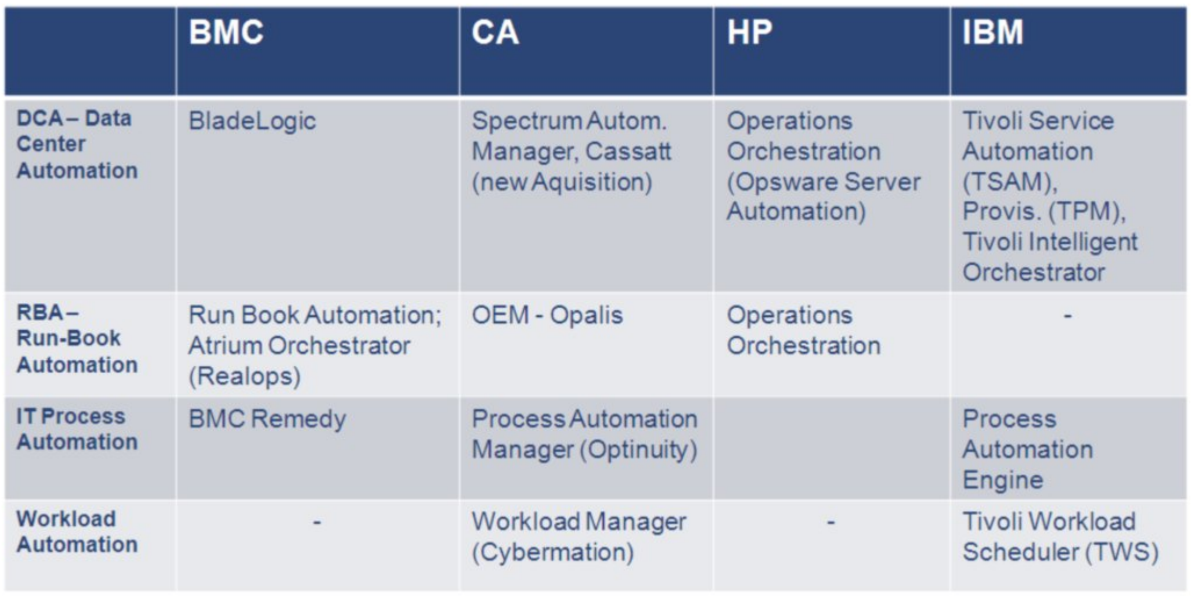
Figure 4.6: A real-world `crontab` example. The format is min, hour, day, month, day of week, user, command. Here, first there are recursive calls to periodical scripts, for example the hourly script is run at 0:17, 1:17, . . . , the daily one every day at 6:25, the weekly one every Monday morning at 6:47, the monthly one every 1st of the month at 6:52. The tar backup every 1st and 15th of the month at 4:00.

Finally, commercial management automation tools, (see Fig. 4.7 for an overview) build upon the schedule-based automation principle but add features like pre-defined action sets, a graphical user interface, logging an debugging. They are rather workflow based, but the loops form when the workflows are executed periodically.

4.3.7 Feedback control loops

Feedback control loops are known from systems engineering, where they are used extensively for well specifiable and mathematically modeled operations. A single input, single output control loop is shown in Figure 4.8. The typical arrangement of the boxes in control theory is different, but here it has been redrawn to align to the MAPEK structure. As emphasized by this layout of the boxes in the figure, control loops themselves can be used as target systems forming cascades of control loops. This matches the hierarchical composition of loops described before. New is the explicit reference input, which is the goal of the controller.

Hellerstein et al. [HDPT04] have chosen to use feedback control loops over MAPE-K loops, as control loops come with a whole set of theory and mathematics, which can be applied as soon as IT management problems are modeled as control loops. This allows predictions about the behavior (SASO properties: stability, accuracy, settling time, overshoot) even before the loops are put in action.



	BMC	CA	HP	IBM
DCA – Data Center Automation	BladeLogic	Spectrum Autom. Manager, Cassatt (new Aquisition)	Operations Orchestration (Opsware Server Automation)	Tivoli Service Automation (TSAM), Provis. (TPM), Tivoli Intelligent Orchestrator
RBA – Run-Book Automation	Run Book Automation; Atrium Orchestrator (Realops)	OEM - Opalis	Operations Orchestration	-
IT Process Automation	BMC Remedy	Process Automation Manager (Optinuity)		Process Automation Engine
Workload Automation	-	Workload Manager (Cybermation)	-	Tivoli Workload Scheduler (TWS)

Figure 4.7: Applications for IT management automation based on schedules and workflows (see www.hcboos.net).

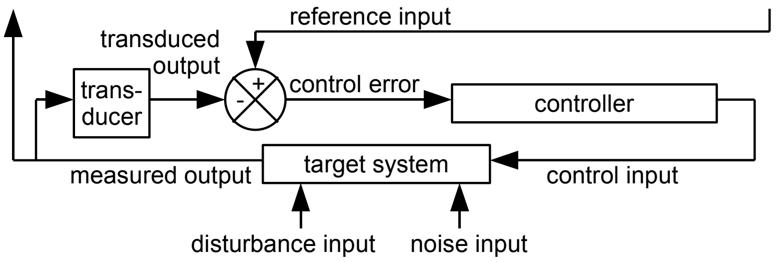


Figure 4.8: Block diagram of a feedback control system (Source: [HDPT04]). The elements in the loop have been rearranged by the author of this work to show the alignment to the MAPE steps.

4.4 Existing knowledge about loops in related domains

4.4.1 Quality management

PDCA cycle by Deming The Deming cycle, also known as Shewhart cycle, (see Fig. 4.2 on 97) describes a continuous improvement of processes. Unlike the other cycles, it essentially covers two iterations: plan the changes to or the objectives of a process, do at small scale, check/study if successful, act as necessary: if successful, do at large scale and standardize, if the desired objectives are not met, repeat planning. It is interesting that the PDCA cycle does not inherently start with observing what the current situation is, as do all the other loops, but instead starts with planning a change or objectives. This expresses a strategic attitude.

The PDCA cycle by Deming in process improvement has already been used in IT service management, and can certainly well be taken over to IT management automation. The two-step approach (do at small scale, check, do at larger scale) is already common practice in IT management with a one, some, many approach, but at a resource-level of abstraction. This means to start and test changes (such as software patches) in a test lab on a test machine, then in a midsize environment, before deployment at large scale.

DMAIC cycle in Six sigma Six Sigma is a method for quality management that defines a cyclic procedure to measure and compare the quality of services or manufacturing processes. This loop consists of five phases: define (the goals and the process), measure (relevant properties of the current process), analyze (the interdependencies), improve (the process based on the analysis), control (continuously monitor for deviations).

The definition step is one that is asserted by other loops, but not explicitly mentioned, measure, analyze, improve are the same as monitor, analyze, execute, and finally, the control part is a second long-term loop. Seen this way, the loop itself adds not much new, apart from necessary definition at the start and continuous control once implemented.

4.4.2 Business management

Cycle in decision support systems Decision support systems are designed to rationalize mostly business decisions. For this purpose, Mora et al. have also proposed a loop at the decision-making level, see Fig. 2.4 on page 34. In comparison to MAPEK: “Intelligence” matches monitoring and analyzing certain operational processes, “design” matches planning, “choice” is a new step that selects a certain design/plan, “implementation” matches execution. Learning matches knowledge acquisition.

The most interesting aspect of decision support systems is the distribution of tasks: for decision makers, others collect and aggregate data, others pre-analyze the data and may provide forecasts on the different alternatives, the decision maker (a single person or a group of persons) is supposed to decide on the matter based on the information input, but also strategy, objectives, policy and intuition. Finally, the decision is implemented by others. With respect to automation systems, we can conclude that this decision step is the most likely one, that humans would like to keep. As we will see in the next chapter, the details of this decision step will also have the main impact on user acceptance.

The typical result of decision support systems is an enterprise dashboard / cockpit, where business managers have an aggregate view on the state of processes and variables in the managed enterprise.

4.4.3 Human Factors Engineering

Cognitive loop as scheme for task analysis (Miller, 1962) Miller suggests a simple cognitive loop (see Fig. 4.9) as the basis for task analysis. He advises to do task analysis along this scheme, even though he had in mind much simpler aids than assistant systems. To improve retention of task information to memorize more system variables for example, he thinks of "an instrument or note-pad".

As we can see, it matches the MAPEK steps. "Reception of task information" is monitoring, "interpretation and problem solving" is analysis and planning, "motor response mechanisms" matches execution. "Retention of task information" matches the knowledge part. "Goal orientation and set" matches the goals given from upper instances.

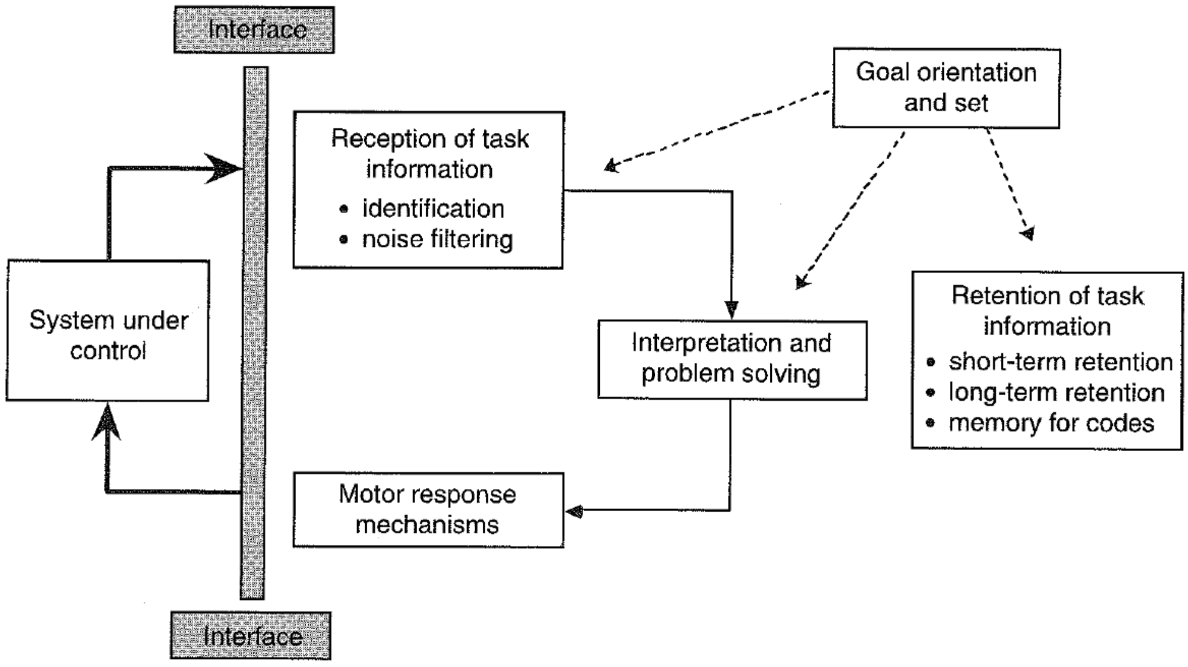


Figure 4.9: Loop by Miller for cognitive task analysis (see [WC05])

OODA loop (Boyd, 1970's) In the 1970ies, John Boyd used the OODA loop (see Fig. 4.10) in briefings on military strategy (unpublished), and especially fighter pilot strategy. In this loop, each of the two opponents in a fight between jet fighters over and over observes the surroundings as well as his opponent, orients not only within space but as a complex result of his background and experience, then decides and finally acts. In his theory, he includes the proposal that these steps can be done very rapidly, creating plans quickly and making it hard for an enemy to adopt to feint maneuvers, delusion and intentions to mislead. As such, he includes his wishes on not necessarily a heavy and most powerful fighter jet, but a dynamic one, that allows sudden moves and excellent controllability. With his description of the OODA loop, he influenced the design of fighter jets like the General Dynamics F-16.

At first sight the OODA loop is very similar to MAPE. With the exception of deciding vs. planning, the words only seem to be different in syntax, not semantics. In general, Boyd uses a totally different setting though. He considers competing humans running the cognitive loops instead of cooperating entities. He creates shortcuts in the loop to gain quick adaptation speed. There's simply no time for an extra planning phase, but the orientation includes analysis and planning (plan synthesis). In contrast,

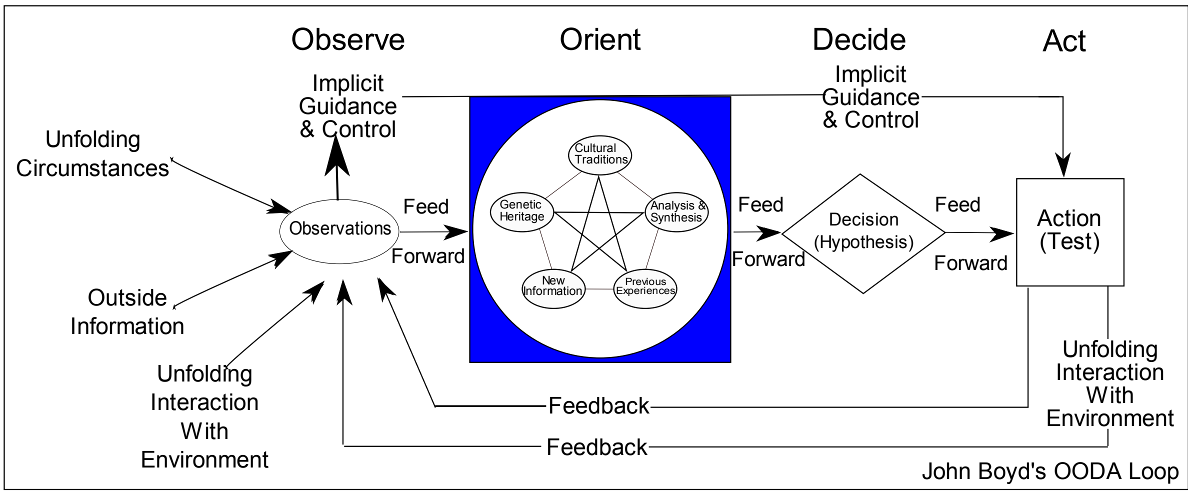


Figure 4.10: OODA loop by Boyd (see Wikipedia, OODA Loop)

the decision itself is modeled as an extra step.

The OODA loop is not a technical one, but a human-performed one.

AADI loop (Sheridan, 2002) [She02] describes four stages of a complex human-machine task. “Acquire information” is a monitoring step. “Analyze and display” matches analysis, the display action is intended for the human, as a machine does not need a display. Then “decide action” does not match a step in MAPEK but the decision in OODA. “Implement action” matches the execution step. Knowledge is not made explicit, but assumed.

Sheridan puts particular emphasis on the matter, that not all steps need to be automated to the same depth. See the next chapter for levels of automation.

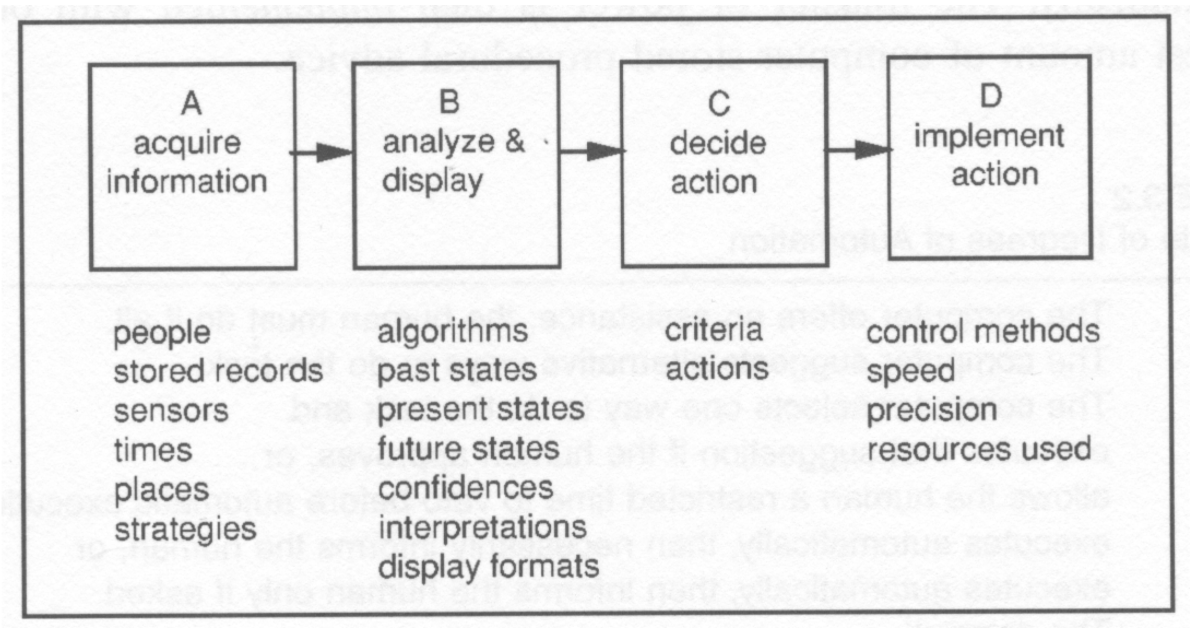


Figure 4.11: Four stages of a complex human-machine task (see [She02])

4.4.4 Medical science

Periodic SOAP notes At first sight, medical science may seem unrelated to IT management. But we can recognize the relationship, that both domains deal with treating incidents or problems in complex systems: medical doctors deal with patients and irregularities in humans, IT administrators spend a great deal of their time on the anomalies in IT systems, in addition to tasks like provisioning and improvement tasks.

SOAP (subjective, objective, assessment, plan) notes are used in medical care to structure the documentation in diagnosis and treatment of patients. First, there is a subjective part, that describes the status of the patient in his own subjective terms, potentially added by medical facts of his relatives. Then, an objective parts adds the results of measurements and laboratory tests. The assessment part contains the diagnosis, derived from the history and objective data, with symptoms and likely causes, usually ordered by likelihood. Finally, the plan part contains the derived planned actions and therapy. As SOAP notes (see Fig. 4.12 for an example) are taken repeatedly and the planning part of former iterations will have effects on the input of later one, they form feedback loops.

In problem oriented medical records (POMR) this system is advanced to include standardized tags (numbers and keywords) to later make the data more accessible to statistical evaluations by machines.

```
Patient Name: Lisa Smith DOB: 2/3/1955
Record No. B-584uw607
Date: 08/11/2004

S---Patient here for 6 months. follow-up visit, no complaints.
       no known drug allergies, allergic to latex
O---blood pressure 142/88; Atenolol 50 mg daily
A---hypertension controlled
P---continue Atenolol; return to office in 6 months
```

Figure 4.12: A SOAP note example

In comparison to MAPEK, in SOAP subjective and objective match monitoring, assessment matches analysis, plan is the planning part, the execution is silently assumed to be performed by medical staff, knowledge is assumed to be with the doctors, and only indirectly expressed in the assessment and planning part, where the doctor is supposed to reason based on his knowledge.

SOAP adds one new aspect: Different to all other loops described in this chapter, there is an explicit distinction between subjective and objective monitoring input. We should consider taking over a similar scheme to IT management automation, where for example for efficiency reasons the administrator is asked for this opinion of the most likely causes of an incident which are then checked first. Before a rule-based or knowledge-based reasoning system without much guidance searches a vast space of causes and effects only to confirm the administrators intuition. We should however, keep the distinction of subjective from objective knowledge. e.g. with metadata.

4.4.5 Systems engineering

Systems engineering can be performed at multiple levels of abstraction as well as the other domains.

Reinforcing and balancing loops Meadows [Mea08] wraps up systems thinking and systems theory. There, (complex) systems are modeled with loops at a high level of abstraction for analysis.

To reduce the necessary detail in modeling and still have a chance to predict the essence of overall system behavior, they use influences between entities which form two kinds of loops: reinforcing loops (these are loops where some parameter will increase/decrease more and more, as the loop is repeatedly executed) and balancing loops (here the loop approaches some desired value for a certain parameter). See Fig. 4.13 for an example of an intelligent heating system. If then the growth of reinforcing loops is qualified (e.g. linear, polynomial, exponential) similar to big O notation and the behavior of balancing loops is qualified regarding its rate of adoption, the resulting behavior of simple loop constructs can be estimated.

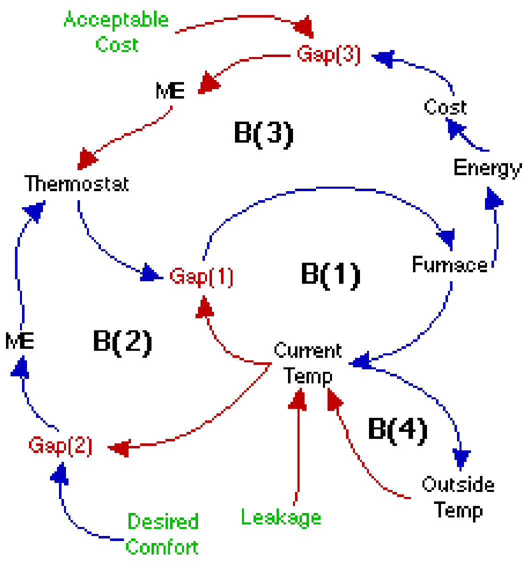


Figure 4.13: A system model for an intelligent heating system based on desired comfort and acceptable costs. Here, four balancing loops interact with each other. Source: `http://www.systems-thinking.org/vossov/vos.htm`

Loops in RCS (Albus/Barbera, 1993) In [AB05], Albus describes the evolution of four generations of a real-time control systems (RCS-1 to RCS-4). There, technical feedback loops are presented as functional boxes, which show ambitions for considerable automation in the resulting embedded systems approaching autonomy. As the author states himself in [AB05]: “RCS has evolved through a variety of versions over a number of years as understanding of the complexity and sophistication of intelligent behavior has increased.” See Fig. 4.14 for an illustration.

RCS-1 was intended for a single robot arm with a vision system. It overlays external commands with feedback from the vision system. It is based on state machines that have a response for every combination of command, current state, and sensory feedback. This essentially is a rule-based system.

RCS-2 adds a function G for sensory image preprocessing and analysis. Planning does not take place, the execution in H still relies on a state transition table. Its inventors also created a hierarchy with 8 control levels: Servo, Coordinate Transform, E-Move, Task, Workstation, Cell, Shop, and Facility. The first six were actually built. Applications were projects on an automated manufacturing workstation, a field material handling robot and a semi-autonomous land vehicle.

RCS-3 then adds global memory, and an operator interface. It renames H to a task decomposition function (TD), and uses a world model for task planning and model-based sensory processing. It was

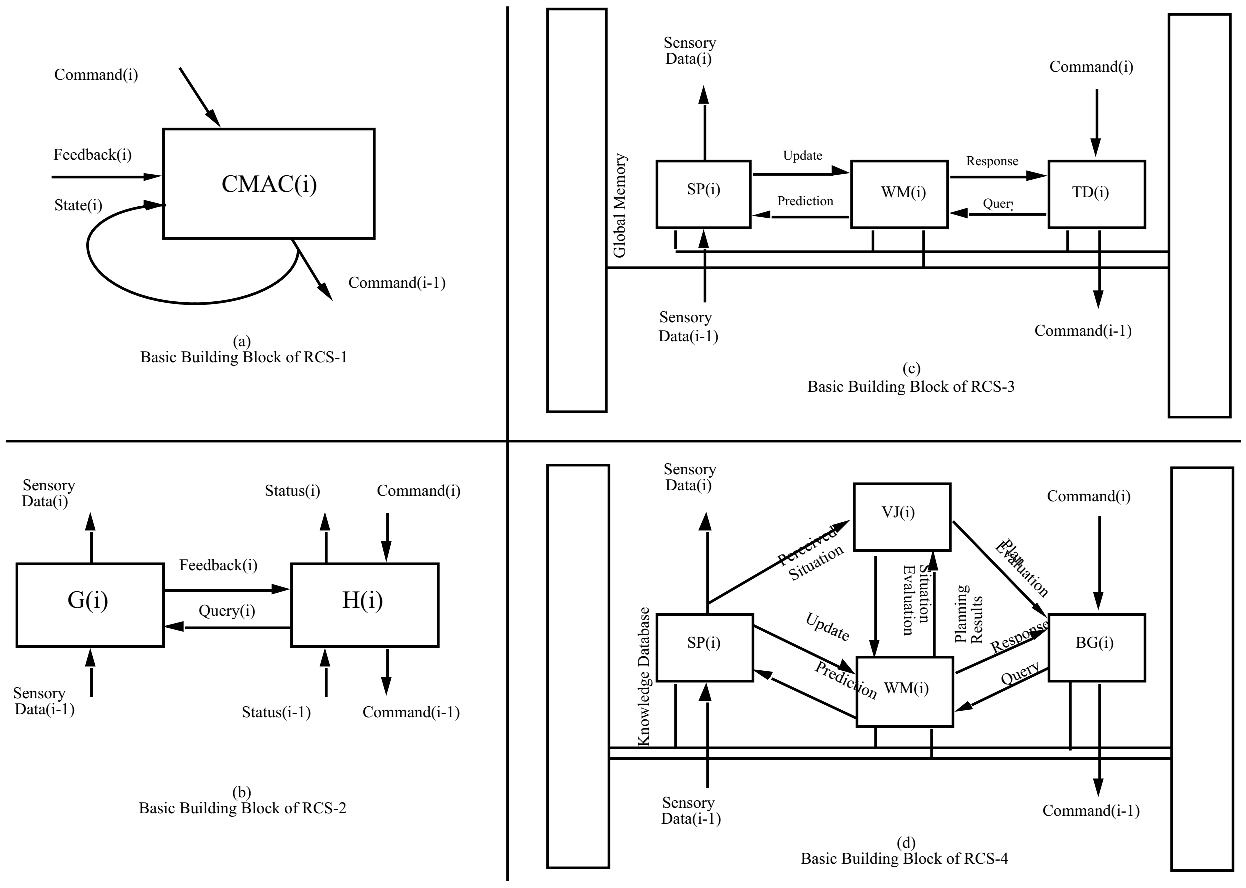


Figure 4.14: Basic building blocks in different generations of RCS systems. [AB05]

designed for multiple autonomous undersea vehicles and adapted for a telerobot control system. This loops matches the MAPEK loop, when we abstract from name differences.

Finally, RCS-4, under development at the NIST robot systems division in 1993, adds an explicit representation of a value judgment function (VJ). It is based on results in brain research. To prepare for learning and to improve planning, this component contains processes that compute cost, benefit, and risk of planned actions, and that place value on objects, materials, territory, situations, events, and outcomes. Value state-variables define what goals are important and what objects or regions should be attended to, attacked, defended, assisted, or otherwise acted upon. In addition, the TD function has been refactored to behavior generation (BG) to emphasize the degree of autonomous decision making. Global memory has been advanced to a knowledge database. RCS-4 was intended to be used for highly autonomous applications in unstructured environments, where high bandwidth communications are impossible, such as unmanned vehicles operating on the battlefield, deep undersea, or on distant planets.

The interesting aspect about RCS is the *development* of the component blocks and its relationship to brain research. We can see, how each new version was used for more and more demanding applications. And as more and more compute power and memory became available, more and more features were added that we can relate to artificial intelligence. As this development was performed until 1993, we can assume that latest developments in robots (such as self-driving cars) have even improved upon this set with functions like machine learning. An interesting function is “value judgment”, which can probably approach in a technical way the difference in semantics between subjective and objective information, that had been present in SOAP notes.

The RCS nicely shows how its authors started by off-loading physical workload to robots, and then progressed by more and more automating the mental workload of system administrators and operators. With applications in a domain such as remote space missions, they were forced to increase system autonomy not for economic reasons, but for its necessity.

Loops in DASADA In an approach to come up with a component architecture that adds “intelligent” behavior to existing legacy systems, the “Dynamic Assembly for System Adaptability, Dependability, and Assurance” (DASADA) reference architecture [PKGV06] has been created at the Air Force Research Laboratory. DASADA attaches to the managed system via sensors and effectors, as we have seen before, but adds gauges (similar to Sheridan’s displays) and controllers (as in control loops), see Fig. 4.15.

New is, that the feed back is performed not only via the external controlled system, but also that the controllers can directly reconfigure the sensors, gauges and effectors. This overall construct can be broken down in two ways: 1) if we leave it as described in DASADA, then there is self-modification within the loop itself. Or 2) We could breakdown the DASADA loop into multiple hierarchical loops.

Feedback control loop Control loops, see Fig. 4.16 for an example, are the basic element of control systems in engineering across all application domains from heating systems to flight control systems.

Along this illustration, we can compare a control loop to a MAPEK loop: The target system is the managed entity. But different to the other loops, here the external noise and disturbance is pictured explicitly. The control loop now uses sensors (not shown, may coincide with the shown transducers), both of which can be regarded as monitoring. Then analysis takes place with a comparison of the reference input (also made explicit here) with the transduced input. The resulting difference from this comparison (the control error) is then used by the controller to perform further analysis and execution,

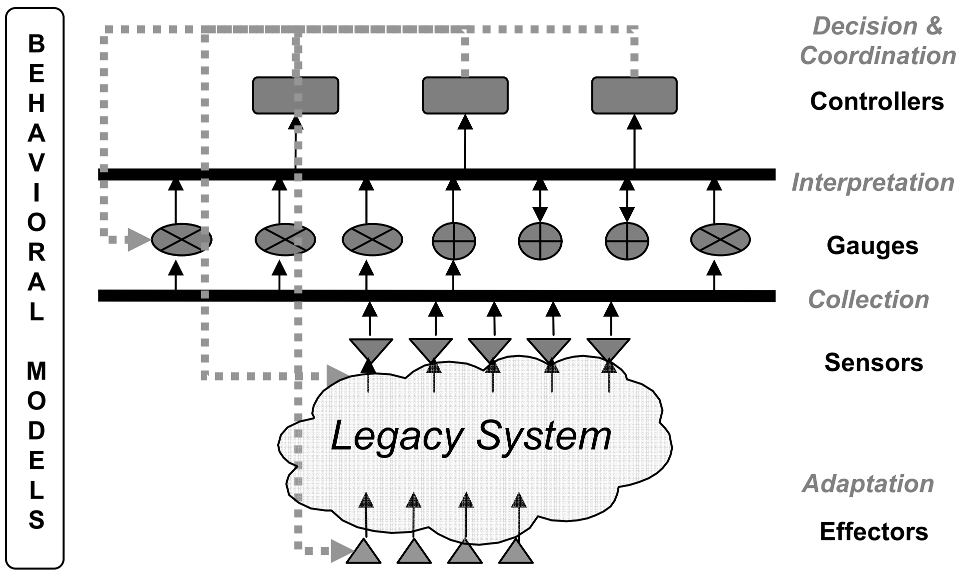


Figure 4.15: DARPA Dynamic Assembly for Systems Adaptability, Dependability, and Assurance (DASADA) reference architecture (see [PKGV06])

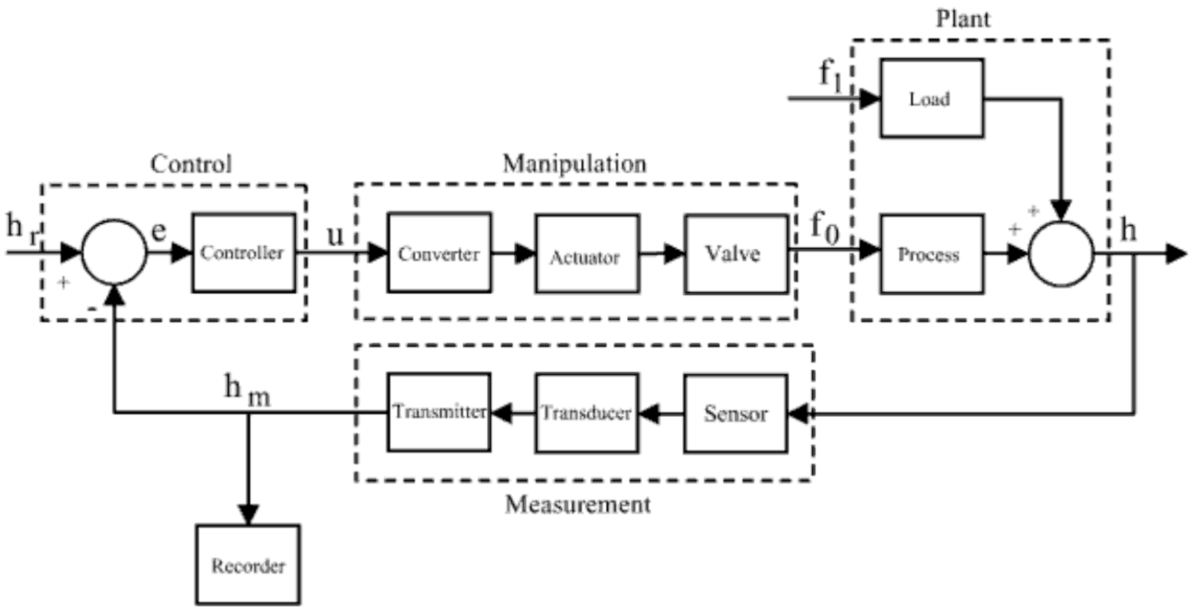


Figure 4.16: Example comprehensive control loop of a level control system(see [Lov07])

usually based on simple equations according to control laws. The control input is finally implemented by actors. Planning usually does not take place.

Simple controller types use simple control laws (on-off control, linear control, such as proportional control, integral control, derivative control, or combinations such as PID control) to derive the control input from the control error. In early controller layouts the control logic itself was based on very simple electronics. For these controllers with their relatively simple control laws, control theory has come up with a range of theoretical consideration on the resulting behavior, when control loops are composed. As microcontrollers became cheaper and more powerful, and fuzzy logic controllers and other types were introduced.

Control loops are designed for a certain objective: 1) regulatory control. This means that the measured output should match the reference and the reference is to be tracked when it changes. 2) disturbance rejection. Here, the focus is on changes in the disturbance input. And 3) optimization. Here, the controller is designed to maximize (or minimize) a certain measured output.

Other mathematical models for automation Other modeling techniques that would also allow to model loop-like structures, however at a level of states and transitions, are: Petri networks, Markov chains, finite state machines (FSM). See [Lun06, Lun07] for an overview.

In these loops the MAPDEK steps would remain as pure annotations.

4.5 Requirements Check

In the following subsections, we investigate the aforementioned loops along the three requirements stated at the top of this chapter.

4.5.1 Regarding R2.1 Loops must be usable at multiple levels of abstraction

Regarding their distinctive features in the level of abstraction, two main aspects can be extracted, where the loops have commonalities and differences: the proximity to resource-related tasks, and the mathematical formality. An overview on the evaluation results is shown in Tab. 4.1.

Loops by level of resource proximity

Due to IT management ranging from technical management to operative management/business management, and the breadth of the inspiration domains, regarding the level of abstraction we can roughly differentiate between loops closer to the business manager level and loops closer to the managed systems level.

Loops close to business The following loops belong to this category: improvement cycles in ITSM frameworks in IT management, and PDCA, DMAIC (Six Sigma) and the outer loops in (business) decision support systems. These loops are close to strategic management and process improvement. Typically, they are expressed in a verbal manner, not in a formal, mathematical way. They are at an appropriate level of abstraction for the business managers side of customers in management automation systems.

Table 4.1: Loops by level of abstraction

	Level of abstraction of the modeled loops	**Mathematical formality of the loop model**
IT MANAGEMENT		
Improvement cycles in ITSM frameworks	business, process improvement	informal
Cognitive loop in FOCALE architecture	variable, resources and loop orchestration	informal
MAPEK loop of the ACI	variable, resources and loop orchestration	informal
Triggered application of ECA rules	variable, machine rules, business rules	semi-formal, policy languages exist
Cyclic application of fix-point operators	resources, repair actions	semi-formal, with theoretic background
Schedule-based automated procedures	resources, calendar	semi-formal, form-based, math. informal
Feedback control loops	resources, simple actions	formal, application of control theory
RELATED MANAGEMENT		
PDCA cycle by Deming	business, process improvement	informal
DMAIC cycle in Six sigma	business, process improvement	informal
Decision support systems	business, general business management	informal
Cognitive loop as scheme for task analysis	human cognition	informal
Observe, Orient, Decide, Act loop	human cognition, quick response	informal
Acquire, Analyze, Decide, Implement loop	human cognition	informal
SOAP notes	resources (patients), process documentation	form-based, but mathem. informal
System theory - Reinforcing and balancing loops	variable, general system modeling	informal, trend indicative, composable
Loops in RCS	resources, mid-level control	semi-formal, system architecture level
Loops in DASADA	resources, mid-level control	semi-formal, system architecture level
Feedback control loop	resources, low-level control	formal, with extensive control theory

Loops close to managed systems Loops close to resources are close to the actual technology like servers, network equipment, software, airplanes in aerospace, or patients as “managed objects” in medical science.

They are much more concrete than the higher level loops, and often come with a theoretic background or concrete existing software applications. Among them are feedback loops with their control theory and applications in mostly embedded systems or specific software applications, RCS and DASADA as concrete reference frameworks building upon them but liberating from the narrow standard forms of control loops, and in IT management: fix-point operators (e.g. `cfengine` software) and schedule based automation (e.g. `cron` and datacenter automation systems).

They are at a known level of abstraction for all involved people, that have a good understanding of the managed resources, so mainly system administrators, and system developers. They are less applicable to the business management side at the customer, who may be the primary party interested in automation.

Loops between business and resources Recognizing that none of the former is truly appropriate for both sides, technical management and higher level management, hybrid approaches cover aspects of both. In IT management: MAPEK and FOCALE include matters on resource management and loop orchestration aligned to business goals. ECA rules aim to translate business rules into machine rules for policy-based management. Also in systems design and analysis, there are loops at such a higher level of abstraction: systems theory and systems thinking abstracts away most details and only keeps the information on the general trends caused by loops.

These loops with a variable abstraction level best fit the multiple levels of abstraction, that are required above.

Loops close to human cognition These loops (Miller, Boyd, Sheridan) all concentrate more on human side of human-machine interaction. With a background in human-factors engineering/ergonomics, they stress two aspects, that we should also keep in mind in IT management automation systems: the later system operators/system administrators do have their own mental loops. IT management automation systems need to be designed in line with them. For related work on matters of psychology in IT management, see papers published by the team headed by Paul P. Maglio (Head of Service Systems Research, IBM Almaden Research Lab), for example [KHBM09, KBMH08, BMKB05, BKM+04].

Work on psychological matters in IT administrators was later reflected e.g. in the inclusion of limited system operators memory in the Change Management with Planning and Scheduling (CHAMPS) system [KHW+04]. Or the Eclipse IDE software, where a whole range of standard checks are made over and over again to lead the developer to good, consistent software and to support all operations within Eclipse.

Loops by mathematical formality

Mathematical formality is necessary in technical systems

As can be seen in Tab. 4.1, mathematical formality is another differentiating property of the loops.

Abstract, verbal, commonly understood Informal descriptions are usually verbal, versatile and flexible. This can prove to be a negative property, when for example many loop models exist, that are semantically close to each other. Or because of its lack of mathematical definitions, equations, theory, standard variable names, that render mathematical derivations, proofs or forecasts impossible. At the same time, however, the same informality makes it much easier for people to communicate about a problem, and ideas for its solutions in general at a more abstract level.

In the loops, we can recognize informality in all business-related and human-related loops. The loops with a variable level of abstraction are mostly informal, with some optional formalities, like policy languages to formalize policies.

Detailed, mathematically exact Loops closer to the managed resources have a higher level of formality. While some of them are marked as informal. Usually, data formats exist, but not much theory to base validations on. This is different for control loops. Here, we see the benefit, that the class of control loops cover all mathematical aspects at this layer and do not allow many more competing loops with similar mathematical content. For certain types of control laws, control theory can predict aspects on the behavior of the loops. Generally speaking: formality only eases the communication among people who understand the formality and its relationship to the real world. This may not always be the case for the people involved. For simulations, formality is definitely a necessity.

Disadvantages of different loop models

Theoretically, we have three levels of abstraction for the tasks and loop steps and we could use individual loops on each level:

- loops at business/process improvement abstraction level: process improvement loops from an ITSM framework
- loops at intermediate/process abstraction level: MAPEK
- loops at resource abstraction level: feedback control loops, Petri networks, FSM, ...

However, modeling the loops this way would have the following negative consequences:

- Loops at the process improvement abstraction level could get too abstract for an automation engineer.
- Loops close to resource abstraction level would have to be described in a very detailed manner, refining them from the high-level structure proposed in Chapter 3. This would mean a high effort for modeling, and a loss of flexibility to adopt other ways of modeling. E.g. it is unlikely, that a task laboriously refined to an finite state machine would later be replaced with a more sophisticated and adaptive AI method. In the transition the former effort may pay off, but it may also occur, that a different model needs to be created to make the AI method suitable for the task.
- Uncertainty: EIMA is used early in system design of an IT management automation system. At this point it is not even known for certain tasks, at what level of abstraction they should be modeled to later fit its position in the task hierarchy.
- Heterogeneity: Due to the three kinds of models, we could not treat all loop models equally and would have to investigate how to link between them. For each loop model we would have to define automation levels, and an implementation pattern catalog.
- Lack of flexibility in automation: If we put each task in one of this these three categories, the freedom of choice regarding the actual level of automation (assignment to human or machine

would be pre-chosen here. As we can hardly expect humans to align their work to a finite state machine, for example.

Recommendations

Take one of the loops for EIMA, that have been marked as variable in Tab. 4.1. This applies to FOCALE, MAPEK, ECA rules, and the reinforcing and balancing loops in systems theory.

4.5.2 Regarding R2.2 Commonly structured base loop

So far, every layer of abstraction uses its own base loop so far (see before) hardly any relationship has yet been seen between them. As a result of the loop comparison, Figure 4.17 aligns the loop models from Engineering and IT management regarding their semantically common steps. Many of the comparisons with MAPEK were included with the descriptions of the individual loops. To recognize the loop structure in the graphics, the individual lines should be read starting on the left hand side then proceeding to the right hand side and wrapping around to the left side for the next iteration.

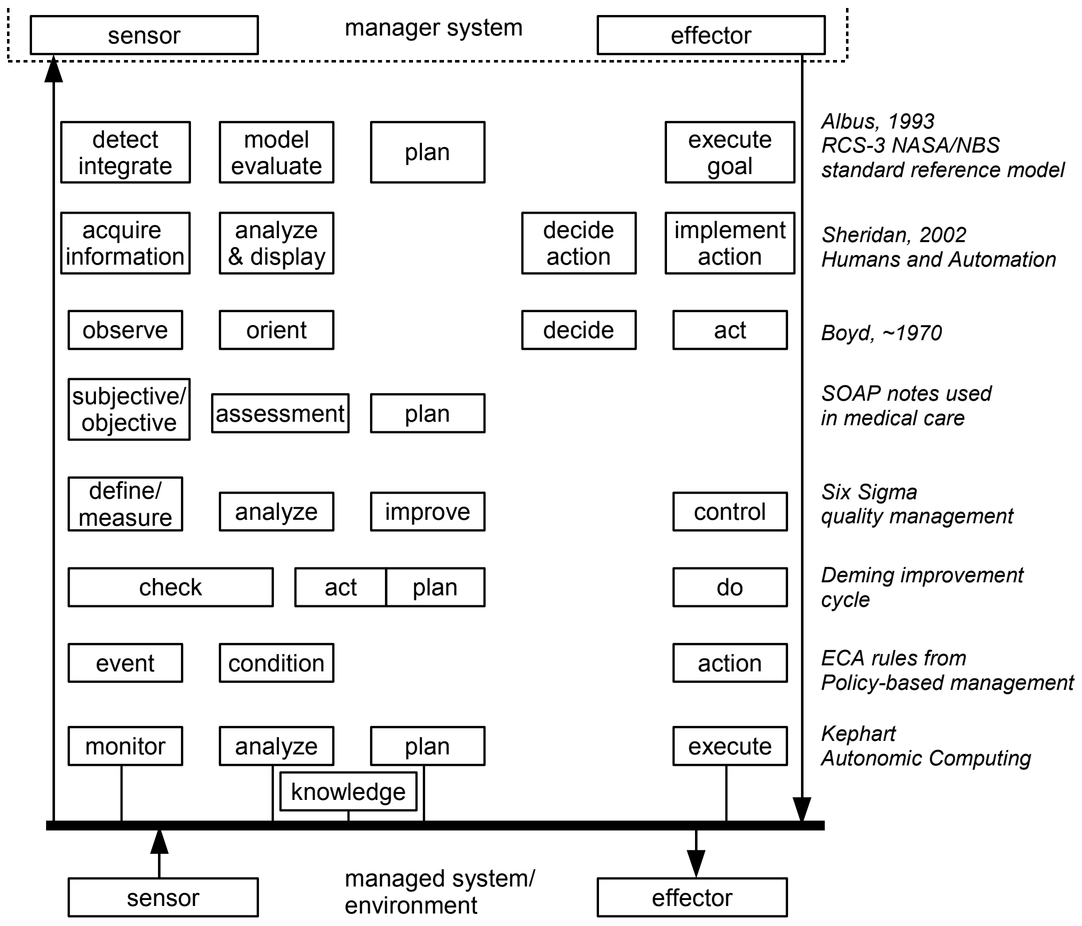


Figure 4.17: Automation loops in IT management and Engineering show a good alignment in general. Current loop models in IT management however lack the “decide” step. As a result of this comparison, this work will add a decision step to MAPEK and use the MAPDEK loop.

As can be seen, the loops show remarkable similarities in their individual steps as patterns to decompose tasks. By nature, loops match the repetitive character of management tasks. Still the level of abstraction of these loops fills a wide range from 1) control loops in engineering to 2) abstract improvement cycles in IT management. The execution frequency differs from thousand times per second in control loops to once every several months in typical process improvement cycles. The different kinds of loops are all similar, so that concepts can be transferred between domains.

Regarding the number of steps, the loops differ however. During the introduction of the individual loops, we found a couple of loops that do align with MAPE very well and that have a similar task breakdown, e.g. RCS, Sheridan's loop and Six Sigma. Others are similar, but they group certain steps and do simplify the loop structure, such as ECA rules, fix-point operators, schedule-based automation calls, and control loops. For a base loop to align knowledge on implementation methods to, they are therefore less applicable for the reason of a lack of fine-grained steps.

Recommendations

For EIMA, we can reduce the syntactic overlap that partly exists in IT management, and partly resulted from the cross-domain view and instead name a common model. For the overlapping MAPEK steps, we argue to stay with MAPEK, as it was established in the ACI (the closest relative of EIMA in the table) and fits IT management automation. Regarding these steps, yet another mixture would enhance heterogeneity without any purpose.

It's visible in this comparison, that some domains with critical decisions—decision support systems used for demanding business applications with long cycles, Sheridan's model of a complex human-machine tasks, and OODA for advanced technical embedded systems with quick reactive control—have added the "decide" step as main key for switching between levels of automation. Levels of automation, or "adjustable autonomy" as it was called in other domains, is the difference in the assignment of tasks to either the machine or a human. It is very important to achieve high utility from the management automation system, while staying in control as an admin, and therefore a key to acceptance of such systems. For this reason we include the "decide" step into the MAPEK loop in EIMA. In summary, for analysis and design purposes, we thus argue to use the MAPDEK union set of the loop steps (monitor, analyze, plan, decide, execute based on knowledge).

For each MAPDEK loop, the following aspects should also be considered: analysis of control and data flow (general properties of workflows), and timing (loop trigger and frequency). The explicit description of a loop trigger (as proposed by ECA) will improve comprehensibility. Frequency and timing are probably the most important properties, when deciding about whether to assign steps, loops, or tasks to humans or machines.

Regarding dynamic self-adaptation, the MAPDE steps should stay as they are algorithmically. Only the knowledge repository is intended to be changed which then may influence parameters used in the MAPDE steps. The advantage of this approach is, that the MAPDE steps can be identified via a version tag and a central repository. This way, only the knowledge repository inside the loops is data, that needs to be memorized in its entirety. Large chunks of it, e.g. tables, models, and graphs, can certainly be referenced as well, being equal for many loops, the rest can be considered configuration data of the loop and log and historic data. Quick and relatively easy identification of a loop and all components is a prerequisite for tracing, debugging and verification.

Non-Recommendations

Multiple additions would have be possible from the other loops: A "define" step from DMAIC seems unnecessary, as it's clear that loops need to be defined first. The additional "subjective" monitoring input (as proposed in medical SOAP notes) from administrators may be useful at a later stage, but we first stay with the "objective" technical monitoring input to concentrate on a core set, and essentially it's more a different input in the monitoring step than a new step. In this loop architecture, the administrator can still rely on his subjective opinion in the decide step.

Using self-modification/self-configuration within the loops (as proposed in DASADA) may reduce code size and increase adaptivity, but will have a significant impact on predictability and verifiability. EIMA loops should therefore restrain from self-modifying the code of the MAPDE steps.

4.5.3 Regarding R2.3 Machine-interpretable loop modeling and linking back to the tasks in task analysis

Of interest in the scope of this chapter is the data about loops. As with tasks, EIMA proposes to use machine-interpretable files to store the modeled loops, in a similar way to CAD or CASE tools. This means, that we need an information and data model for the modeled MAPDEK loops.

We will first investigate the previously mentioned for existing support in terms of models and then derive recommendations. With respect to modeling, we will not consider graphical elements, stencils and shapes for flowchart applications or drawing applications to draw the loops. Such exist for basically all formats, but then the information is stored at the graphical abstraction level not the logical, semantic one.

Evaluation of loop modeling

In an interest to not reinvent the wheel, we first look for existing standards to model loops, that are similar to MAPDEK loops. The intention is to find existing models (information and data model) and modeling tools.

For EIMA's MAPDEK loops only a few of the formerly mentioned loops are actually relevant regarding their tools, information and data models. As we need to model loops in IT management automation, medical SOAP notes are irrelevant, other loops are not structured appropriately. The remaining loops of interest in IT management must be close to the proposed MAPDEK steps, they are. See Tab. 4.2 for an overview on modeling standardization efforts regarding the loops. Feedback control loops have been left out in the table and are covered in more detail in the following section.

Models for feedback control loops Feedback control loops differ from all other loops mentioned before, because there is a mathematical model as the basis, see e.g. [DFT02]. This mathematical basis standardized basic models, variable names, and typical function names and types.

We mentioned above, that modeling at the level of control loops is probably not done at the right level of abstraction as a means of communicating about a management automation system to be built, if the involved people are system administrators, their business managers, and system designers. For this level of abstraction, MAPDEK loops are better suited. However, the verbal models need refinement, if they are to be used to simulate a prototypic system behavior. Here, we will need either pre-defined control loops, or need to do control loop design with its mathematical modeling, symbols, (transfer) functions to translate the functional boxes into a model.

For this purpose, there are general numerical math tools. A known commercial one is Matlab by Mathworks, together with a control system toolbox, to name only one of a whole range of application-specific . For Matlab, there is also a toolbox to link its artifacts to requirements specifications described

Table 4.2: Existing modeling standardization effort for a relevant subset of loops

	Information and data modeling
IT MANAGEMENT	
Improvement cycles in ITSM frameworks	Outer improvement loops are rarely modeled so far, they are rather an expression of business culture inside an enterprise. This may change with the introduction of not only the ISO 20.000/ITIL processes itself, but the actual process improvement loops. While there are upcoming ITSM tools, the improvement cycle needs to be added around those tools.
MAPEK loop	There is no generally agreed upon information/data model. Academic work in this direction includes the following, but did not lead to a general model: [VP07, VM09, LP05, Dow04]
FOCALE loop	Not publicly made available to the best of the author's knowledge.
ECA rules	There is no generally agreed upon information/data model. Policy languages do exist, e.g. Ponder allows to express policies in a formal language. Ponder is accompanied by a Ponder compiler and Ponder toolkit.
Periodic fix-point operators	There is no generally agreed upon information/data model. Application example: `cfengine` rule sets.
Schedule-based automated routines	There is no generally agreed upon information/data model. Application example: `crontab` and `crond`.
RELATED MANAGEMENT	
PDCA cycle by Deming	There is no generally agreed upon information/data model. PDCA cycles to improve business processes are business processes themselves, they could be modeled usind general business process modeling tools. There are applications for business intelligence (see e.g. `http://www.luenendonk.de/business.php`) and business process simulation (see [JVN06]), both of which add advanced features than just being a plain format for documenting processes."
DMAIC cycle in Six sigma	There is no generally agreed upon information/data model. With roots in statistics and quality management there are tools that support Six Sigma Processes, see e.g. `http://en.wikipedia.org/wiki/List_of_Six_Sigma_software_packages`
Observe, Orient, Decide, Act loop	There is no common model, to the knowledge of the author, general purpose modeling tools are applicable.
Acquire, Analyze, Decide, Implement loop	There is no common model, to the knowledge of the author, general purpose modeling tools are applicable.
System theory - Reinforcing and balancing loops	"There is no common model, to the knowledge of the author. Several applications on abstract systems modeling and simulation do exist, see e.g. "Extend" by Imagine That, "ithink" by isee Systems, "myStrategy" by Strategy Dynamics, "Vensim" by Ventana Systems"

in DOORS. Free alternatives include Scilab (by the Institut National de Recherche en Informatique et Automatique in France, INRIA) or GNU Octave. Typically, these tools are command-line oriented and deal with the pure numerical mathematics in formulae and plots.

To aid engineers and link mathematical models to an actual system model (e.g. a block diagram with function boxes), graphical modeling of control loops/systems has been added to the previously mentioned math-oriented tools. They provide simple visualization facilities (often based on components from a certain predefined set) and simple simulation and animations. Commercial tools include Simulink by Mathworks (as a partner of Matlab) and ASCET (Advanced Simulation and Control Engineering Tool) by ETAS Group. The later one exports to Matlab/Simulink and UML. A free alternative and partner to Scilab is Scicos, also by INRIA.

There are also lower-level tools such as Labview by National Instruments with main applications in measurement engineering, control engineering and automation. As a specialty, Labview uses data-flow oriented modeling and graphical programming.

4.6 Proposal on modeling MAPDEK loops in EIMA

After having checked related work in various domains against the criteria, that were stated at the beginning of this chapter, this section now shows the proposed approach to deal with loop analysis in EIMA. It acts on the recommendations in the requirements check section, but makes them more explicit and actionable.

4.6.1 Regarding R2.1 Loops must be usable at multiple levels of abstraction

Use the MAPDEK scheme at all levels of abstraction: from process improvement to resources!

The recommendations in section 4.5 proposed a selection of the current loops that are usable at multiple levels of abstraction. Then, the requirements check for a common base loop suggested the MAPDEK loop as a combination of MAPEK and an explicit decision step.

MAPDEK loop For the reason of homogeneity, we argue to only use MAPDEK loops in EIMA. The MAPDEK loop scheme is illustrated in Fig. 4.18.

The outside of the loop consist of an upper and lower "touchpoint" interface, via which loops communicate. This way, hierarchies of loops can be formed. A loop communicates status updates and escalations to a superordinate one. In turn, a superordinate loop communicates to a subordinate loop either via high-level parameters such as goals, policies and constraints, but can also use more explicit function calls to match lower levels of abstraction. The lowest loops do interface to the actual managed resources.

The interior of the loop is dominated by a local knowledge repository, a communication bus (shown as an ellipse), and five processing steps:

- a monitor component that receives incoming status information, escalations, and requests from the upper and lower sensor interfaces and does basic preprocessing of the incoming data.

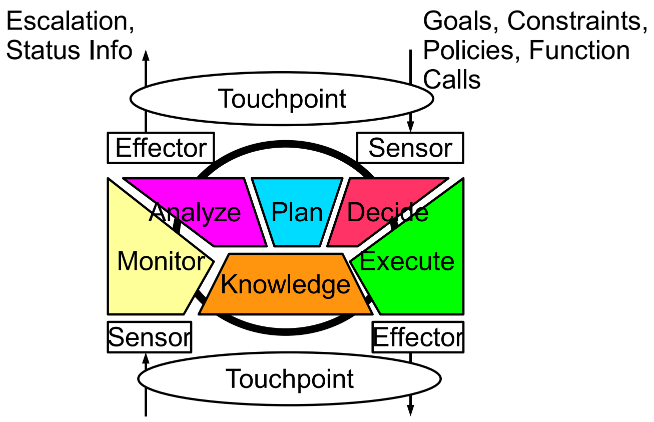


Figure 4.18: The MAPDEK loop proposed for loop analysis in EIMA combines the MAPEK loop from IBM's Autonomic Computing Initiative with an explicit decision step as a result of loop structures in decision support systems and systems engineering

- an analyzer to investigate the incoming data, this can mean reasoning, correlations, trend analyses, pattern matching, comparisons, and various other techniques, which will be detailed in Chapter 6.
- a planner component, that reacts on the analysis results. Usually it will plan changes in a certain state space, and the alternatives to reach the desired state from the current state.
- a decider component, that chooses from the available options.
- an executor, that implements and performs the actions in the chosen plan via output on the actor interfaces.
- all sharing knowledge via the knowledge repository, which holds logs, models, past decisions, and configuration parameters, to name just a few items.

The general loop structure matches the one of cognitive loops, which fits human workers and management automation systems alike. Therefore, full loops and individual steps can later be assigned to either side, see Ch. 5 for details.

Advantages of the MAPDEK loop model The following benefits are expected as a result of using the MAPDEK loop model as a common loop model:

- MAPDEK was designed in a way, so that it covers all three levels of abstraction. A consolidated modeling approach makes it easy to shift loops around in the task hierarchy. MAPDEK is close enough to be applicable to business processes, and at the same time reaches into resource-related management.
- A consolidated modeling approach allows a common approach for the levels of automation and a common approach for the implementation pattern catalog.

It is clear, that MAPDEK loops will have to be refined later at systems implementation time, but we can postpone this to the point when the EIMA method has been completed. At this point, EIMA output will have transformed the input in a way, that the applicability of automation to the loops can be inferred from the sum of all output of the EIMA steps.

Modeling with MAPDEK loops Input for loop analysis in EIMA is the task decomposition tree from EIMA task analysis. The tasks in this tree are examined for matching MAPDEK categories, for which the pattern catalog in Chapter 6 will be valuable input to know which kinds of tasks are associated with what category. Then each task is assigned to a category.

To identify hierarchies of the loops, former role assignment and role hierarchies (both being results from Chapter 3: Task Analysis) are helpful to recognize the control dependencies among tasks. The depth of the loop tree should not exceed three layers of MAPDEK loops to keep complexity manageable.

As stated before, loop analysis is not a mechanical task. Instead, it requires experience and a feeling for both, organizational hierarchies, responsibility and distribution of control to come up with a good scheme of MAPDEK loops. In the same way, the person doing loop analysis is free to change the task structure, to add, refine, remove or fold/unfold tasks to achieve a good loop analysis result. Such a result shall show a hierarchy of loops with individual MAPDEK steps.

A certain depth of loop analysis is not dictated by EIMA, as it was designed to be flexible in this regard due to its hierarchical structure for task decomposition. A depth should be used that is enough to serve as means of communication between the participating stake holders. When an EIMA automation assessment scenario is performed in a real business case, this way the depth of modeling can be chosen in line with available personnel, time, budget and task expertise.

Clearly, it will need the work of system analysts and system administrators to jointly refine the tasks to loops.

4.6.2 Regarding R2.2 Commonly structured base loop

The requirements check in section 4.5 already has motivated the use of the MAPDEK loop model.

Align the sub-tasks to the MAPDEK scheme!

With the reduction to the MAPDEK task categories we have a reasonable number of six task categories with a profound background in related work, as Fig. 4.17 indicates. Other loops show, that more steps (such as modeling, learning, or verification) can extend certain properties of a loop, and will make them more applicable to the rare systems that indeed do have such capabilities. However, at this point, EIMA—being an approach, where with low efforts we can gain a remarkable return—is based on the experience by Hitoshi Takeda ("Low-Cost Intelligent Automation") and common theories by Pareto, that a low percentage of the potentially invested efforts can result in a high return, meaning that also the activity of automation itself is efficient and effective. In other words: EIMA does not strike for the maximum "intelligence" that can be built into systems based on currently known methods, but limits its automation ambitions to a reasonable level, that is covered by MAPDEK loops.

It should be noted here, that the MAPDEK steps are not meant to be executed in strict sequential order. Instead we keep the idea of a common bus among the steps and common access to local knowledge.

When there is the need to break down the individual tasks into loops recursively, in task analysis in Chapter 3, the tasks were refined along the scheme in Tab. 3.10 , which includes the following properties of tasks: information required, decision to be made, control action required. In task refinement to loops, this already captures the steps monitoring (M), decision (D), and execution (E). Now during task to loop refinement, we add analysis (A), planning (P) and knowledge (K). The addition of these steps investigates the actual "intelligence" required to make good decisions for each situation. However, this "intelligence" can be created in a different way than formerly performed by the responsible entity.

Check the loops for completeness and redundancy!

When modeling with MAPDEK loops, a few situations can apply, that are signs of inconsistency and should be treated individually:

- *There can occur MAPDEK loops, that do not consist of all six steps, in which case they are underspecified.* MAPDEK was designed in a way, so that it covers the essential steps of cognitive tasks. If they do not apply, certain steps of the MAPDEK loop (such as planning) can be excluded from loops. However, it is typically only a matter of task refinement to describe the missing steps inside the loop. They were probably left out for reasons of presumed non-relevance, or they may have been forgotten. This should be investigated on a case-by-case basis.
- *It can occur, that the order of the MAPDEK steps differs from the MAPDEK scheme.* Given the background of MAPDEK loops, a different order of the steps is possible. It should be decided on a case-by-base basis, whether the original order or the MAPDEK order seems beneficial.
- *There can occur left-over tasks that do not really fit into a MAPDEK loop.* As stated above, there is nothing like a "complete" loop scheme. So extra tasks will occur. Their existence however raises questions of necessity and consistency, they are yet another case for an individual treatment.
- *Redundancy can be seen in the MAPDEK loops.* Redundancy is known to have good effects (such as spare resources to increase availability, or double checking certain matters) and bad effects (waste of resources, over-regulation). Therefore, we cannot judge about redundancy in general, but the tendency in automation projects shows that it is common to reduce redundancy to the minimum necessary and acceptable level in the course of automation, as this reduction is a main contributor to more efficiency.

4.6.3 Regarding R2.3 Machine-interpretable loop modeling and linking back to the tasks in task analysis

Put loop models in data files!

Engineering of all types of systems relies on a) requirements data and b) construction data.

Regarding a) requirements: Ideally, the requirements are specified at the task level (see previous chapter). Then, to refine task requirements to loop requirements is a matter of requirements decomposition, which needs to be done in parallel to task decomposition into loops. This decomposition needs to be problem-related. The management of requirements itself however, can be regarded a typical task in requirements engineering. We can be confident, that standard tools for this purpose do exist, e.g. Telelogic DOORS, which aids requirements engineers and systems engineers in this decomposition task or at least in its documentation and consistent linking.

As systems get more complex, b) construction data was moved to computers. In comparison to paper-based engineering, computers were especially used for organisation of information, consistence, easier changes, undo's, and copy and paste, and easier access to and distribution of the data. Then, as models were made machine-interpretable, and computers got more and more powerful, computer-aided engineering was introduced to replace simulations performed on a computer occasionally before with more and more simulations, forecasts, visualization integrated into the user interface of the engineering systems.

This development is the main reason, that EIMA must be based on machine-interpretable data. Here as well, computer support will first be used for better organization of large models and consistent linking between the piece parts. A use case of EIMA will be shown in Chapter 8, potential advanced uses of this data are presented in the Outlook section of this thesis (see Chapter 8).

Use a proper modeling application!

The same argumentation applies here as in Chapter 3. Modeling MAPDEK loops is a challenging task as system analysts/designer deal with large task hierarchies. During task to loop refinement, designers/analysts will especially profit from views, filters, a template mechanism (for MAPDEK), versioning, graphical editing, online consistency checks, statistics and simple simulation facilities.

In addition to the task decomposition tree which now gets associated with loops, a view of a likely prototype of the managed resource pool would improve completeness and consistency of the modeled loops.

In terms of applications, there is none (to the knowledge) of the author, that supports high-level loop constructs, while applications based on control loops are existent (such as Matlab/Simulink, ASCET, and others). General purpose systems modeling tools like Artisan Studio with data formats based on SysML once again (as with task analysis) look very promising to offer a majority of the needed functions.

Use a format suitable for data exchange!

As the evaluation has shown, a common information model or data model as a data exchange file format does not exist for the MAPDEK loops or related loops.

A typical approach for modeling would be to create a new information model (e.g. using entity-relationship diagrams), which then in terms of a data model can be translated to either to a certain structure of tables in a relational database or a new formal languages, typically in XML along a new schema. From the point of view of the author of this thesis, there's no real use in this approach, as it adds to the heterogeneity and lacks tools, both not contributing to standardization. However, an approach like this may be the fastest to implement, as standardization being based on agreements is a matter of long periods and , while individual solutions can be implemented quickly and made "compatible" via small conversion applications.

As loop steps are sub-tasks, also see the recommendations on task analysis in Chapter 3. The applications and approaches listed there to do task analysis are therefore also applicable to the MAPDEK loops. The most promising formats seem to be UML and SysML with their block definition diagrams.

In SysML, tasks can be identified by task IDs, so that loops have an ID they can refer to. The association of tasks with the MAPDEK categories can be performed via either stereotypes, from which task blocks can be derived, or via inheritance, via annotations, or a SysML allocation matrix. Especially regarding data exchange, each of these approaches will have its drawbacks and benefits.

SysML and UML have file format representations (MOF and XMI) that are suitable for data exchange. Data about loops that does not natively fit into the current versions of these languages can be added either via an extension to a language or by using comment fields. This approach is not optimal, but an extension approach will be much more actionable than a new information and data model.

4.6.4 Example

This section once again picks up the example on change control from the previous chapter. See the workflow in Fig. 3.2 which had been the input to task analysis, and the results of EIMA task analysis in section 3.6.3.

Use the MAPDEK scheme at all levels of abstraction: from process improvement to resources!

With the new MAPDEK structure we can easily check, how the actions fit this loop. The first observation that is clear from the role hierarchy, is that the requestor must reside in a superordinate loop, that deals with some kind of product improvement. We do not know his reasons for sending the change request, but can assume that the change request is part of an execution step and that the success of the change request will be monitored by the same external entity in a following cycle.

For the reason of a superordinate and a subordinate loop we have split the task decomposition tree into two loops. Basically, even more loops could have been used (see the structure of the role hierarchy to get an idea of the hidden complexity in this process), but we kept it as two loops here for reasons of brevity and so that the figures fit on a page. In systems modeling tools, we would not have such physical restrictions and could (because of tools for navigation views and zooming) model more subsequent loops in more detail.

As a result, we have two levels of abstraction: Loop L01 for Change Control and closer to the actual changed IT products, and a superordinate Loop L02 for Product Improvement, which is more remote from the managed IT product(s).

Align the sub-tasks to the MAPDEK scheme!

In the previous chapter, the tasks were already pre-aligned to the MAPDEK loop scheme. There, the association to the MAPDEK steps was performed via inheritance. For visualization, in the figure the individual steps were colored to hold the MAPDEK steps apart. The figure is reprinted here in Fig. 4.19 to make it easier for the reader:

The following list contains the explanations why the tasks have been assigned to MAPDEK steps in that manner:

- Monitor:
  - The workflow only specifies T01 Request Change. The following reception of the change request was not explicitly modeled in the original workflow, so we left it this way in task analysis.
- Analyze:
  - The initial evaluation (T02 Assign Evaluators and T03 Log Evaluation) is beyond simple monitoring and preprocessing, but they are sub-tasks of an analysis of the incoming request. Evaluations are a typical task for an analyzer, in fact "evaluate" and "analyze" have a similar semantics. Interestingly, the actual evaluation task was not modeled in the original workflow. We leave it for refinement later.
- Decide:
  - T04 Review Change Request is an interesting task. As can be seen in the original workflow, this is the step that actually decides on whether the change request is accepted or not. On urgent requests it decides without an in-depth evaluation and before planning. For these reasons, it does not belong to analysis, but it is a decision task that is located in an unusual place. In the original workflow, how can a change manager decide on urgent requests

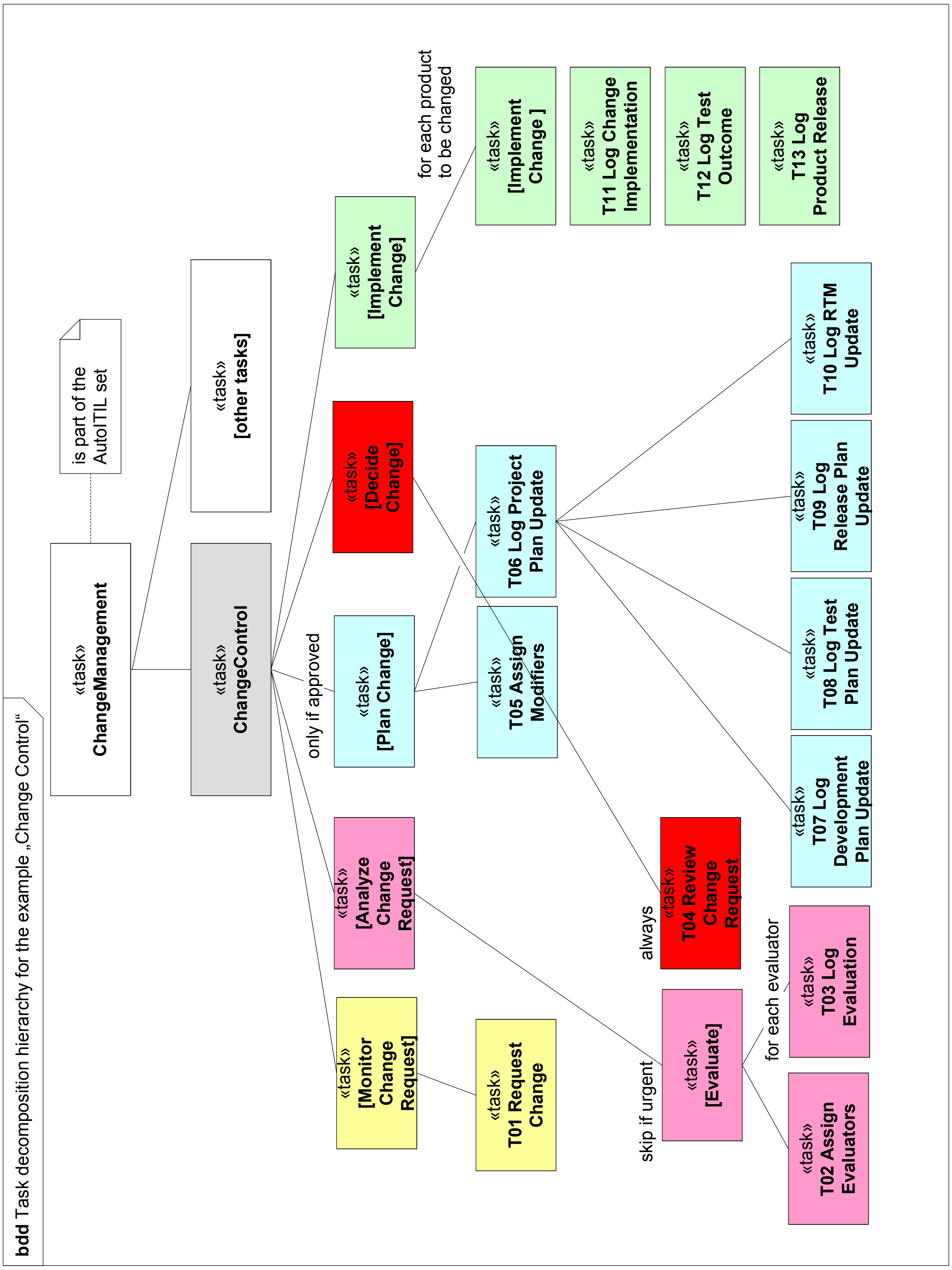

bdd Task decomposition hierarchy for the example „Change Control“
is part of the AutoITIL set
«task» ChangeManagement
«task» ChangeControl
«task» [other tasks]
«task» [Monitor Change Request]
«task» [Analyze Change Request]
only if approved
«task» [Plan Change]
«task» [Decide Change]
«task» [Implement Change]
for each product to be changed
«task» T01 Request Change
«task» T05 Assign Modifiers
«task» T06 Log Project Plan Update
«task» [Implement Change]
«task» T11 Log Change Implementation
«task» T12 Log Test Outcome
«task» T13 Log Product Release
skip if urgent
«task» [Evaluate]
always
«task» T04 Review Change Request
for each evaluator
«task» T02 Assign Evaluators
«task» T03 Log Evaluation
«task» T07 Log Development Plan Update
«task» T08 Log Test Plan Update
«task» T09 Log Release Plan Update
«task» T10 Log RTM Update

Figure 4.19: SysML block definition diagram to define the task decomposition hierarchy in the example. The tasks in brackets are new composite or leaf tasks, that did not appear in the workflow. They are added for clarity. (Figure reprinted here from page 85).

without having the alternatives about the following change plan? In Fig. 4.19 we have left T04 at its position by sequential order of tasks but already associated it with the decision loop step.

- Plan:
  - T06 to T09 are straightforward, because the word "plan" already appears in their names. T10 Log RTM Update (RTM = ready to market) is a prediction of the future release date, and so naturally also fits planning. T05 Assign Modifiers does assign the planning and executing bodies and is associated with (staff) planning, and therefore planning.
  - As in the evaluations, the original workflow misses to add the actual planning tasks. They are all folded into logging tasks. We leave this for refinement later.
- Execute:
  - The workflow does not even contain the actual change implementation so we added a task for this purpose first. Then T11 to T13 are easily assigned to the execution phase, even though they rather care about the bureaucratic housekeeping tasks than the actual change implementation.
- Knowledge:
  - The workflow does not contain any advanced knowledge tasks such as learning or inference, therefore there's no knowledge category in this loop. All this is implicitly assumed to happen in the human actors naturally. As the local knowledge base also contains a log of activities, we can associate all the logging activities (T03, T06-T13) with an interaction between the individual loop steps and local knowledge.

Check the loops for completeness and redundancies!

When associating the tasks with the two loop constructs, we could identify a whole number of issues and fixed them during the transformation from a task decomposition tree to a loop scheme, for the result see Fig. 4.20:

The actual list of changes makes them more explicit and reasons on their necessity:

- General changes:
  - Two loops were introduced as an ordering scheme. In the figure they are represented visually with strong boxes. In SysML, this can be modeled with a package diagram.
  - The MAPDEK steps are now arranged in a MAPDEK loop layout with the branches of the tree facing upwards for easy visual reference. This is similar to a different view style in a modeling application.
  - All logging tasks were renamed. In EIMA we assume an implicit logging of all intermediate results and actions and can therefore give the tasks more applicable names.
- In Monitoring:
  - We now split T01 Request Change into a sending and a receiving task, as they belong to different loops. The receiving task can later be refined with checks, statistics, time stamping and related monitoring tasks.
- In Analysis:
  - Renamed T03 to express the actual evaluation. The logging of the result is a sub-task of this activity.
- In Planning:

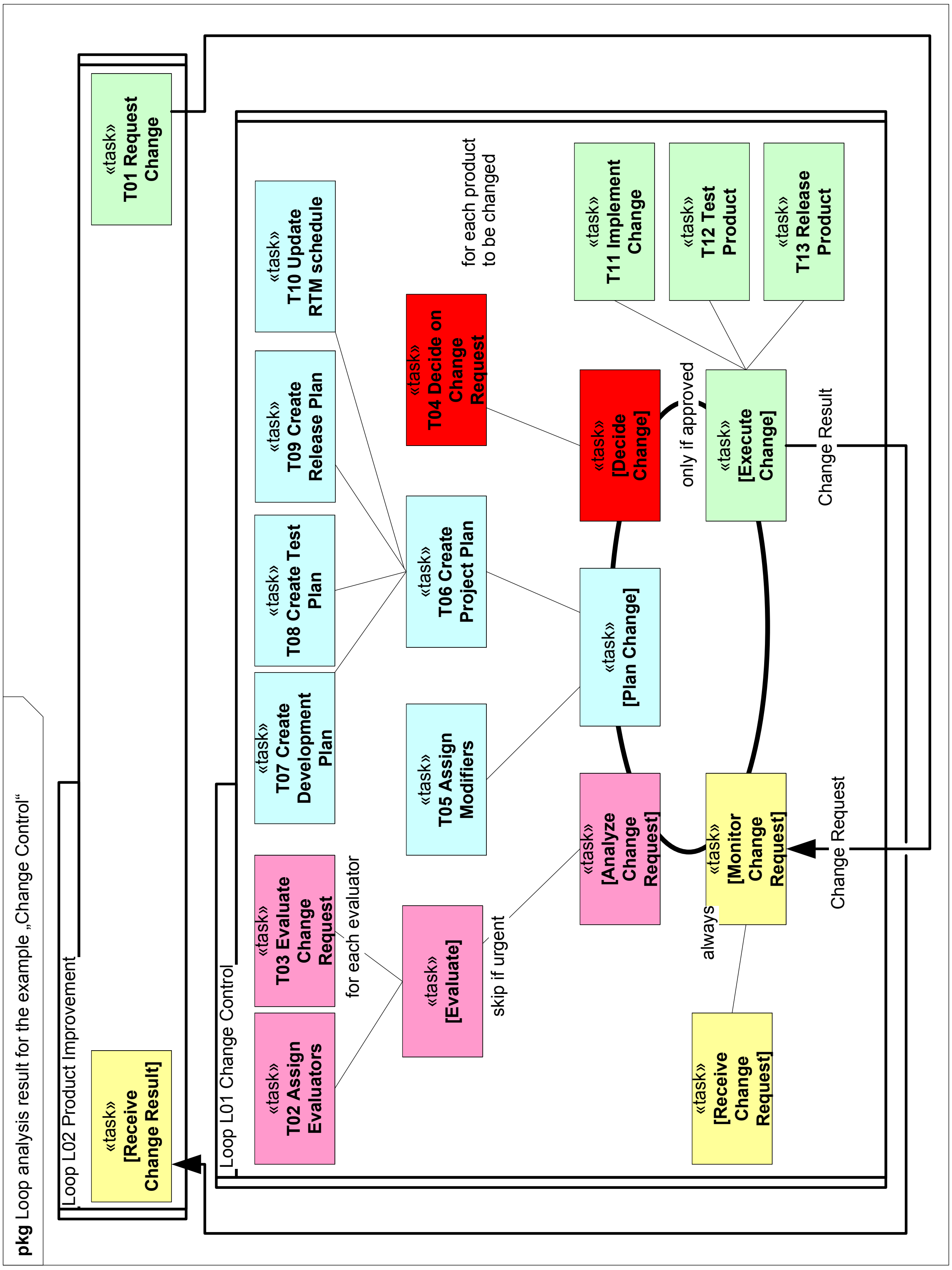

pkg Loop analysis result for the example „Change Control“
Loop L02 Product Improvement
«task» T01 Request Change
«task» [Receive Change Result]
Loop L01 Change Control
«task» T02 Assign Evaluators
«task» T03 Evaluate Change Request
«task» T07 Create Development Plan
«task» T08 Create Test Plan
«task» T09 Create Release Plan
«task» T10 Update RTM schedule
for each evaluator
«task» [Evaluate]
«task» T05 Assign Modifiers
«task» T06 Create Project Plan
«task» T04 Decide on Change Request
for each product to be changed
skip if urgent
«task» [Analyze Change Request]
«task» [Plan Change]
«task» [Decide Change]
only if approved
«task» [Execute Change]
«task» [Monitor Change Request]
always
«task» [Receive Change Request]
«task» T11 Implement Change
«task» T12 Test Product
«task» T13 Release Product
Change Result
Change Request

Figure 4.20: The result of loop analysis for the Change control example shows two MAPDEK loops.

  - Theoretically, a project manager overlooks the whole change planning and cares about consistency, when an overall goal (the new version) is planned via multiple plans (T06 to T10). However, this necessary coordination is not visible in the original workflow, there the planning looks independent, which cannot be the case. We can only see it from the role dependency diagram. We have put individual plans below the overall project plan.
  - We renamed all tasks to reflect the actual actions instead of the logging sub-tasks.

- In Decision:
  - In the original workflow, T04 "Review change request" acts as the central decision about the request, based on the result of an evaluation. However, the change itself is planned after this decision. What jumps at us is, that the evaluators and change planners will likely do similar tasks. We have now moved T04 to the decision phase after evaluations (for non-urgent requests) and planning (for all requests).
    The rationale behind this change is: When automated, then monitoring will reject non-compliant requests, analysis will tag the incoming request in case there are serious issues. These tags will cause planning to review it and quickly guide it to decision. This way, non-successful requests are quickly rejected in any case as before. For the normal request however, now the decider can base its decision on the results of change planning! It can even choose among plans. This is far better than the original workflow design.
- In Execution:
  - T11 to T13 have been renamed to their actual actions.
  - A change result notification the upper loop has been added.
- In Knowledge:
  - NA.

Already at this early phase, we can see a number of potential improvements, which was the main reason for using EIMA on this problem:

- In Monitoring:
  - There is no instance, that checks the misuse (frequency or necessity) of URGENT marks. Input must therefore come from a trustworthy requestor. We can add steps to monitoring to work around this issue. Monitoring is a step that usually is very well applicable to automation.
  - Eventually we can drop the "urgent" tag in incoming requests completely once evaluations and change planning have been automated and are performed much more quickly.
- In Analysis:
  - Evaluators must base their evaluations on certain criteria. If the evaluations are performed on the basis of objective criteria and comparisons, than this action should be formalized to support evaluations with automation.
  - Likewise, the selection of the evaluators (presumably humans in the original workflow) must be based on rules. If we could formalize these rules, we could apply automation here or drop this task completely, as the automatic evaluator is quick and much better in terms of objectivism than a human evaluator.
- In Planning:
  - Many plans need to be created here, that all must be consistent. This step is very likely to profit from automated planning support.

- In Decision:
  - For a decision, many objective criteria (development time, necessary budget, invoked resources) can be summarized and presented to the decider before his decision. Such a decision support system will much more base decisions on objective criteria. Still, as decisions have also a strategic element that is hard to quantify, a human decider will remain, but it will be much easier for him to make a decision on aggregated facts.
- In Execution:
  - The actual change in the product is unlikely to be automated by a great extent. Even though we can imagine things like a system automatically identifying the affected products and potentially modules to be changed, the change itself (apart from changing parameters or configuration files) requires too much creativity. However, then testing and release are likely to be supported by automation. The original workflow had defined that the external requestor would do the test, here we can make improvements if we would require that the requestor sends his test routines and expected results together with his request.

4.7 Conclusion

The general aim of loops in EIMA is to find an appropriate model that breaks down tasks, sorts the sub-tasks into common categories and thus eases both task allocation to either humans or machines and quickly finding related machine methods to implement tasks allocated to machines.

As a result of task analysis, we know that the tasks themselves are of a cognitive nature and repetitive thus forming cognitive loops. The previous sections listed a couple of these loops, but for EIMA we need loops at a level of abstraction that is suitable for communication, coordination and trade-offs between systems designers and later customers and their system administrators.

However, we have multiple perspectives on a management automation system: on the user side: the one of the system administrators and the one of the later business management. During system design of a management automation system, these two perspectives must be taken care of.

In many places (such as control loops in engineering and MAPEK loops in the ACI), it can be seen that both world essentially overlap. We take over the model-based design approach from engineering, as there quite complex systems have been tackled and equipped with mature management automation routines.

The MAPDEK loops in EIMA have been carefully chosen to combine features of both: technical management and business management. They enhance the current MAPEK model used in the ACI, but add the “decision” step as a focal point to later adjust the level of autonomy to the potentially dynamic needs of the users.

To form loops from task decomposition trees (the result of the previous chapter) to loops, the task needs to be associated with the corresponding task categories and then associated with loops. Especially, modeling analysis, planning and knowledge is the key entry point to later attach sophisticated methods and to truly reduce the mental workload of operators.

MAPDEK loops should be modeled in standard modeling software. Given the limited success of academic efforts to establish languages for the artifacts in the ACI, we believe that the use of a well-accepted tool to model systems in SysML is appropriate for the EIMA purpose. The details then depend on factors such as the preferences of the involved people or compatibility of tools.

The next chapter will deal with allocating the loop steps (or functions as they are called by Sheridan) to either humans or machines.

Outlook: Loop Coordination As IT infrastructures grow, the hope is that we can simply keep such a loop structure and raise the cycle count. This trivial parallelization would keep IT management automation scalable, but we're well aware of the fact, that we will need to add coordination and integration to the loops.

A question yet unsolved is the one on composing the loops and predicting their cooperative behavior based on mutual influence. It remains unclear how to compose different control systems to control one entity, and how to deal with openness and unexpected events. It remains difficult to compose or combine autonomic systems (Alva Couch, Tufts University, Hot Autonomics Workshop 2008) and to deal with unpredictable behavior.

This thesis will not solve this issue on its own, but we add the known requirement, that models for the loops should allow reasoning about the overall behavior of a system that is composed of loops. What helps in our case is the proposed hierarchical task organization presented in the previous chapter, while other researchers see a more promising approach in non-hierarchical structures such as peer-to-peer management automation systems. But keeping this requirement on the list will remember us to add work on loop composition and global loop coordination in EIMA to the list of potential follow-up work.

5 Step 3: Function allocation to humans/machines

Contents

5.1 Motivation and current examples

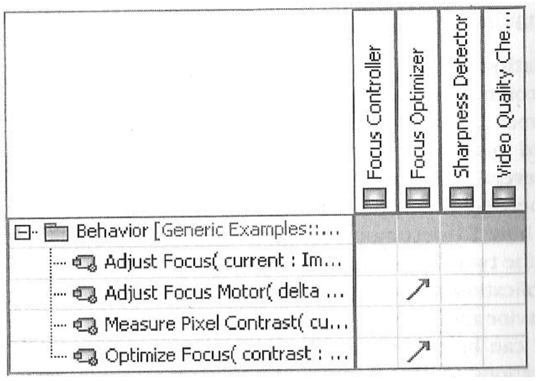
The previous steps in EIMA did 1) task analysis (resulting in: task decomposition tree, role hierarchy, task automation table) and 2) loop composition (resulting in a loop diagram). Both of these were designed to be independent of the assignment of tasks to humans or machines. The leafs of the task decomposition tree (and therefore leafs in the loop diagrams) are the base tasks that really need to be done. These base tasks are called “functions” in systems engineering (see [She02, FMS09]). Only these functions are of interest in function allocation. If the tasks in a workflow are fully expanded, then only functions will show up, so they are also easy to identify there. In the task decomposition tree, all other superordinate tasks (intermediate nodes above the leafs) are composites, and their allocation will result from the allocation of its sub-items.

5.1.1 Allocating functions to humans or machines

In systems engineering, as a process, function allocation refers to the sequence of steps involved in establishing the alternate roles, responsibilities, and requirements for humans and machines in a complex human-machine system.

In contrast, current IT management automation in terms of definition and documentation of IT service management processes (e.g. see CoBit) works with staffing matrices or responsibility assignment matrices, also called RACI charts. Here, the result is, which *human* role is responsible, accountable, consulted, or informed on what tasks.

Fig. 5.1 shows two example allocation matrices.



	Focus Controller	Focus Optimizer	Sharpness Detector	Video Quality Che...
Behavior [Generic Examples::...				
Adjust Focus(current : Im...				
Adjust Focus Motor(delta ...		↗		
Measure Pixel Contrast(cu...				
Optimize Focus(contrast : ...		↗		

	R01 Requestor	R02 Change Manager	R03 Evaluator	R04 Project Manager	R05 Development Manager	R06 Test Manager	R07 Release Manager	R08 Requirements Manager	R09 Modifier
T01 Request Change	X								
T02 Assign Evaluators		X							
T03 Log Evaluation			X						
T04 Review Change Request		X							
T05 Assign Modifiers		X							
T06 Log Project Plan Update				X					
T07 Log Developm. Plan Upd.					X				
T08 Log Test Plan Update						X			
T09 Log Release Plan Update							X		
T10 Log RTM Update								X	
T11 Log Change Implementation									X
T12 Log Test Outcome	X								
T13 Log Product Release									X

Figure 5.1: Example function allocation matrices. The left one shows an example from a video system, where functions are allocated to components of this *system*, and therefore machine roles. The example was taken from [FMS09].
The right one was directly derived from the swimlanes in the change control workflow example in Ch. 3. Function allocation is marked with “X” being a place holder, as the correct RACI category had not been encoded in the original workflow diagram. In this transcript of the original workflow, implicitly *human* roles are meant to allocate functions to.

For function allocation in EIMA, we have to combine these two views, as technical IT management is right at the boundary between the components of a management automation system and administrators' work. However, due to the management automation system only being planned in this phase, we cannot refer to individual components in the system. We also only have a rough idea at this point about the involved human roles. For the reason of this uncertainty, and to keep complexity low, EIMA simplifies functional allocation to the most simple and most fundamental automation decision for each function: Which of both sides deals with this task? The answer offers three alternatives: a) a machine ("fully automatic"? b) a machine and a human ("half-automatic")? c) a human ("manual")?

5.1.2 Automation and Self-Management

Function allocation is therefore the focal point of automation. In Ch. 3, we have defined the range of IT management tasks, that EIMA is intended to cover. They are essentially the AutoITIL tasks identified there.

According to the Webster dictionary, "automation" is "(1) the technique of making an apparatus, a process, or a system operate automatically or means (2) automatically controlled operation of an apparatus, process, or system by mechanical or electronic devices that take the place of human labor." Then, etymology tells us, that "automatic" is derived from Greek "automatos", which means "aut-" + "-matos": "self-acting". In related work in IT management, initiated by IBM's Autonomic Computing Initiative, there has been created a synonym called "IT self-management" or in short "self-management". But there are even occurrences before the ACI, as in [FHSL96].

Despite their similar semantics, automation and self-management have different connotations. "Automation" clearly refers to a technical context (self="in technical systems, here self = automated, happens in technology regardless of their location and with minimal human interaction, given a certain goal is known"), while self-management raises the question of discrimination of "self" and "non-self" (see [FPAC94, Ish04]). "Self" is therefore relative to a certain environment and a certain boundary.

It is unclear, whether "self" also refers to "local". We can make this more explicit along the following list of roles, that can occur around a locally concentrated IT resource pool :

	management automation	self-management
functions allocated to *local humans*	no	no
functions allocated to *remote humans*	no	no
functions allocated to *local machines*	yes	yes
functions allocated to *remote machines*	yes	unclear

For (IT) "self-management" it is unclear, whether remote machines are allowed to take over tasks. To avoid misunderstanding, we refrain from using the self-* terms in this thesis and instead use more technical "automation" terms.

5.1.3 Intended effects

The goal of EIMA in general can now be defined in terms of the entries of such a matrix: moving function allocation marks from humans to half-automatic, and from half-automatic to fully automated solutions, where accepted and feasible.

The overall intended effects are classic goals in automation: intended improvements in system functional performance (effectivity) and/or cutting management costs (labor) per amount of resources (efficiency). IT management automation therefore refers to:

- replacing human skills with capabilities built into a machine system to approach a human/manager specified abstract goal in consideration of context information and human/manager specified constraints and policies.
- reducing the frequency of necessary management actions
  - increase time between necessary management commands (part-time autonomy)
  - increasing the time between necessary management SET commands to be sent to the system
- raising the level of abstraction in management interactions
  - specify (fewer) goals at a higher level of abstraction than before
  - allowing more abstract management commands
- by means of internal knowledge and context information.

As we explicitly refrain from shifting tasks among humans in EIMA, we leave certain options out, which might achieve the cost effects of automation, without being automation: namely, shifting tasks to people, who can handle the same workload cheaper and or more quickly. In fact, out-sourcing, off-shoring, near-shoring, and remote maintenance/tele-operations are all interesting economic alternatives to automation, moving labor to countries with lower wages, or to specialists, but they are out of scope of this thesis, as they are not automation.

5.1.4 Kinds of management automation systems

It is important to note, that EIMA deals with management automation system in a sense of a gradual transition towards systems with more and more IT management automation features, step by step taking over certain MAPDEK steps. We do not, to state this explicitly, only deal with the goal of fully automated systems.

Instead in EIMA, certain tasks are moved from being performed manually in the operations phase, to being analyzed and designed at system design time for a shift to be performed by machines.

To achieve this, system development must predict future management operations, and automate all major potential management tasks, where complexity allows automation and frequency demands automation. It must also provide support/interfaces for non-automated mgmt tasks.

The following list of examples from systems engineering should give the reader a clue on the range of "management automation systems", that we can use for inspiration:

- driver assistance systems
  - fuel consumption monitor (a simple monitor system that displays a variable derived from sensor input)
  - navigation system (a recommendation system giving directions but not controlling anything)
  - park assist (a recommendation system with steering wheel control but manual override)
  - gear change recommendation (a recommendation system with no control)
  - night vision system (a support system to overcome human incapabilities, no control)
- control systems
  - flight control system (controls the flight actors in an airplane based on pilot input)
  - air-traffic control system (guides arrivals and departures of flights near an airport, and flights en-route remotely by communication)

  - engine control system (controls air, fuel, ignition for an engine based on driver goals)
  - transmission control system (controls valves, clutches)

As EIMA is about management automation in *information technology*, the interesting systems here would be assistant or management systems for AutoITIL tasks, such as:

- Auto Incident Management
  - incident classification system (for a given incident, determine related ones)
  - known error database analysis system (from known errors, achieve statistical results)
- Auto Configuration Management
  - negotiation protocols between components
  - asset management systems (search for resources and identify)
  - automated updates
- Auto Availability Management
  - high-availability systems
  - fault-tolerant systems, fail-over mechanisms
- Auto Capacity Management
  - capacity and demand trend analysis systems
  - backup and archival planning and automation

From these lists, the reader shall see the similarities in both of these domains. The basic overall structure of such systems had already been shown in Ch. 1 in Fig. 1.3 on page 3.

In the last years, more and more management automation systems have evolved, creating a stack/layered architecture of management systems, so that there are many management systems and managers, see Fig. 5.2. As can be seen, function allocation in such systems will get more complex than the simple EIMA approach, whether to automate or not. It shows, that for both sides (human and machines) there are lots of possibilities how to allocate the functions within the set of machines, and within the set of humans.

5.1.5 Function allocation in the examples

Following the EIMA approach, tasks, loops, and steps have been defined on the vendor side. During the EIMA process, all humans/machines will be abstracted by roles, which is a common approach to generalize the function allocation decision. On the operator side, roles must later be decided to be done by which humans and machines, as roles need to be mapped to individual persons/management systems.

Looking back at the examples at the beginning of Ch. 3, example 1 (see Fig. 3.1 on page 47) does not work with roles, but with instances of human workers. Altogether, five persons are shown by photo attached to a task. These are tasks assigned to these humans. Then, there are four tasks with a gear symbol, which are obviously performed by a machine, e.g. a script. However, the machine identification is not shown. Responsibility in the enterprise is implicitly shown by the swim lanes naming departments in an enterprise. This is a hint on who is responsible for the machine-performed tasks, but not a good allocation.

Example 2 from Ch. 3 (see Fig. 3.2 on page 48) does use roles, but we can assume, that all of them are implicitly allocated to humans. There is no notion on responsibilities.

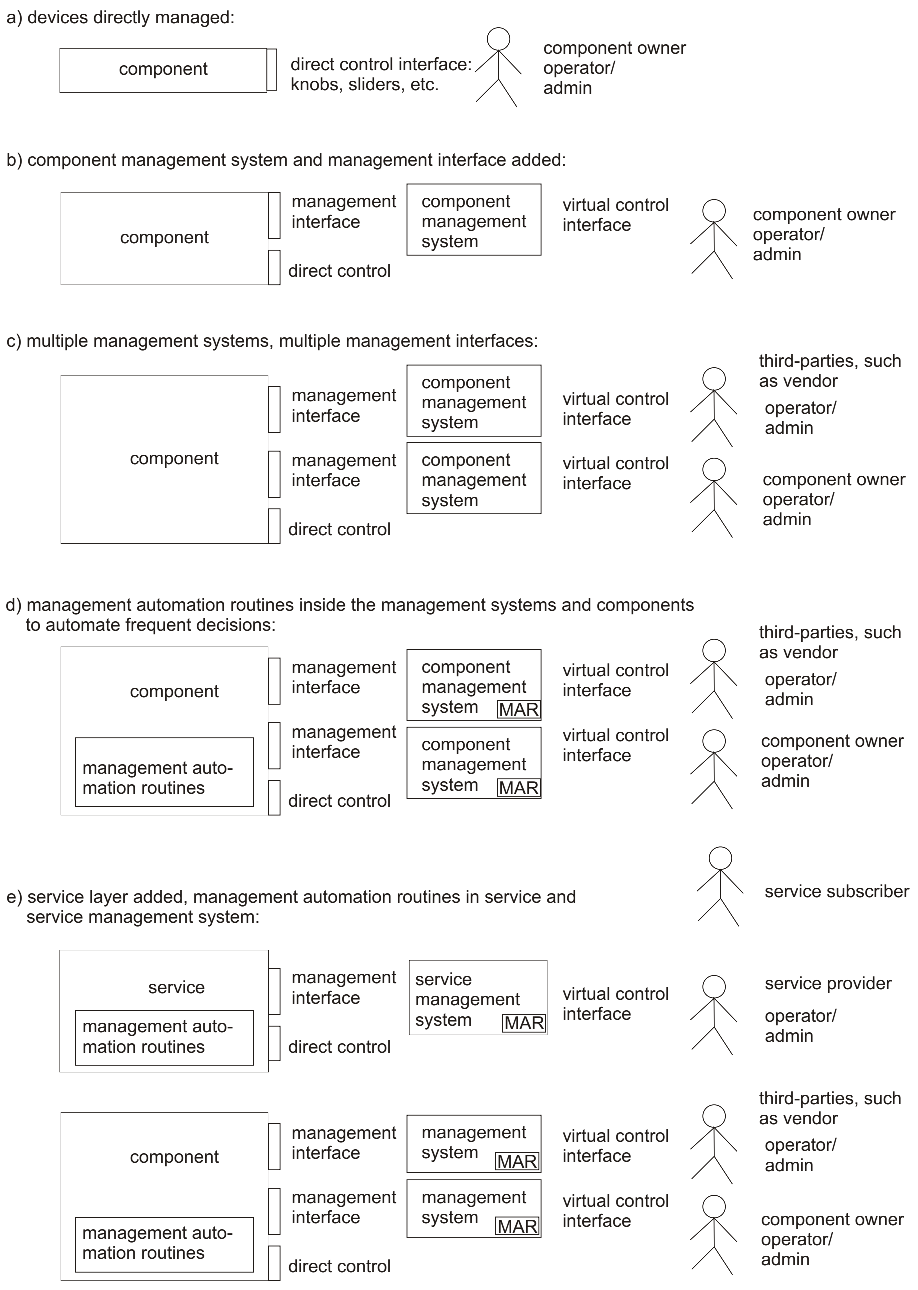

a) devices directly managed:
component
direct control interface: knobs, sliders, etc.
component owner operator/ admin
b) component management system and management interface added:
component
management interface
direct control
component management system
virtual control interface
component owner operator/ admin
c) multiple management systems, multiple management interfaces:
component
management interface
management interface
direct control
component management system
component management system
virtual control interface
virtual control interface
third-parties, such as vendor
operator/ admin
component owner operator/ admin
d) management automation routines inside the management systems and components to automate frequent decisions:
component
management auto-mation routines
management interface
management interface
direct control
component management system MAR
component management system MAR
virtual control interface
virtual control interface
third-parties, such as vendor
operator/ admin
component owner operator/ admin
e) service layer added, management automation routines in service and service management system:
service subscriber
service
management auto-mation routines
management interface
direct control
service management system MAR
virtual control interface
service provider
operator/ admin
component
management auto-mation routines
management interface
management interface
direct control
management system MAR
management system MAR
virtual control interface
virtual control interface
third-parties, such as vendor
operator/ admin
component owner operator/ admin

Figure 5.2: Over time hierarchies of management hierarchies have been established. From external device management by administrators or operators with physical user interfaces, to administrators using management systems with virtual user interfaces, to systems with remote-management capabilities or management automation routines. In such structures, function allocation will get more and more complex due to the multitude of available options, where to place management automation routines.

Both examples leave a lot to be desired in terms of explicitness, genericness, dynamics of function allocation. The requirements on function allocation that have been derived from these and other examples will be listed in the next section.

5.2 Requirements on function allocation in EIMA

The following list of requirements proceeds the motivation and aims at better function allocation in EIMA.

5.2.1 R3.1 Static binary function allocation concepts

EIMA should define a scheme for human/machine function allocation and accountability for functions explicitly!

At first, EIMA needs a concept for static function allocation. Static function allocation (at design time) is the common way today. It results in tasks either being automated or not. In static function allocation, EIMA should support two important kinds of information:

- Does a human or a machine perform this task? Tasks that have been split to a manageable size and complexity and assigned to loops (functions) can be allocated to roles of humans or machines.
- Who is accountable that it is done correctly?
  - Automation can be powerful, but also risky. In addition to the role performing the task, we should have the notion of an accountable human person, as machines cannot take accountability. For human tasks, the person doing the task (responsible for the work) is also accountable, unless specified otherwise. For machine tasks however, we should name this accountable human role/person explicitly to avoid unknown accountability.

EIMA should allow explicit tagging of roles, whether they are allocated to humans or machines!

Another deficit, that EIMA should overcome, could be seen in the examples, where usually, there is an implicit assumption on each of the roles, whether the role will later be impersonated by a human role or a machine role. This is sometimes expressed by names and human intuition (semantics of words), e.g. a “Focus Optimizer” being a machine component, but a “Project Manager” being a human. We can foresee, that with assistant systems, it gets more and more fuzzy, whether a role is indirectly allocated to the machine or human side. Human interpretation however is a common source of misunderstanding and should be avoided in EIMA where possible.

EIMA should allow function allocation at loop, loop step or function level!

Also as a part of static function allocation, we now with MAPDEK loops we have the chance to consider function allocation at three levels of function allocation, which is a chance for increased abstraction and coping with complex models:

- function level: This would be the traditional case, in quite great detail.

- loop step level: This is function allocation for a whole M, A, P, D, E, K step, which could span a number of functions.
- MAPDEK loop-level: This would mean to specify allocation for all steps in a whole loop.

5.2.2 R3.2 Automation levels and standards

The previous requirement contained the automation decision mainly in a black and white fashion in terms of the extremes of (full) automation and non-automation. However, we can also foresee, that, given the high level of abstraction of tasks and loops in EIMA, the extremes will be rare, and most of the loops and loop steps will be allocated to half-automation. Which in turn means, that for these composite tasks, sub-parts will be performed by a machine, and sub-parts by humans. in EIMA the level of functions is still relatively high-level because the model must be manageable due to constraints in time and certainty.

EIMA should specify levels of automation!

While in general, most researchers strive for solutions with the highest level of automation, from the point of view of the author of this thesis, the role of systems with partial automation (assistant systems) is still underrated. Truly autonomous systems would lack a management interface, they could only be influenced via environmental changes that are registered by the system via sensors at the functional interface.

Half-automation is therefore an enormous white area in IT management automation. It is interesting for the reasons of lower effort for implementation, and administrators staying in the loop and in control, which can improve general acceptance and accelerate the introduction of such systems. The problem with half-automation however, is that “half” can vary widely, which opens room for new heterogeneity. EIMA therefore needs to structure the space of half-automation in a better way.

Fine-grained function allocation would result in zillions of possible combinations for the interaction of humans and computers. We believe, that standardized “levels of automation”, collected from common examples, can give good guidance for creating automation levels between automation and non-automation. Examples that catch widely distributed combinations would definitely ease adoption.

5.2.3 R3.3 “Good/best” automation for a task

Automation is always about optimization in terms of some overall explicit or implicit goal, so there should be a notion about what is a good or “the best” automation level for a certain scenario.

EIMA should give advice on automation feasibility!

For the reasons stated before, EIMA concentrates on automation feasibility, this means to quickly assess automation potential. The main question “Would it work?” means therefore, whether there are available machine capabilities that could perform the task, once they are fed with an appropriate model of the problem.

Non-Requirements on EIMA

Due to its nature of a quick technical assessment, EIMA cannot offer answers to those factors that much depend on the situation input. EIMA should therefore not be judged to be a general guideline, which tasks to automate. Instead, it can be used to rephrase overall workflow input to a more cognitive structure and to refine or abstract tasks. it should then be checked depending on the particular situation, whether for the tasks, where automation is indeed desired, machine capabilities can be identified by using EIMA.

Automation need EIMA can give hints on automation feasibility, but the *need* or automation is an external factor, that cannot easily be calculated. There are mainly technical influence factors, that result in a certain automation pressure, such as:

- high frequency of the function (cycles per time) (higher would mean, that automation is needed more)
- necessary short reaction time (shorter would mean, that automation is needed more)
- low complexity in terms of input and output parameters, as well as necessary processing (lower would mean, more applicable to automation).

but also non-technical, emotional or legal influences:

- Acceptance by customers/users/administrators. Administrators cannot completely be replaced, but rather be supported in their most troublesome tasks. Just like airplane pilots, that do not engage auto-pilot all the time, but who use instrument landing system (ILS) for landings.
- The perception of complexity differs with people, as people differ. The exactly same task can ask too much of one administrator, while another one loves this job (personal preference).
- Automation can also be a matter of policy. For example, self-firing, part-time-autonomous, non remote-controlled military objects can be disallowed in systems engineering. In a similar way, critical operations in IT management (such as content sniffing transmitted personal data) may be restricted by law or policies.
- Finally, effects caused by automation on other interfacing processes may make automation less or more applicable to automation. This is especially true for maintenance. If a management automation routine complicates maintenance, diagnostics, troubleshooting to an undesired level, then administrators and their management may restrain from automation capabilities.

Automation effects metrics. The common metrics applied to automation so far have an economic background: effectivity and efficiency.

Traditionally, IT management systems with automation have so long mainly been compared regarding their application-specific performance of the technical system alone. In this regard, management automation capabilities need to have a positive effect on performance or availability, without at the same time causing too much overhead. This would mean that in terms of effectivity, IT management automation increases the performance of the main system function. As we can replace "service" for the system's function technically speaking, this is also equivalent to service level improvements.

However, in such cases, the service level of the resulting systems alone is not the most interesting point, it's rather how much the system's service level depends on external management instructions. Obviously, the more an external entity has to manage the resource pool, that provides the service, to keep the same service level, the lower the level of automation. The ideal case would be top performance of the system without necessary external management. So to assess the automation effect, we would have to measure, and compare the impact: a) with external management instructions and b) without external management instructions on system functional performance. Clearly, in EIMA we cannot do this comparison, as it would require much deeper modeling than EIMA was designed for.

With respect to efficiency, we would need cost-benefit analyses, with costs including the necessary additional resources for management automation routines, but also indirect costs associated with risks of automation failures. Benefits would include saved expenses for less human work and eventually higher productivity.

Other economic metrics would be absolute profit or return on investment. As stated in Ch. 2, the aforementioned economic measures are not investigated in EIMA, given the enormous complexity and dependence on external economic parameters.

Approach for automation. A typical question arising in automation projects with constraints in time and budget is: Where to start applying/incorporating IT management automation systems? Essentially there are three classes of problems:

- Start with the complex, demanding tasks. Here, the goal of an assistant system would be to make the complex task look less complex to the administrator e.g. by taking into account constraints and only offering consistent selections, and therefore off-loading the human. The reasoning is, that complex stuff cannot be understood by humans, and will be a constant source of errors. If made easier, than we would work around these errors, which is a clear automation benefit. The risk of applying automation here is, that the task itself is not understand well enough to actually model and code it. In fact, based on imperfect input, the automation could make the same wrong decisions, that humans made.
- Start with the simple tasks. Here, we would use an automation system to keep up with the very large number of very simple, repetitive manual tasks. Because they are simple and well understood, automation should be simple too. The risks of this kind of automation are that due to their simplicity, automation is applied in a patchwork fashion, automating bits and pieces (including past redundancy) but overlooking the enormous benefits of an automation redesign. A human factor is, that humans lacking basic, simple tasks also lack relief from the stress of the more demanding tasks, that are left to them. Over time, they may even forget about basic procedures and basic knowledge with negative effects on their more complex jobs.
- Start with the medium stuff. If there are two extremes, then obviously this is the average choice, which combines both positive and negative features of the extremes.

We will use the cross-domain view to search, how in other domains recommendations on "appropriate" automation have been made.

5.2.4 R3.4 Dynamic function allocation at operation phase

We had seen in the examples, that today, function allocation is mostly static. For example, in Example 2 (see Fig. 3.2 on page 48) when an incoming request is marked as being "Urgent", then evaluation is skipped, and directly forwarded to the change manager. Who can then check for the necessity of the "Urgent" label and give it to the evaluators anyway. It can be seen that this matter could be changed if there was an additional automation level, where an mEvaluator machine role would automatically perform automated evaluation and an mPrioritizer would sort incoming requests according to a defined order of precedence.

EIMA should allow dynamic function allocation!

In essence, in this situation, a *dynamic* function allocation would be used: Those requests that can afford the time delay of human evaluators, will receive this "in-depth treatment". But those that cannot, will be glanced over by a machine in a few seconds and at least produce some evaluation input to the change manager, or give enough advice to question an urgent mark.

We had seen in the previous section, that the "best" automation level for a certain job can much depend over time on the situation, on utility functions, parameters and policies.

Resource consumption can be a reason for non-automation. On the one hand, Moore's Law in CPU power as well as miniaturization, falling prices for memory and disk space increase machine capabilities quickly, if software development holds pace with hardware improvements by using algorithms and details in models that become feasible and have often been described by mathematicians in AI long before machines were that capable.

Skepticism by administrators about the machine capabilities can be another reason for non-automation. On the other hand, we don't have a good understanding of the abilities of humans either. Human strengths have been analyzed for some professions where humans are critical and in situations of stress, e.g. astronauts, pilots, air traffic controllers, soldiers, power plant operators, and doctors to name a few. In comparison, not much work has been carried out specific to administrators in IT management, exceptions include Barrett et al (Usable autonomic computing systems: the administrator's perspective 2004), (Field studies of computer system administrators: analysis of system management tools and practices, 2004) and work published at conferences on topics like computer human systems and interaction.

In addition, with IT administrators' only marginally standardized education and necessary practical training (compare to air traffic controllers or pilots, for example!) it can be assumed, that there is a wide range in the abilities of IT administrators as well, working in different settings from SME to large companies or service providers.

Overall, limited confidence and trust in IT management automation routines may arise, when the level of automation was fixed by the systems designer at a level that is regarded as too high by administrators. With the level set too low, chances of automation are missed.

In data centers, typically night shifts and day shifts differ in the number of employees available for work due to humans' biological habit to sleep at night and to work at day-time. Given the varying number of staff, automation can seem more promising in some situations, but less in others. The same is true for troubleshooting work, or maintenance windows. So, there will occur situations, when the desired level of automation will vary over the course of a day or week. With a controllable, dynamic level of automation, we could better react to these variations.

Concluding from these considerations, we should aim for a manageable level of automation for the whole system, each of the loops, eventually each loop step or task, instead of a fix level hard-wired and hard-coded into the management automation system by the systems designer along the lines of a small sample of hardware capabilities and a few administrators who contributed with their personal opinions. This way, function allocation would become dynamically managed object (a configuration parameter) on the aforementioned entities.

Certainly, this would not have the same automation power as well-modeled and well-balanced utility functions, but it would be more versatile and flexible, and less hard to implement. And with the administrators staying in the loop and in control, it might have a better chance for being accepted.

5.2.5 R3.5 Computer-aided modeling of function allocation

The EIMA way of function allocation with its new requirements, and new static and dynamic complexity will necessitate, that these properties are also modeled at design time. Ideally, these models would have value apart from documentation, such as visualization, checks, or simple visual simulation facilities.

EIMA needs an application to model function allocation and a format for data exchange!

So we need to make the previous ideas interpretable and actionable for systems modeling of IT management automation systems. We need modeling constructions for:

- static function allocation including human/machine allocation and responsibility of humans for automated tasks
- explicitly tagged human/machine roles
- function allocation at loop, loop step or function level
- standard levels of automation
- advice on automation feasibility
- dynamic function allocation

5.3 Existing knowledge and concepts from IT management automation

This section now investigates selected related work, that the author has identified to match in some way the requirements on EIMA. It should give an insight into function allocation, as being done today in the domain of IT management.

Two major kinds if artifacts can be considered, when dealing with function allocation in current IT management automation: 1) standards in IT management, and 2) management tools in IT.

5.3.1 R3.1 Static function allocation concepts

Scheme for human/machine function allocation and accountability for functions

As stated before, function allocation deals with allocating functions (base tasks) to roles.

Roles at operational phase IT management happens at the operational phase of an IT resource pool, therefore function allocation in IT management automation needs to take into account the human roles in this phase. As stated in Ch. 1, they include resource-related administrators, IT service-related staff, up to business management. In remote management scenarios, they would also include remote personnel. For IT solutions (resource pool with management automation routines) they also include the vendor supporting his product.

The relevant coverage of tasks in several management frameworks was already described in Ch. 1, so we concentrate on roles and responsibilities here.

ITSM frameworks Looking at the ITSM frameworks ITIL V3 [Ogc07], CoBit [Ins07] and Microsoft Operations Framework (MOF) [Pub08], we can see, that all three of them do not take into account the resource-related IT management staff, instead ITIL and MOF cover process-level management personnel (mainly process managers, and service desk), CoBit also includes upper business management (CEO, CFO, CIO, Chief Architect, head development, head IT, project management, compliance, audit, risk and security groups). So, in relation to technical IT management, they are relatively high-level. None of them includes machine roles.

In terms of responsibility, CoBit [Ins07] uses RACI charts for each of its processes, see Fig. 5.3 for an example. Accountable means, that this is the person who provides direction and authorises an activity. Responsibility is attributed to the person who gets the task done. The other two roles (consulted and informed) ensure that everyone who needs to be is involved and supports the process.

RACI Chart

Activities \ Functions	CEO	CFO	Business Executive	CIO	Business Process Owner	Head Operations	Chief Architect	Head Development	Head IT Administration	PMO	Compliance, Audit, Risk and Security
Develop and implement a process to consistently record, assess and prioritise change requests.				A	I	R	C	R	C	C	C
Assess impact and prioritise changes based on business needs.				I	R	A/R	C	R	C	R	C
Assure that any emergency and critical change follows the approved process.				I	I	A/R	I	R			C
Authorise changes.				I	C	A/R		R			
Manage and disseminate relevant information regarding changes.				A	I	R	C	R	I	R	C

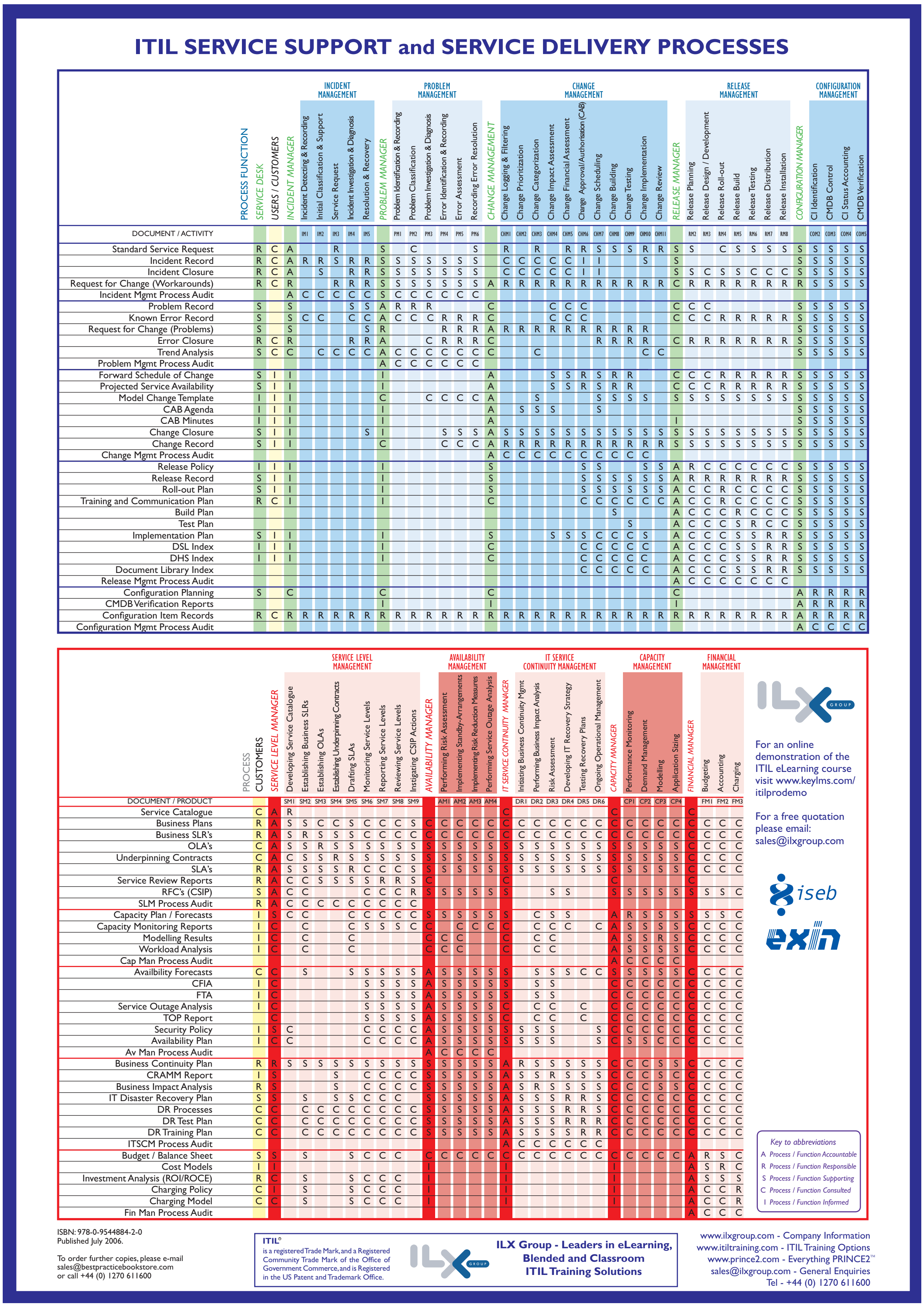
A **RACI** chart identifies who is **R**esponsible, **A**ccountable, **C**onsulted and/or **I**nformed.

Figure 5.3: RACI chart for Process AI6 “Manage Changes” in CoBit 4.1, see [Ins07]

ITIL V3 [Ogc07] itself (with one exception, see page 139 in the volume Continual Service Improvement for a RACI matrix on Change Management) and MOF do not include RACI charts, instead it uses verbal descriptions to state the roles’ responsibilities. RACI charts, however, have been created by an ITIL Training Company for ITIL V2 and V3. See Fig. 5.4 and 5.5 for these RACI matrices. They only cover the first refinement level of the processes and are aligned to documents and activities in ITIL.

IT management tools In current tools, function allocation is only roughly planned by the developer of the tool. Detailed RACI matrices are not part of them, instead most management tools do not even differentiate between the administrators that use them, they may not even have a notion of its users and their responsibilities. Many tools around IT management processes are rather targeted at process documentation instead of process automation. In fact, overall standardization of function allocation has not even started.

Rare exceptions include monitoring tools and watchdogs, which indeed take over the role of an “assistant”. In Siemens 7500 and 7700 systems with the BS2000 operating system, there was available an



ITIL SERVICE SUPPORT and SERVICE DELIVERY PROCESSES

				INCIDENT MANAGEMENT						PROBLEM MANAGEMENT							CHANGE MANAGEMENT												RELEASE MANAGEMENT								CONFIGURATION MANAGEMENT			
PROCESS FUNCTION	SERVICE DESK	USERS / CUSTOMERS	INCIDENT MANAGER	Incident Detecting & Recording	Initial Classification & Support	Service Request	Incident Investigation & Diagnosis	Resolution & Recovery	PROBLEM MANAGER	Problem Identification & Recording	Problem Classification	Problem Investigation & Diagnosis	Error Identification & Recording	Error Assessment	Recording Error Resolution	CHANGE MANAGEMENT	Change Logging & Filtering	Change Prioritization	Change Categorization	Change Impact Assessment	Change Financial Assessment	Change Approval/Authorisation (CAB)	Change Scheduling	Change Building	Change Testing	Change Implementation	Change Review	RELEASE MANAGER	Release Planning	Release Design / Development	Release Roll-out	Release Build	Release Testing	Release Distribution	Release Installation	CONFIGURATION MANAGER	CI Identification	CMDB Control	CI Status Accounting	CMDB Verification
DOCUMENT / ACTIVITY				IM1	IM2	IM3	IM4	IM5		PM1	PM2	PM3	PM4	PM5	PM6		CHM1	CHM2	CHM3	CHM4	CHM5	CHM6	CHM7	CHM8	CHM9	CHM10	CHM11		RM2	RM3	RM4	RM5	RM6	RM7	RM8		COM2	COM3	COM4	COM5
Standard Service Request	R	C	A			R			S		C				S		R		R		R	R	S	S	S	R	R	S	S		C	S	S	S	S	S	S	S	S	S
Incident Record	R	C	A	R	R	S	R	R	S	S	S	S	S	S	S		C	C	C	C	C	I	I			S		S								S	S	S	S	S
Incident Closure	R	C	A		S		R	R	S	S	S	S	S	S	S		C	C	C	C	C	I	I					S	S	C	S	S	C	C	C	S	S	S	S	S
Request for Change (Workarounds)	R	C	R			R	R	R	S	S	S	S	S	S	S	A	R	R	R	R	R	R	R	R	R	R	R	C	R	R	R	R	R	R	R	R	S	S	S	S
Incident Mgmt Process Audit			A	C	C	C	C	C	S	C	C	C	C	C	C																									
Problem Record	S		S				S	S	A	R	R	R				C				C	C	C						C	C	C						S	S	S	S	S
Known Error Record	S		S	C	C		C	C	A	C	C	C	R	R	R	C				C	C	C						C	C	C	R	R	R	R	R	S	S	S	S	S
Request for Change (Problems)	S		S					S	R				R	R	R	A	R	R	R	R	R	R	R	R	R	R										S	S	S	S	S
Error Closure	R	C	R				R	R	A			C	R	R	R	C							R	R	R	R		C	R	R	R	R	R	R	R	S	S	S	S	S
Trend Analysis	S	C	C		C	C	C	C	A	C	C	C	C	C	C	C			C							C	C									S	S	S	S	S
Problem Mgmt Process Audit									A	C	C	C	C	C	C																									
Forward Schedule of Change	S	I	I						I							A				S	S	R	S	R	R			C	C	C	R	R	R	R	R	S	S	S	S	S
Projected Service Availability	S	I	I						I							A				S	S	R	S	R	R			C	C	C	R	R	R	R	R	S	S	S	S	S
Model Change Template	I	I	I						C			C	C	C	C	A			S				S	S	S	S		S	S	S	S	S	S	S	S	S	S	S	S	S
CAB Agenda	I	I	I						I							A		S	S	S			S													S	S	S	S	S
CAB Minutes	I	I	I						I							A												I								S	S	S	S	S
Change Closure	S	I	I					S	I				S	S	S	A	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
Change Record	S	I	I						C				C	C	C	A	R	R	R	R	R	R	R	R	R	R	R	S	S	S	S	S	S	S	S	S	S	S	S	S
Change Mgmt Process Audit																A	C	C	C	C	C	C	C	C	C	C														
Release Policy	I	I	I						I							S						S	S			S	S	A	R	C	C	C	C	C	C	S	S	S	S	S
Release Record	S	I	I						I							S						S	S	S	S	S	S	A	R	R	R	R	R	R	R	S	S	S	S	S
Roll-out Plan	S	I	I						I							S						S	S	S	S	S	S	A	C	C	R	C	C	C	C	S	S	S	S	S
Training and Communication Plan	R	C	I						I							C						C	C	C	C	C	C	A	C	C	R	C	C	C	C	S	S	S	S	S
Build Plan																								S				A	C	C	C	R	C	C	C	S	S	S	S	S
Test Plan																									S			A	C	C	C	S	R	C	C	S	S	S	S	S
Implementation Plan	S	I	I						I							S				S	S	S	C	C	C	S		A	C	C	C	S	S	R	R	S	S	S	S	S
DSL Index	I	I	I						I							C						C	C	C	C	C		A	C	C	C	S	S	R	R	S	S	S	S	S
DHS Index	I	I	I						I							C						C	C	C	C	C		A	C	C	C	S	S	R	R	S	S	S	S	S
Document Library Index																						C	C	C	C	C		A	C	C	C	S	S	R	R	S	S	S	S	S
Release Mgmt Process Audit																												A	C	C	C	C	C	C	C					
Configuration Planning	S		C						C							C												C								A	R	R	R	R
CMDB Verification Reports									I							I												I								A	R	R	R	R
Configuration Item Records	R	C	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	A	R	R	R	R
Configuration Mgmt Process Audit																																				A	C	C	C	C

			SERVICE LEVEL MANAGEMENT										AVAILABILITY MANAGEMENT					IT SERVICE CONTINUITY MANAGEMENT							CAPACITY MANAGEMENT					FINANCIAL MANAGEMENT		
PROCESS	CUSTOMERS	SERVICE LEVEL MANAGER	Developing Service Catalogue	Establishing Business SLRs	Establishing OLAs	Establishing Underpinning Contracts	Drafting SLAs	Monitoring Service Levels	Reporting Service Levels	Reviewing Service Levels	Instigating CSIP Actions	AVAILABILITY MANAGER	Performing Risk Assessment	Implementing Standby-Arrangements	Implementing Risk Reduction Measures	Performing Service Outage Analysis	IT SERVICE CONTINUITY MANAGER	Initiating Business Continuity Mgmt	Performing Business Impact Analysis	Risk Assessment	Developing IT Recovery Strategy	Testing Recovery Plans	Ongoing Operational Management	CAPACITY MANAGER	Performance Monitoring	Demand Management	Modelling	Application Sizing	FINANCIAL MANAGER	Budgeting	Accounting	Charging
DOCUMENT / PRODUCT			SM1	SM2	SM3	SM4	SM5	SM6	SM7	SM8	SM9		AM1	AM2	AM3	AM4		DR1	DR2	DR3	DR4	DR5	DR6		CP1	CP2	CP3	CP4		FM1	FM2	FM3
Service Catalogue	C	A	R														C							C					C			
Business Plans	R	A	S	S	C	C	S	C	C	C	S	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Business SLR's	R	A	S	R	S	S	S	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
OLA's	C	A	S	S	R	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	C	C	C	C
Underpinning Contracts	C	A	C	S	S	R	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	C	C	C	C
SLA's	R	A	S	S	S	S	R	C	C	C	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	C	C	C	C
Service Review Reports	R	A	C	C	S	S	S	S	R	R	S	C					C							C					C			
RFC's (CSIP)	S	A	C	C				C	C	C	R	S	S	S	S	S	S			S	S			S	S	S	S	S	S	S	S	C
SLM Process Audit	R	A	C	C	C	C	C	C	C	C	C																					
Capacity Plan / Forecasts	I	S	C	C			C	C	C	C	C	S	S	S	S	S	S		C	S	S			A	R	S	S	S	S	S	S	C
Capacity Monitoring Reports	I	C		C			C	S	S	S	C	C		C	C	C	C		C	C	C		C	A	S	S	S	S	C	C	C	C
Modelling Results	I	C		C			C					C	C	C			C		C	C				A	S	S	R	S	C	C	C	C
Workload Analysis	I	C		C			C					C	C	C			C		C	C				A	S	S	S	S	C	C	C	C
Cap Man Process Audit																								A	C	C	C	C				
Availbility Forecasts	C	C		S			S	S	S	S	S	A	S	S	S	S	S		S	S	S	C	C	S	S	S	S	S	C	C	C	C
CFIA	I	C						S	S	S	S	A	S	S	S	S	S		S	S				C	C	C	C	C	C	C	C	C
FTA	I	C						S	S	S	S	A	S	S	S	S	S		S	S				C	C	C	C	C	C	C	C	C
Service Outage Analysis	I	C						S	S	S	S	A	S	S	S	S	C		C	C		C		C	C	C	C	C	C	C	C	C
TOP Report		C						S	S	S	S	A	S	S	S	S	C		C	C		C		C	C	C	C	C	C	C	C	C
Security Policy	I	S	C					C	C	C	C	A	S	S	S	S	S	S	S	S			S	C	C	C	C	C	C	C	C	C
Availability Plan	I	C	C					C	C	C	C	A	S	S	S	S	S	S	S	S			S	C	S	S	C	C	C	C	C	C
Av Man Process Audit												A	C	C	C	C																
Business Continuity Plan	R	R	S	S	S	S	S	S	S	S	S	S	S	S	S	S	A	R	S	S	S	S	S	C	C	C	S	S	C	C	C	C
CRAMM Report	I	S				S		C	C	C	C	S	S	S	S	S	A	S	S	R	S	S	S	C	C	C	S	C	C	C	C	C
Business Impact Analysis	R	S				S		C	C	C	C	S	S	S	S	S	A	S	R	S	S	S	S	C	C	C	S	S	C	C	C	C
IT Disaster Recovery Plan	S	S		S		S	S	C	C	C		S	S	S	S	S	A	S	S	S	R	R	S	C	C	C	C	C	C	C	C	C
DR Processes	C	C		C	C	C	C	C	C	C	C	S	S	S	S	S	A	S	S	S	R	R	S	C	C	C	C	C	C	C	C	C
DR Test Plan	C	C		C	C	C	C	C	C	C	C	S	S	S	S	S	A	S	S	S	R	R	R	C	C	C	C	C	C	C	C	C
DR Training Plan	C	C		C	C	C	C	C	C	C	C	S	S	S	S	S	A	S	S	S	S	R	R	C	C	C	C	C	C	C	C	C
ITSCM Process Audit																	A	C	C	C	C	C	C									
Budget / Balance Sheet	S	S		S			S	C	C	C		C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	A	R	S	C
Cost Models	I	I										I					I							I					A	S	R	C
Investment Analysis (ROI/ROCE)	R	C		S			S	C	C	C		I					I							I					A	S	S	S
Charging Policy	C	I		S			S	C	C	C		I					I							I					A	C	C	R
Charging Model	C	C		S			S	C	C	C		I					I							I					A	C	C	R
Fin Man Process Audit																													A	C	C	C

For an online demonstration of the ITIL eLearning course visit www.keylms.com/itilprodemo

For a free quotation please email: sales@ilxgroup.com

Key to abbreviations

A	*Process / Function Accountable*
R	*Process / Function Responsible*
S	*Process / Function Supporting*
C	*Process / Function Consulted*
I	*Process / Function Informed*

ISBN: 978-0-9544884-2-0
Published July 2006.

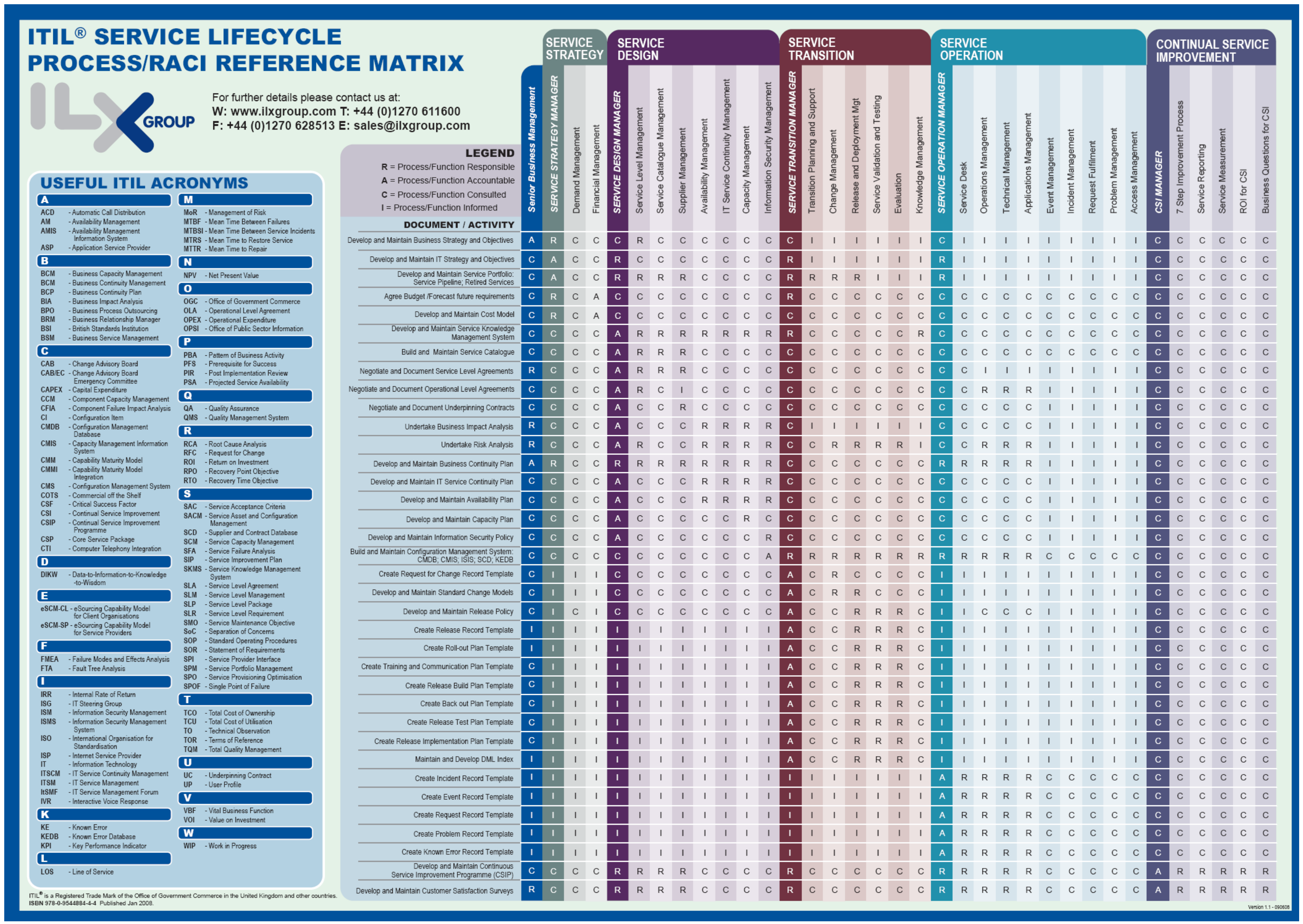
To order further copies, please e-mail sales@bestpracticebookstore.com or call +44 (0) 1270 611600

ITIL® is a registered Trade Mark, and a Registered Community Trade Mark of the Office of Government Commerce, and is Registered in the US Patent and Trademark Office.

ILX Group - Leaders in eLearning, Blended and Classroom ITIL Training Solutions

www.ilxgroup.com - Company Information
www.itiltraining.com - ITIL Training Options
www.prince2.com - Everything PRINCE2™
sales@ilxgroup.com - General Enquiries
Tel - +44 (0) 1270 611600

Figure 5.4: RACSI matrix for the ITIL V2 Service Support and Service Delivery Processes (see [ILX05])



ITIL® SERVICE LIFECYCLE PROCESS/RACI REFERENCE MATRIX

For further details please contact us at:
W: www.ilxgroup.com T: +44 (0)1270 611600
F: +44 (0)1270 628513 E: sales@ilxgroup.com

LEGEND
R = Process/Function Responsible
A = Process/Function Accountable
C = Process/Function Consulted
I = Process/Function Informed

		SERVICE STRATEGY			SERVICE DESIGN								SERVICE TRANSITION							SERVICE OPERATION										CONTINUAL SERVICE IMPROVEMENT					
DOCUMENT / ACTIVITY	Senior Business Management	SERVICE STRATEGY MANAGER	Demand Management	Financial Management	SERVICE DESIGN MANAGER	Service Level Management	Service Catalogue Management	Supplier Management	Availability Management	IT Service Continuity Management	Capacity Management	Information Security Management	SERVICE TRANSITION MANAGER	Transition Planning and Support	Change Management	Release and Deployment Mgt	Service Validation and Testing	Evaluation	Knowledge Management	SERVICE OPERATION MANAGER	Service Desk	Operations Management	Technical Management	Applications Management	Event Management	Incident Management	Request Fulfilment	Problem Management	Access Management	CSI MANAGER	7 Step Improvement Process	Service Reporting	Service Measurement	ROI for CSI	Business Questions for CSI
Develop and Maintain Business Strategy and Objectives	A	R	C	C	C	R	C	C	C	C	C	C	C	I	I	I	I	I	I	C	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain IT Strategy and Objectives	C	A	C	C	R	C	C	C	C	C	C	C	R	I	I	I	I	I	I	R	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain Service Portfolio: Service Pipeline; Retired Services	C	A	C	C	R	R	R	R	C	C	C	C	R	R	R	R	I	I	I	R	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Agree Budget /Forecast future requirements	C	R	C	A	C	C	C	C	C	C	C	C	R	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Develop and Maintain Cost Model	C	R	C	A	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Develop and Maintain Service Knowledge Management System	C	C	C	C	A	R	R	R	R	R	R	R	R	C	C	C	C	C	R	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Build and Maintain Service Catalogue	C	C	C	C	A	R	R	R	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Negotiate and Document Service Level Agreements	R	C	C	C	A	R	R	R	C	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Negotiate and Document Operational Level Agreements	C	C	C	C	A	R	C	I	C	C	C	C	C	C	C	C	C	C	C	C	C	R	R	R	I	I	I	I	I	C	C	C	C	C	C
Negotiate and Document Underpinning Contracts	C	C	C	C	A	C	C	R	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	C	C	C	C	C	C
Undertake Business Impact Analysis	R	C	C	C	A	C	C	C	R	R	R	R	C	I	I	I	I	I	I	C	C	C	C	C	I	I	I	I	I	C	C	C	C	C	C
Undertake Risk Analysis	R	C	C	C	A	R	C	C	R	R	R	R	C	C	R	R	R	R	I	C	C	R	R	R	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain Business Continuity Plan	A	R	C	C	R	R	R	R	R	R	R	R	C	C	C	C	C	C	C	R	R	R	R	R	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain IT Service Continuity Plan	C	C	C	C	A	C	C	C	R	R	R	R	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain Availability Plan	C	C	C	C	A	C	C	C	R	R	R	R	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain Capacity Plan	C	C	C	C	A	C	C	C	C	C	R	C	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain Information Security Policy	C	C	C	C	A	C	C	C	C	C	C	R	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	C	C	C	C	C	C
Build and Maintain Configuration Management System: CMDB; CMIS; ISIS; SCD; KEDB	C	C	C	C	C	C	C	C	C	C	C	A	R	R	R	R	R	R	R	R	R	R	R	R	C	C	C	C	C	C	C	C	C	C	C
Create Request for Change Record Template	C	I	I	I	C	C	C	C	C	C	C	C	A	C	R	C	C	C	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain Standard Change Models	C	I	I	I	C	C	C	C	C	C	C	C	A	C	R	R	C	C	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Develop and Maintain Release Policy	C	I	C	I	C	C	C	C	C	C	C	C	A	C	C	R	R	R	C	I	I	C	C	C	I	I	I	I	I	C	C	C	C	C	C
Create Release Record Template	I	I	I	I	I	I	I	I	I	I	I	I	A	C	C	R	R	R	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Create Roll-out Plan Template	I	I	I	I	I	I	I	I	I	I	I	I	A	C	C	R	R	R	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Create Training and Communication Plan Template	C	I	I	I	I	I	I	I	I	I	I	I	A	C	C	R	R	R	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Create Release Build Plan Template	C	I	I	I	I	I	I	I	I	I	I	I	A	C	C	R	R	R	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Create Back out Plan Template	C	I	I	I	I	I	I	I	I	I	I	I	A	C	C	R	R	R	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Create Release Test Plan Template	C	I	I	I	I	I	I	I	I	I	I	I	A	C	C	R	R	R	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Create Release Implementation Plan Template	C	I	I	I	I	I	I	I	I	I	I	I	A	C	C	R	R	R	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Maintain and Develop DML Index	I	I	I	I	I	I	I	I	I	I	I	I	A	C	C	R	R	R	C	I	I	I	I	I	I	I	I	I	I	C	C	C	C	C	C
Create Incident Record Template	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	A	R	R	R	R	C	C	C	C	C	C	C	C	C	C	C
Create Event Record Template	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	A	R	R	R	R	C	C	C	C	C	C	C	C	C	C	C
Create Request Record Template	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	A	R	R	R	R	C	C	C	C	C	C	C	C	C	C	C
Create Problem Record Template	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	A	R	R	R	R	C	C	C	C	C	C	C	C	C	C	C
Create Known Error Record Template	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	A	R	R	R	R	C	C	C	C	C	C	C	C	C	C	C
Develop and Maintain Continuous Service Improvement Programme (CSIP)	C	C	C	C	R	R	R	R	C	C	C	C	R	C	C	C	C	C	C	R	R	R	R	R	C	C	C	C	C	A	R	R	R	R	R
Develop and Maintain Customer Satisfaction Surveys	R	C	C	C	R	R	R	R	C	C	C	C	R	C	C	C	C	C	C	R	R	R	R	R	C	C	C	C	C	A	R	R	R	R	R

USEFUL ITIL ACRONYMS

A	
ACD	- Automatic Call Distribution
AM	- Availability Management
AMIS	- Availability Management Information System
ASP	- Application Service Provider
B	
BCM	- Business Capacity Management
BCM	- Business Continuity Management
BCP	- Business Continuity Plan
BIA	- Business Impact Analysis
BPO	- Business Process Outsourcing
BRM	- Business Relationship Manager
BSI	- British Standards Institution
BSM	- Business Service Management
C	
CAB	- Change Advisory Board
CAB/EC	- Change Advisory Board Emergency Committee
CAPEX	- Capital Expenditure
CCM	- Component Capacity Management
CFIA	- Component Failure Impact Analysis
CI	- Configuration Item
CMDB	- Configuration Management Database
CMIS	- Capacity Management Information System
CMM	- Capability Maturity Model
CMMI	- Capability Maturity Model Integration
CMS	- Configuration Management System
COTS	- Commercial off the Shelf
CSF	- Critical Success Factor
CSI	- Continual Service Improvement
CSIP	- Continual Service Improvement Programme
CSP	- Core Service Package
CTI	- Computer Telephony Integration
D	
DIKW	- Data-to-Information-to-Knowledge -to-Wisdom
E	
eSCM-CL	- eSourcing Capability Model for Client Organisations
eSCM-SP	- eSourcing Capability Model for Service Providers
F	
FMEA	- Failure Modes and Effects Analysis
FTA	- Fault Tree Analysis
I	
IRR	- Internal Rate of Return
ISG	- IT Steering Group
ISM	- Information Security Management
ISMS	- Information Security Management System
ISO	- International Organisation for Standardisation
ISP	- Internet Service Provider
IT	- Information Technology
ITSCM	- IT Service Continuity Management
ITSM	- IT Service Management
ItSMF	- IT Service Management Forum
IVR	- Interactive Voice Response
K	
KE	- Known Error
KEDB	- Known Error Database
KPI	- Key Performance Indicator
L	
LOS	- Line of Service
M	
MoR	- Management of Risk
MTBF	- Mean Time Between Failures
MTBSI	- Mean Time Between Service Incidents
MTRS	- Mean Time to Restore Service
MTTR	- Mean Time to Repair
N	
NPV	- Net Present Value
O	
OGC	- Office of Government Commerce
OLA	- Operational Level Agreement
OPEX	- Operational Expenditure
OPSI	- Office of Public Sector Information
P	
PBA	- Pattern of Business Activity
PFS	- Prerequisite for Success
PIR	- Post Implementation Review
PSA	- Projected Service Availability
Q	
QA	- Quality Assurance
QMS	- Quality Management System
R	
RCA	- Root Cause Analysis
RFC	- Request for Change
ROI	- Return on Investment
RPO	- Recovery Point Objective
RTO	- Recovery Time Objective
S	
SAC	- Service Acceptance Criteria
SACM	- Service Asset and Configuration Management
SCD	- Supplier and Contract Database
SCM	- Service Capacity Management
SFA	- Service Failure Analysis
SIP	- Service Improvement Plan
SKMS	- Service Knowledge Management System
SLA	- Service Level Agreement
SLM	- Service Level Management
SLP	- Service Level Package
SLR	- Service Level Requirement
SMO	- Service Maintenance Objective
SoC	- Separation of Concerns
SOP	- Standard Operating Procedures
SOR	- Statement of Requirements
SPI	- Service Provider Interface
SPM	- Service Portfolio Management
SPO	- Service Provisioning Optimisation
SPOF	- Single Point of Failure
T	
TCO	- Total Cost of Ownership
TCU	- Total Cost of Utilisation
TO	- Technical Observation
TOR	- Terms of Reference
TQM	- Total Quality Management
U	
UC	- Underpinning Contract
UP	- User Profile
V	
VBF	- Vital Business Function
VOI	- Value on Investment
W	
WIP	- Work in Progress

Version 1.1 - 090608

Figure 5.5: RACI matrix for the ITIL V3 Service Support and Service Delivery Processes (see [Ilx08])

"automatic operator" (ATOP) [Mah83], that allowed ringing people by phone and giving voice output for error messages already many years ago. Today, simple paging services via SMS or Mail have a similar function. They are mainly made for telling people the problem, so that standby operators at night can decide, whether it is worth to drive to the data center and fix things or not. Or so that they can call appropriate personnel quickly.

OSI Systems Management Functions Mostly defined and standardized in 1992 to 1996, OSI management's Systems Management Functions (see ISO 10164) are much more in line with EIMA. They describe 22 basic functions that can be implemented in OSI management's CMIP agents mainly to off-load central management stations. See [HAN99] for a summary on them, or the ISO standards for details.

- 1 Object management function
- 2 State management function
- 3 Attributes for representing relationships
- 4 Alarm reporting function
- 5 Event report management function
- 6 Log control function
- 7 Security alarm reporting function
- 8 Security audit trail function
- 9 Objects and attributes for access control
- 10 Usage metering function for accounting purposes
- 11 Metric objects and attributes
- 12 Test management function
- 13 Summarization function
- 14 Confidence and diagnostic test categories
- 15 Scheduling function
- 16 Management knowledge management function
- 17 Changeover function
- 18 Software management function
- 19 Management domains and management policy management function
- 20 Time management function
- 21 Command sequencer
- 22 Response time monitoring function

OSI management is used in telecommunications devices today, but we could not identify any particular resoures of interest to IT management, that indeed include these functions.

In fact, OSI management was not successful in widespread IT management, mainly for reasons of complexity. Instead, Internet Management and SNMP prevailed in our domain. With Remote Monitoring (RMON) in RFC 2819:2000 and the creation of the Distributed Management Event MIB (RFC 2981:2000) and Expression MIB (RFC 2982:2000), there are similar functions for a small subset of the OSI systems management functions. They all shift bulk operations on large amounts of management data closer to the deviced that are managed.

OSI systems management functions define functions to be built-in at design time into a managed system. OSI management also includes recommendations on delegation of tasks. The current descriptions are a welcome list of machine capabilities, and will be included in the catalog of machine capabilities. Doing so, we offer the available knowledge in this "pattern catalog" of EIMA, and leave it to designers of future mangement automation systems, to what extent they reuse this readily available knowledge. It is undoubted, that if we had many of the 22 functions listed above in today's managed resources in their standardized form, then this would have positively enabled management automation by local processing.

Management by delegation with mobile agents With roots in business administration and being one of multiple ways of leadership of *people*, the principle of management by delegation was proposed in IT management in order to manage more dynamic environments. Main technologies are mobile management agents being based on mostly platform-independent code (e.g. Java), which are sent to managed resources and reside there temporarily. Management by delegation is used relatively rarely and mostly in a pull fashion, examples include Java applets or ActiveX code in the WWW, offered to people to run certain functions like local malware scanning or local software version checks right on their local computers. Usually, mobile code is signed and authenticated, with the user having to authorize access to sensitive operations like local file access.

In comparison to EIMA, that deals with management automation in terms of shifting work from human managers to machine managers (see the most fundamental human/machine decision in EIMA function allocation), the main goal of management by delegation is to relieve *central management stations* from collecting and processing management data and instead having a portion of this work done locally at the managed resources. With that respect, management by delegation shifts tasks to different entities within the set of machines (and therefore does change function allocation), but hardly from a human to machines.

An additional difference is, that management by delegation with mobile code/mobile agents typically aims at distributing *arbitrary* management code, that did not even exist before on the managed resource. This makes it very flexible as any new functionality can be brought in. At the same time, hoewever, it is also a reason for skepticism on the side of resource administrators, for they have no control on what the mobile agent actually does. Despite of all taken security precautions like signed code and necessary user confirmation, there remain uncertainties on the actual origin and content of the mobile agent.

EIMA in contrast follows the approach, that necessary functionality on the managed resource (including management automation routines) is taken care of at design time of the resource. If later, changes or additions are necessary to the local management automation code, then a defined update/upgrade process should replace the version residing at the managed resource, instead of a mobile agent temporarily changing the behavior.

Explicit tagging of roles, whether they are allocated to humans or machines

ITSM frameworks In ITSM frameworks like ITIL, CoBit, MOF, all roles are implicitly assumed to be impersonated by humans. The intended implementation is via humans being trained on vocabulary and process knowledge. The refinement of ITIL into a control system with human and machine roles, and common data formats does not take place so far.

IT management tools The concept of machine roles (proxies for humans) does not exist. So, machine roles are not tagged.

Managed resources The concept of human roles and machine roles does hardly exist. In some cases (e.g. the Linux Operating System), application-specific users are created for major server applications (e.g. a `postgres` or `apache` user) but there is nothing at the level of abstraction of EIMA.

Exceptions include special pre-defined accounts or groups in OS's (e.g. root or Administrator) and network equipment, and special application specific users (postgres, apache) so that authorization decisions (file access, process ownership) can be given to an explicit machine role.

Function allocation at loop, loop step or function level

ITSM frameworks All ITSM frameworks have to fight complexity. They are written for humans and for printed publication. So, they do contain function allocation only for high-level tasks. Typically, processes are broken down one level or the documents are used as proxies for the tasks that they imply. Yet, even at this high-level task layer, they fill pages, as seen in the Figures for ITIL RACI charts.

IT management tools Function allocation is not part of tools as of today. Loops and loop steps are not part of today's tool infrastructures, so function allocation is not applicable at these levels. BPMN charts can contain swim lanes to express static function allocation to certain roles.

Summary and Requirements Check

ITSM frameworks commonly define roles and responsibilities in RACI charts, that grow large and complex, even though they only cover one refinement step in the processes. They are mainly meant as a means for documentation and training people, instead of automation, as they only include human roles.

The standards stop at this level of abstraction, as they already more complex than can be comprehended by an average reader. This shows the need for tool support for these matrixes. And simple mockup animations that show processes and actually allow to experience, what these charts would imply.

If discrete event simulation would be used, than given certain frequencies of events/tasks/functions, we could investigate how often certain roles would be involved. This would give excellent support for automation, because it would make us aware of hotspots in the distribution of tasks.

Overall, even without an explicit check matrix on the EIMA requirements, we can see that IT management cannot contibute much on this aspect, apart from RACI charts.

5.3.2 R3.2 Automation levels and standards

Defined Levels of automation

The following non-exclusive list of related work was found on matters of automation levels.

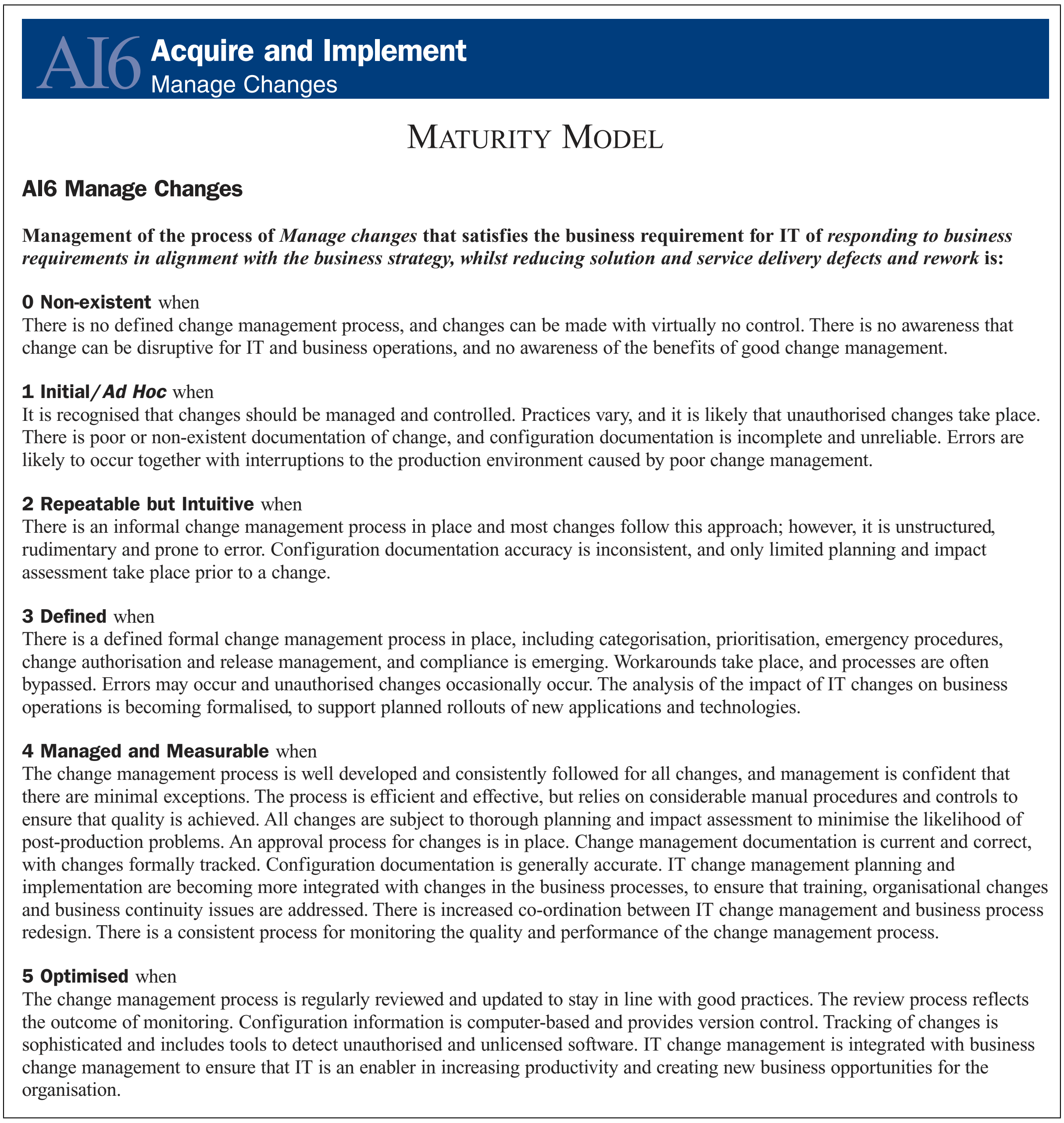
SAMM, 1993 The System Administration Maturity Model (SAMM) [Kub93] is an adaptation of the CMM to make it more relevant for system and network administration organizations. The SAMM seeks to describe the key processes required for a system and network administration organization to flourish into higher levels of process maturity and enjoy the benefits associated with mature organizations such as high quality products and services produced on time, and within budget limits.

Cobit 4.1, 2007 CoBit [Ins07] defines five standard maturity levels. The specialty of CoBit is, that the maturity levels are indeed defined and elaborated for each individual CoBit task, see Fig. 5.6 for an example.

ITIL V3, 2007 ITIL V3 [Ogc07] does only have a general Service Management process maturity framework (which is aligned with the CMMI-SVC, which in turn in based on the CMMI), that is defined in Appendix H of the volume on "Service Design". It contains the general maturity levels inspired by the CMMI: Initial (Level 1), Repeatable (Level 2), Defined (Level 3), Managed (Level 4), Optimizing (Level 5) and considers five aspects in each level: Vision and steering, Process, People, Technology, Culture.

EIMA clearly concentrates on the Technology aspect, where the five levels are described in ITIL V3 like this:

- Initial (Level 1)
  - Manual processes or a few specific, discrete tools (pockets/islands)
- Repeatable (Level 2)
  - Many discrete tools, but a lack of control
  - Data stored in separate locations
- Defined (Level 3)
  - Continuous data collection with alarm and threshold monitoring
  - Consolidated data retained and used for formal planning, forecasting and trending
- Managed (Level 4)
  - Continuous monitoring measurement, reporting and threshold alerting to a centralized set of integrated toolsets, databases and processes
- Optimizing (Level 5)
  - Well-documented overall tool architecture with complete integration in all areas of people, processes and technology



AI6 **Acquire and Implement**
Manage Changes

MATURITY MODEL

AI6 Manage Changes

Management of the process of *Manage changes* that satisfies the business requirement for IT of *responding to business requirements in alignment with the business strategy, whilst reducing solution and service delivery defects and rework* is:

0 Non-existent when
There is no defined change management process, and changes can be made with virtually no control. There is no awareness that change can be disruptive for IT and business operations, and no awareness of the benefits of good change management.

1 Initial/*Ad Hoc* when
It is recognised that changes should be managed and controlled. Practices vary, and it is likely that unauthorised changes take place. There is poor or non-existent documentation of change, and configuration documentation is incomplete and unreliable. Errors are likely to occur together with interruptions to the production environment caused by poor change management.

2 Repeatable but Intuitive when
There is an informal change management process in place and most changes follow this approach; however, it is unstructured, rudimentary and prone to error. Configuration documentation accuracy is inconsistent, and only limited planning and impact assessment take place prior to a change.

3 Defined when
There is a defined formal change management process in place, including categorisation, prioritisation, emergency procedures, change authorisation and release management, and compliance is emerging. Workarounds take place, and processes are often bypassed. Errors may occur and unauthorised changes occasionally occur. The analysis of the impact of IT changes on business operations is becoming formalised, to support planned rollouts of new applications and technologies.

4 Managed and Measurable when
The change management process is well developed and consistently followed for all changes, and management is confident that there are minimal exceptions. The process is efficient and effective, but relies on considerable manual procedures and controls to ensure that quality is achieved. All changes are subject to thorough planning and impact assessment to minimise the likelihood of post-production problems. An approval process for changes is in place. Change management documentation is current and correct, with changes formally tracked. Configuration documentation is generally accurate. IT change management planning and implementation are becoming more integrated with changes in the business processes, to ensure that training, organisational changes and business continuity issues are addressed. There is increased co-ordination between IT change management and business process redesign. There is a consistent process for monitoring the quality and performance of the change management process.

5 Optimised when
The change management process is regularly reviewed and updated to stay in line with good practices. The review process reflects the outcome of monitoring. Configuration information is computer-based and provides version control. Tracking of changes is sophisticated and includes tools to detect unauthorised and unlicensed software. IT change management is integrated with business change management to ensure that IT is an enabler in increasing productivity and creating new business opportunities for the organisation.

Figure 5.6: Fine-grained description of process-individual maturity levels in Process AI6 “Manage Changes” in CoBit 4.1, see [Ins07]

IBM Autonomic Computing Initiative, 2001 and 2006 Unlike previously listed work, the ACI defined evolution steps for autonomic *systems* or systems components, instead of processes. For "autonomic", they use the following definition [Hor01]:

> Autonomic systems must adapt to environmental changes and strive to improve their performance over time. They must be robust and be able to routinely overcome internal component failures. Autonomic systems must interact and communicate with other systems in a heterogeneous computing infrastructure. Their approach to building autonomic systems is based on combining autonomous intelligent agents in a well-structured way. This approach mirrors the structure of the human brain wherein there are clearly defined, function-specific processing centers connected by forward and backward communication channels and adaptive feedback loops.

According to their view, autonomic systems reduce the management interface to setting/getting high-level and mostly business-oriented policies, goals and constraints, without direct get/set operations to system variables. The self-star mechanism will take care of the variables and provide high-level managed objects for monitoring, such as health status. The system is more service-oriented with its implementation hidden inside its shell. Sensors provide context information for local processing.

On an evolutionary course to this goal, IBM in [IC01] defined five maturity levels in autonomic computing: basic - managed - predictive - adaptive - autonomic, see Fig. 5.7.

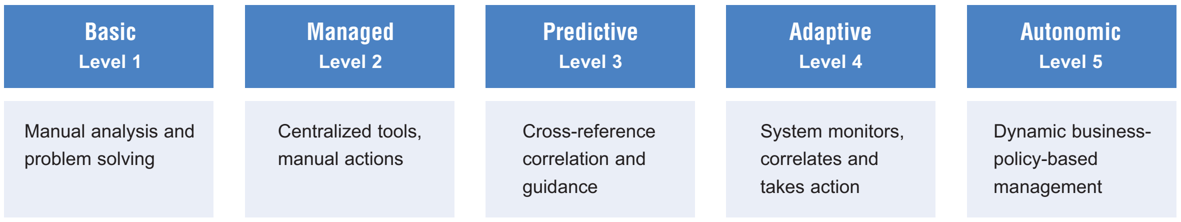

Basic
Level 1
Manual analysis and problem solving
Managed
Level 2
Centralized tools, manual actions
Predictive
Level 3
Cross-reference correlation and guidance
Adaptive
Level 4
System monitors, correlates and takes action
Autonomic
Level 5
Dynamic business-policy-based management

Figure 5.7: Early automation levels by IBM, see [IC01]

They are not explicitly associated with anything (such as autonomic elements, IBM's concept of a loop), but with "systems". They provide a verbal description of maturity on the course to autonomic IT management systems, that are ultimately managed by humans setting business policies and goals. These levels are mentioned as intermediary steps in the evolution of business-policy driven IT management. These levels have not been used as properties or certifications of products, that were part of this development.

In 2006, IBM in [IBM06] refined the levels to a three-dimensional metrics, see Fig. 5.8. There, autonomic maturity can evolve in three dimensions:

- Automating more functions as the maturity level increases
- Applying automated functions to broader resource scopes
- Automating a range of tasks and activities in various IT management processes

The individual axis of the cube are: Axis 1 "Maturity", which is aligned with the loop structure:

- *Manual level:* IT professionals perform the management functions.
- *Instrument and monitor level:* systems management technologies can be used to collect details from manageability endpoints, helping to reduce the time it takes for the administrator to collect and synthesize information as the IT environment becomes more complex.

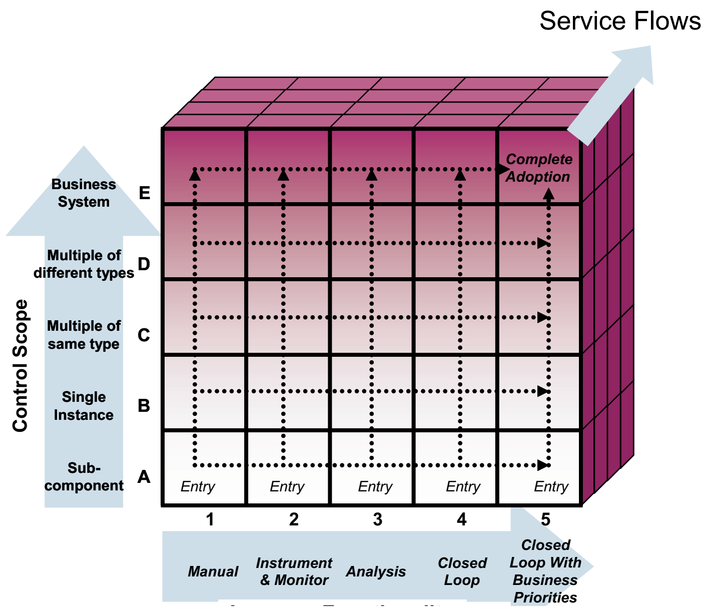


Figure 5.8: Levels of automation, now defined along a three-dimensional cube to take into account automation maturity, control scope, and the diverse processes/service flows, see [IBM06]

- *Analysis level:* new technologies are introduced to provide correlation among several manageability endpoints. The management functions can begin to recognize patterns, predict the optimal configuration and offer advice about what course of action the administrator should take. As these technologies improve, and as people become more comfortable with the advice and predictive power of these systems, the technologies can progress to the closed loop level.
- *Closed loop level:* the IT environment can automatically take actions based on the available information and the knowledge about what is happening in the environment.
- *Closed loop with business processes level:* business policies and objectives govern the IT infrastructure operation. Users interact with the autonomic technology tools to monitor business processes, alter the objectives or both.

It also features a model, that relates the levels of automation to another five levels of resource management scope (Axis 2: “Control Scope”):

- *Subcomponent level:* portions of resources are managed, such as an operating system on a server or certain applications within an application server.
- *Single instance level:* an entire standalone resource is managed, such as a server or complete application server environment.
- *Multiple instances of the same type level:* homogeneous resources are managed, typically as a collection, such as a server pool or cluster of application servers.
- *Multiple instances of different types level:* heterogeneous resources are managed as a subsystem, such as a collection of servers, storage units and routers or a collection of application servers, databases and queues.
- *Business system level:* a complete set of hardware and software resources that perform business processes is managed from the business process perspective, such as a customer relationship management system or an IT change management system.

A third axis “Service flows” covers the service flows, which are the IT management processes. For EIMA this would mean the AutoITIL processes.

IT management tools Common IT management tools do not document, nor measure maturity levels. They also hardly offer a multiple configurable level of automation. If they do, then there are no standardized levels of automation. Each tool might use different automation levels. The tools themselves are not integrated into the education of administrators. Compare this with pilots for example, who learn by heart, how autopilots or an instrument landing system works, how they can use them, and what levels of automation they allow.

Summary and Requirements Check

Levels of automation in terms of multiple technical automation levels do not occur in existing IT management related work. Instead, the frameworks see an evolution that starts with todays ad hoc or repeatable, or defined processes, which gradually become more and more mature, automated and optimizing. Based on the dominance of the CMM [Ins95] in maturity levels, all frameworks align their levels to the CMM general model. In general, the more automated a process is, the higher it is ranked on the diverse maturity metrics.

In terms of maturity, EIMA needs as input workflows at level repeatable or defined, so that the actual process definition has already taken place and so that workflows are available which serve as input for EIMA. It can then help to take processed further to the managed level, where automation replaces or supports human activity.

Only the redefinition of IBM’s levels of automation comes close to the objectives of EIMA: Levels of automation for technology, that indeed describe different capabilities of loop automation.

5.3.3 R3.3 “Good/best” automation for a task

Advice on automation feasibility

CoBit 4.1 In CoBit [Ins07], there is Process A1 (“Identify Automated Solutions”) that tells to identify technically feasible and cost-effective solution, as well as to undertake feasibility studies as defined in the development standards, and to approve (or reject) requirements and feasibility study results. It says to check feasibility based on requirements, risk assessment, impact assessment, and cost-benefit studies. This is so generic that it is of little actual use.

IT management tools We do not know of any tools that given a certain problem, give advice on automation feasibility. What is occasionally built into IT systems are warnings, that occur and persist, in case their internal management automation routines especially those dealing with security (firewalls, virus scanners, automatic updates) are turned off. See e.g. the Windows Security Center in Windows XP and up.

Summary and Requirements Check

IT management lacks general advice on a good/best level of automation.

5.3.4 R3.4 Dynamic function allocation at operation phase

Dynamic function allocation means, that an administrator can control at run-time the level of automation for a certain task. In other words, he can switch among several automation levels, whether a certain action is performed by himself, or assisted by the machine, or by the machine itself. It would mean to delegate (pre-defined) management tasks either to the computer or to do them oneself. More sophisticated dynamic allocation schemes would also offer intermediate levels between these extremes. A human IT admin could gradually decide to which extent he wants to be involved in a certain decision and informed about incoming (sensor input) and outgoing (actor output, actions) data.

As a source, that proves such controllable automation to be worthwhile, see the following book.

Limoncelli, The practice of system administration, 2002 This contribution [Lim02] covers the daily work of system administrators. It is written by system administrators, who document a number of their best practices and general knowledge gained during their professional work. Most interestingly, this book describes how system administrators at the same time like and dislike their work being automated. This is one of the main influence factors to plan future systems with multiple levels of automation and supports Sheridan's argument to add a function allocation step. As another result we see, that administrators like different automation levels at different times. We conclude from this fact, that choosing the right level of automation is a task of operators not designers.

Dynamic function allocation

ITSM frameworks As ITSM frameworks do not include the actual tools, they do not cover dynamic function allocation.

IT management tools Dynamic function allocation in the sense of EIMA (with a central configuration facility auf the level of automation for management automation routines) does not exist in current IT management tools. An early attempt were systems management functions introduced with OSI, but they were rather basic functions intended to be implemented into OSI-managed resources. Concepts on delegation of control were invented there as well, but to the best knowledge of the author an actual impact in IT management and wide-spread distribution could never be witnessed with the advent of SNMP and internet management being the more successful protocol and framework in achieving actual distribution.

Daring a broader view, we can say that management tools mainly do monitoring. While there exist simple analysis routines, like threshold warnings and basic event correlation mechanisms, management software is not really meant for closed-loop control. Instead, big portions of analysis and all planning tasks are left to administrators.

This has changed only recently, when virtualization management tools (like VMware vCenter Server) actually had the ability to deal with (virtual) machines in a whole different way than was possible before, in conjunction with shared storage and server machines that have less and less internal state. They introduced features like live migration of VMs between hosts, hardware pooling for VMs, or snapshots of running systems, and automated fail-over configurations where hot standby servers accept VMs from other machines. Here, indeed automation features can be enabled/disabled once all requirements are met, and the enormous commercial success shows that such solutions are really appreciated by customers. Automation enabling/disabling happens at a feature level, not at the level of loops, and not many intermediate levels of computer-human cooperation.

Managed resources In managed resources, there are various places, where automation (at a very small scale and in each case restricted to a single function) indeed can be switched between different modes. A non-exhaustive list includes:

- auto-updates in applications
  - Open Services Gateway initiative (OSGi) (Firefox and plug-ins, Eclipse)
  - Windows Update (automatic, download only, manual)
  - simple new version notifiers at application start-up (most frequent case)
- BIOS detection routines
  - many detection routines can be enabled or disabled here
- scheduled tasks
  - virus scans
  - backups
  - autostart (run with every boot process) Windows, Linux init scripts

In general, if periodic activities are required, they are typically performed at application startup (user action necessary), or schedule-based in a time-based background job scheduler (cron, task planner).

Academic work In academic work, policy-based management (see e.g. [MSL08, CCK$^+$07, KCES07]) is close to the idea of dynamic function allocation. In this regard, two kinds of interaction between policies (rules) and automation can be imagined:

- Policies that control automation, e.g. that control function allocation in a management automation system. They may for example ensure, that automation is turned on at night shift, or that a lower level of automation is activated once automation failures are noticeable for a certain set of resources.
- A policy-based management system is a management automation system itself. Then ECA rules would determine the actions, that need to happen to react to certain sensor input. In this case policies would more directly control the cooperation between humans and machines.

Another relatively new academic paradigm in IT management automation is to jump directly to the "optimizing" maturity level and to do IT management by utility functions being fed/parameterized to an automation system, see e.g. [dV09, CSW$^+$08, CKS09]. In fact, if we knew the desired state in each situation, the observed state, the alternate plans from observed state to desired state, and also overall utility of all the alternative plans, then we could put all this knowledge into the formulation of a utility function. Based on the utility value of each individual plan, one can be selected and executed. Utility functions would then be a generic step in automation, just like the value judgment function in RCS-4 (see Fig. 4.14).

Indeed, if such a universal function could be found for an individual problem, then it would enable optimal automation, as the function by definition would return the optimal plan in each situation. The difficulty, however, lies in an appropriate, versatile definition of a utility function. It would have to trade-off many influences and constraints (economic, technical, policies). While there is hope, that defined service levels can be an input for the desired state in each situations, there are few recommendations on such functions. If they were proposed they would be hard to verify, hard to manage/change, and probably hard to predict the likely effects. For simple situations close to resources with low complexity at an operational level, they would probably be well applicable, but this would be less and less true, once decision also contain strategic, and tactic influences on decisions.

Summary and Requirements Check

As stated before function allocation is not built into IT management systems as an explicit entity. And so, dynamic function allocation is also not present in the way, that EIMA would require it. Instead, we found many automation piece parts, where individual small functions in managed systems or management systems could be configured regarding their level of automation. They have a high heterogeneity in models, syntax and semantics, in fact each tool handles this aspect differently. An "automation integration" does not take place in general.

Utility functions are proposed in academia as alternative solutions to dynamic function allocation. They are based on the belief that all value can be attained objectively and that a mathematical model would cover reality to such a great extent, that the remaining differences are uncritical. From the point of view of the author of this thesis, this assumption may hold in simple situations but it will fail once trade-offs are necessary, that have not been taken into account during the design of the utility function.

5.3.5 R3.5 Computer-aided modeling of function allocation

Application to model function allocation and a format for data exchange

Current IT management automation does not use special applications or file formats for modeling function allocation. Instead, RACI charts are created in spreadsheet applications, with task and role identifiers written in a way, so that they can be easily interpreted by humans.

In tools and managed resources, function allocation is spread to proprietary configuration files or configuration databases, or not even modeled explicitly (e.g. in BIOS). If automation can be controlled, then there are typically only manual and automatic modes. Many of the shades of gray of automation (list of pre-rated alternatives, recommendation, confirm mode) are typically not implemented.

Summary and Requirements Check

Current IT management does not live up to the expectations of EIMA with respect to modeling function allocation.

5.3.6 Summary

In current IT management, there is no true human-machine-integration in terms of function allocation, which would enable delegation of tasks between humans and machines.

Instead, we could identify split worlds with not much alignment between three main entities:

- People and Processes
- Standard frameworks
- Management tools, process tools, managed resources

RACI charts existing in process frameworks for human roles are not taken over to IT management tools. Instead they are interpreted by humans and used for teaching or human reference. Standard frameworks do not include the whole side of IT management / process management tools and so do not assign responsibilities to them. Managed resources and management tools may have controllable on/off-automation for very specific aspects, but there is no integration, no standardization, no coordination.

And rarely any shades of gray of automation, where a human is supported but stays in control. All in all, we can say, that function allocation is not among the concepts in current IT management automation.

The most promising result for EIMA is the three-dimensional definition of IT management levels of automation, that was defined by the ACI in 2006. Unfortunately, it has never been actually used outside of IBM Global Services internal work, at least to the best knowledge of the author of this thesis.

5.4 Existing knowledge and concepts from Systems Engineering

As we have seen, current IT management leaves a lot to be desired in terms of function allocation. To aid in achieving a better way with *engineered* IT management automation, where function allocation is chosen intentionally, at design and/or operations time, we need input from the systems engineering domain. The review for related work and existing knowledge was limited to this domain, as it is shared among many application domains (e.g. military, aerospace systems, vehicles, medical devices, industrial plants) and well combines the essential core of knowledge, that is most useful in other application domains.

System examples To give the reader a better idea and allow to concentrate on a subset of these, the function allocation methods around the following management automation routines from systems engineering are our main source of inspiration:

- pilot assistance in airliners
  - autopilot / flight director
  - autothrottle / autothrust
  - flight management system
  - central and integrated systems for warnings, checklists and fault handling: e.g. Electronic Centralized Aircraft Monitoring (ECAM)/Engine Indication and Crew Alerting System (EICAS)
- driver assistance systems in cars
  - supervisory, open-loop assistance systems: e. g. lane departure warning, parking proximity alarm
  - closed-loop assistance systems: e. g. adaptive cruise control, auto-braking assistant
  - function allocation in different types of transmissions: automatic, semi-automatic, manual gearboxes

We believe, that these systems are a valid and useful source of inspiration for the following reasons:

- Aircraft and car design is highly influenced by risks in terms of safety incidents in cases of systems faults or operator mistakes. Incidents in current and future large IT installation may not be as life-threatening, but in terms of insured value for incidents and accidents will come close. To name an example: A security incident with credit cards data stolen from millions of users, or a 5 min interruption in the IT system for a national stock exchange or the online shop for a major online vendor will be comparable to risks, as caused by harms to humans and property related to vehicles such as cars and planes. This thinking in terms of insured risks (and therefore money) makes such matters comparable, that before seemed unrelated.
- Airplane design well covers a field, that is highly regulated, standardized and based on extensive training and up-to-date engineering methods. We can rely on the fact, that with its past funding and activities in basic research it is a domain with few resource limitations in terms of research.

Car design shares most of these properties, but in terms of users (car drivers) comes closer to IT administrators: the system vendor has to take into account human-related properties such as their non-standardized training and professional education, occasional use, and less standardized communications.

- IT administration as well as flying an airliner are tasks that are done by more than one human, so matters of task sharing play a role. Also in terms of basic cognitive tasks (monitoring system state, handling incidents and faults, working around temporary disturbances, understanding complex cause-effect relations in complex systems, optimizing operational decisions, trading-off between reaction alternatives) the domains share a remarkable number of similar activities.

5.4.1 R3.1 Static binary function allocation concepts

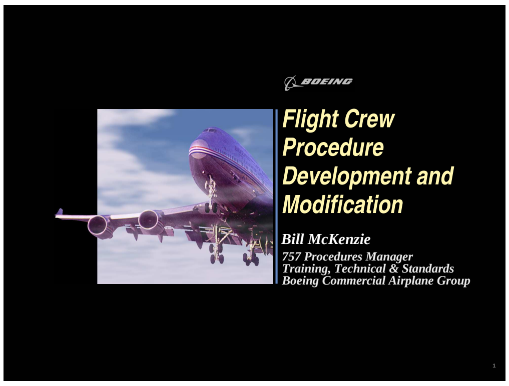
Scheme for human/machine function allocation and accountability for functions explicitly

Systems engineering of human-machine-systems is well aware of a necessary function allocation step. Human factors engineering of such systems defines roles for its operators, such as a car driver or airplane crew personal. In contrast to typical IT management systems, human factors engineering analyzes human task performance in studies. This way, human behavior and performance in situations like stress, boredom, awakeness, sleepiness, enthusiasm is carefully examined and researched. In a similar way, past experience on human performance in incidents and accidents is collected and analyzed, including statistical evaluations. This rarely happens for IT administrators and only recent work took notice of its necessity (e.g. by the team around Paul Maglio at IBM Human Systems Research at the IBM Almaden Research Center.)

The responsibilities of roles (e.g. for a cockpit crew of two, one pilot flying and one pilot not flying) are defined in an operating philosophy. These are general design guidelines on the cooperation of human operator roles and the designed system, may it be a car or an airplane. Explicitly modeling responsibilities of humans cannot be "forgotten" as airplanes are required to be accompanied by a whole number of documentation for their operation by aviation authorities (Federal Aviation Authority (FAA) for the United States, Joint Aviation Authority (JAA) for Europe, for example) : standard operating procedures, crew procedures and checklists, training material, to name only a few. As a specialty in aircraft design and related documentation, all procedures are written for both, the normal procedures (standard operating procedures) as well as abnormal / emergency conditions (abnormal procedures). See the extract of a presentation on "Flight Crew Procedure Development and Modification" by the Procedures Manager for the Boeing 757, Bill McKenzie in Fig. 5.9.

For cars, this is applicable to less extent, as based on far less responsibility for human lives they are allowed to be driven by far less well trained people. Still, driving a car is pre-planned, simulated and tested, and driver assistance systems are designed to aid drivers in extreme situations especially without prior training. Car models in the EU need a Certificate of Conformity for a type approval, which regulates by law the existence of driver assistance and safety systems, as well as other properties, that are of less importance here. The majority of advanced assistance systems / management automation systems however are introduced not based on legal requirements, but as an offer to customers for improved safety, comfort, or simply ease of use.

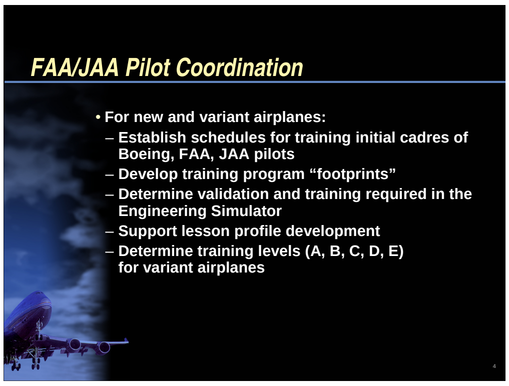
In the past, function allocation and responsibilities were described in extensive documentation, that was written in natural language for the involved system designers. Only recently, the systems engineering domain recognized the need for machine-interpretable modeling of systems engineering (in contrast to computer aided design, manufacturing and engineering, that was already established before). See R3.5



BOEING

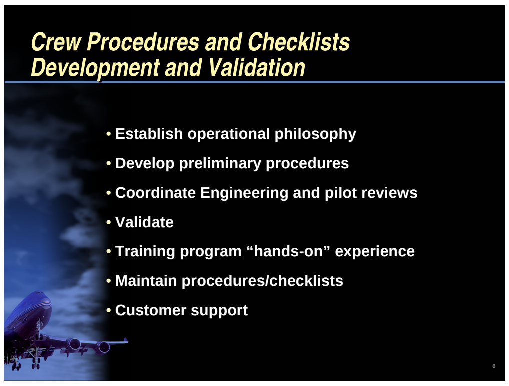
Flight Crew Procedure Development and Modification

Bill McKenzie
757 Procedures Manager
Training, Technical & Standards
Boeing Commercial Airplane Group



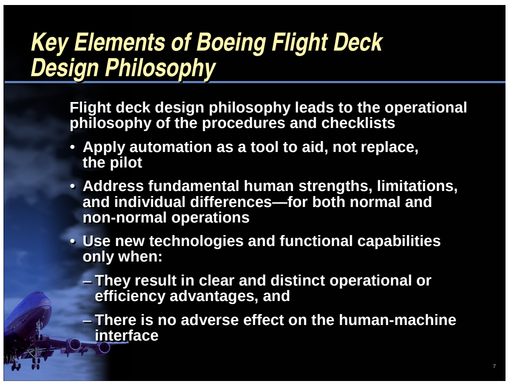
FAA/JAA Pilot Coordination

- For new and variant airplanes:
  - Establish schedules for training initial cadres of Boeing, FAA, JAA pilots
  - Develop training program "footprints"
  - Determine validation and training required in the Engineering Simulator
  - Support lesson profile development
  - Determine training levels (A, B, C, D, E) for variant airplanes



Crew Procedures and Checklists Development and Validation

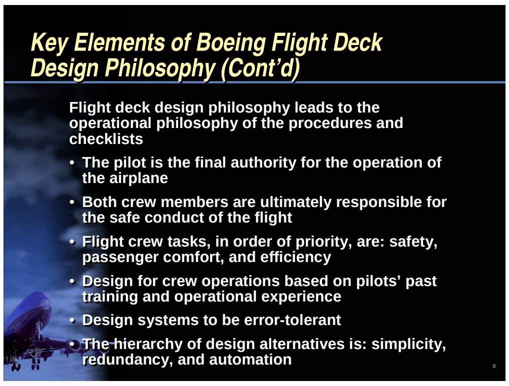
- Establish operational philosophy
- Develop preliminary procedures
- Coordinate Engineering and pilot reviews
- Validate
- Training program "hands-on" experience
- Maintain procedures/checklists
- Customer support



Key Elements of Boeing Flight Deck Design Philosophy

Flight deck design philosophy leads to the operational philosophy of the procedures and checklists

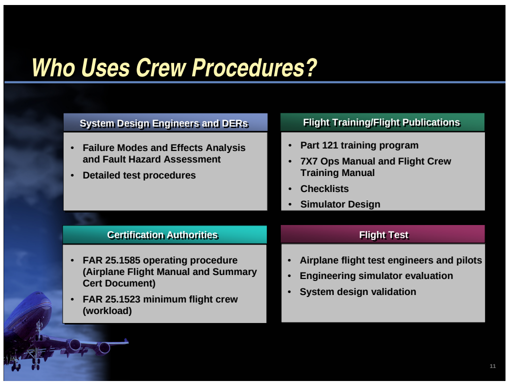
- Apply automation as a tool to aid, not replace, the pilot
- Address fundamental human strengths, limitations, and individual differences—for both normal and non-normal operations
- Use new technologies and functional capabilities only when:
  - They result in clear and distinct operational or efficiency advantages, and
  - There is no adverse effect on the human-machine interface



Key Elements of Boeing Flight Deck Design Philosophy (Cont'd)

Flight deck design philosophy leads to the operational philosophy of the procedures and checklists

- The pilot is the final authority for the operation of the airplane
- Both crew members are ultimately responsible for the safe conduct of the flight
- Flight crew tasks, in order of priority, are: safety, passenger comfort, and efficiency
- Design for crew operations based on pilots' past training and operational experience
- Design systems to be error-tolerant
- The hierarchy of design alternatives is: simplicity, redundancy, and automation



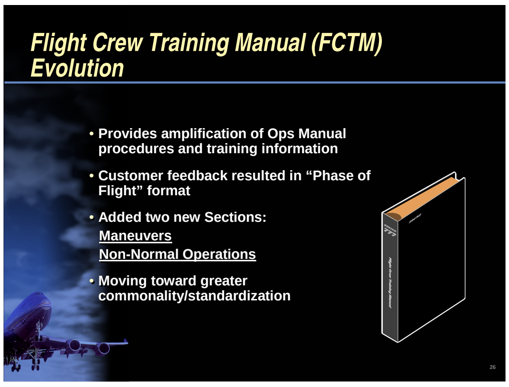
Who Uses Crew Procedures?

System Design Engineers and DERs
- Failure Modes and Effects Analysis and Fault Hazard Assessment
- Detailed test procedures

Flight Training/Flight Publications
- Part 121 training program
- 7X7 Ops Manual and Flight Crew Training Manual
- Checklists
- Simulator Design

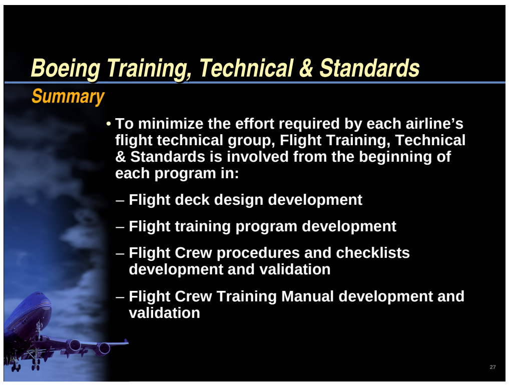
Certification Authorities
- FAR 25.1585 operating procedure (Airplane Flight Manual and Summary Cert Document)
- FAR 25.1523 minimum flight crew (workload)

Flight Test
- Airplane flight test engineers and pilots
- Engineering simulator evaluation
- System design validation



Flight Crew Training Manual (FCTM) Evolution

- Provides amplification of Ops Manual procedures and training information
- Customer feedback resulted in "Phase of Flight" format
- Added two new Sections:
  Maneuvers
  Non-Normal Operations
- Moving toward greater commonality/standardization



Boeing Training, Technical & Standards

Summary

- To minimize the effort required by each airline's flight technical group, Flight Training, Technical & Standards is involved from the beginning of each program in:
  - Flight deck design development
  - Flight training program development
  - Flight Crew procedures and checklists development and validation
  - Flight Crew Training Manual development and validation

Figure 5.9: Airplane industry designs procedures and matters of function allocation together with their artifacts from the beginning of systems design. They include matters of function allocation and non-normal operations. Source: `http://human-factors.arc.nasa.gov/eas/download/EAS_Symposium_Presentations/Procedure_Development.pdf`

below for recent standardization efforts to overcome proprietary, tool-dependent modeling of function allocation and replace it with standard models.

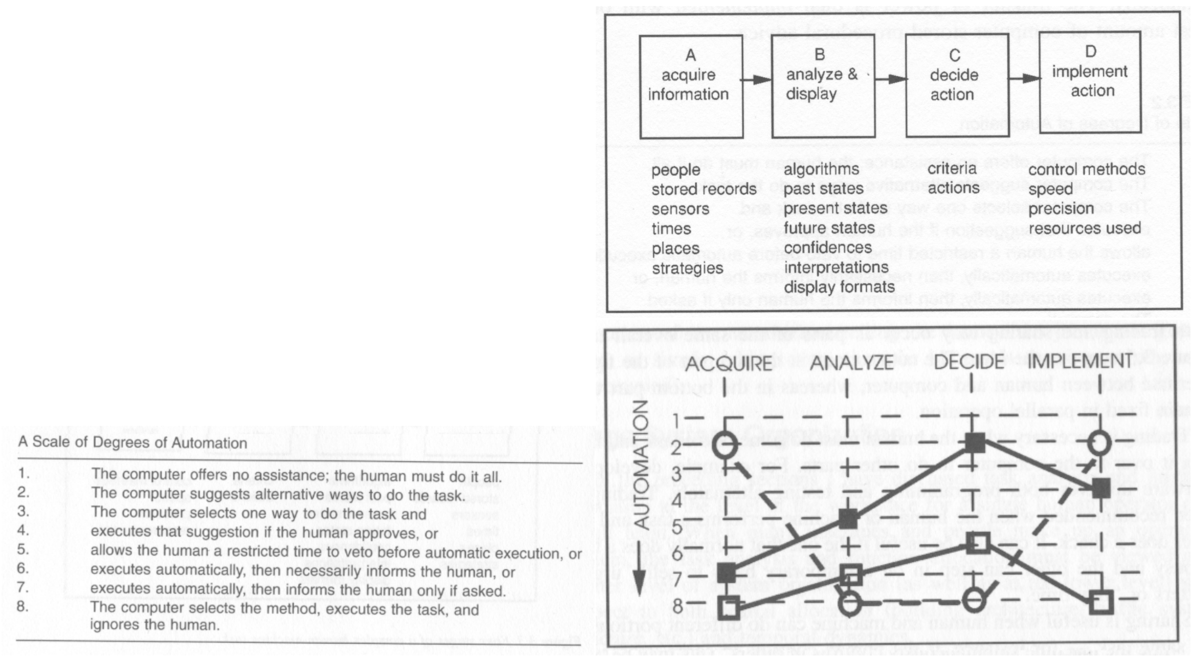
Explicit tagging of roles, whether they are allocated to humans or machines

We could not get access to actual system design material for a car or an aircraft, but based on the necessary documentation for such systems, that needs to be provided, it is for sure, that human and machine roles are clearly defined at systems design time.

Function allocation at loop, loop step or function level

Task decomposition is the common procedure in task analysis, which is performed for all systems in systems engineering, with the lowest-level tasks being called functions. Therefore functions allocation at the functions level is intrinsically tied to systems engineering. At the lower level of control loops, function allocation is less applicable, as humans cannot even work at a pace of the necessary loop iterations in order to adapt quickly enough. Here, the loop needs to work automatically, with humans / superordinate influencing it via control objectives. More recently, systems engineering more and more involved investigating and modeling the cognitive loops of the system operators (see cognitive task analysis). At this level of abstraction is now the freedom for systems designers and system operators to decide on function allocation, assigning tasks deliberately to humans or machines.

Sheridan: Humans and automation, 2002 Sheridan [She02] well describes four stages of a human-machine task and overlays chosen levels of automation. As a result, he illustrates how function allocation can be broken down to loop steps, see Fig. 5.10.



A Scale of Degrees of Automation

1.	The computer offers no assistance; the human must do it all.
2.	The computer suggests alternative ways to do the task.
3.	The computer selects one way to do the task and
4.	executes that suggestion if the human approves, or
5.	allows the human a restricted time to veto before automatic execution, or
6.	executes automatically, then necessarily informs the human, or
7.	executes automatically, then informs the human only if asked.
8.	The computer selects the method, executes the task, and ignores the human.

Figure 5.10: Fig. 2.6, 2.8, 2.9 by Sheridan [She02] arranged in way, show a matrix like definition of levels of automation along the loop steps.

Summary

Obviously, systems engineering better copes with the requirements on function allocation, as function allocation is a natural activity in systems engineering. The necessary prerequisite for this is task decomposition and planning not only the created system artifact (e.g. car, plane) itself, but also the operating procedures to be performed upon it by human roles known in advance. In some cases, where human roles are highly "standardized" by professional training/education (e.g. aircrafts, military systems, but also medical doctors to name a different application domain) skills and abilities can be well predicted at design time. With its additional efforts in human factors research, systems engineering has the necessary input for making design-time decisions on function allocation, at the level of cognitive loops of the users. The majority of all this design time knowledge is typically documented in documents in natural language targeted at system designers, human factors specialists, and system developers, but also involved potential users of the system to be developed. Organization of such documents is based on standards, an actual transition to computer-based modeling is only the result of recent efforts on SysML, that specifically adds systems engineering related features to a subset of UML.

5.4.2 R3.2 Automation levels and standards

When humans and machines collaborate, then there results a machine share of the tasks being performed, and a human share. Based on these shares, we can define levels of automation. They are an indication about the "automation depth", how much of the task has been taken over the machine.

Automation levels can be defined in two ways: 1) generic for all tasks, i.e. the same automation levels would apply for all cognitive tasks and they would have to be interpreted for the particular task or 2) task-dependent, so that for an individual composite task multiple automation levels are defined

General Levels of automation

Parasuraman/Sheridan: A Model for Types and Levels of Human Interaction with Automation, 2000 In "A model for types and levels of human interaction with automation" [PSW00] the authors define ten levels of automation, which are then applied in a scenario around air traffic control automation. Later, Sheridan in [She02] drops two of them and considers eight levels of automation between full execution by either humans or machines.

In both cases, the level of automation is associated with each of the four individual loop steps. We see an additional use in applying the same scale to whole loops to abstract from details. In such cases, the major difference is in the handling of the decision step.

NIST: Autonomy levels for unmanned systems (ALFUS), 2003 A long project (starting in 2003, on-going) at the National Institute of Standards and Technology (NIST) has long been investigating autonomy levels for unmanned systems. Due to the U.S. Army Future Combat System (FCS) focus of the initial ALFUS work, the mission and the environmental considerations initially emphasized military situations. In the most recent publication ALFUS, Volume II: Framework Models Version 1.0, Dec 2007 [HMAG07], however, they mention unmanned systems in the defense domain, manufacturing, urban search and rescue, border security and bomb disposal. During the course of their work, they have investigated a number of previously defined autonomy level metrics, mainly from systems engineering and military applications.

Interestingly in ALFUS (see Fig. 5.11), autonomy levels appear as triplet values along the axes:

- mission complexity
- environment complexity
- human independence (= level of autonomy)

which takes into account dynamic capabilities, e.g. in the evaluation of technology supplied by unmanned systems vendors. A high level of autonomy therefore always must relate to the variability of a certain task.

ALFUS also defines a scheme to aggregate these, for trading-off between individual skills. See App. C in [HMAG07] for details. For their application scenario, they have also created a Terminology document [(ed08].

Function-dependent Levels of Automation

Here, two examples are presented where levels of automation are determined and described verbally along the a particular system. In these cases, the system takes over more and more tasks, resulting in less necessary work for the human.

Autopilots in aircrafts The system "autopilot" in an aircraft is a device that takes over flight control functions from a pilot. However, the number of controlled parameters depends on the capabilities of the device or the level of automation, that a pilot chooses from a very capable auto-pilot.

Sorted from least capable to more and more capabilities, the following listing provides an overview of these capabilities:

- altitude hold
- altitude and heading hold
- altitude and course hold
- auto-throttle (=keep speed constant)
- approach a way-point (e.g. a radio beacon)
- set course (set waypoints), course is changed at waypoints
- coupling with instrument landing system (use an ILS guidance signal to control the trajectory)
- coupling with flight management system (set course, altitudes, waypoints, speeds). This mode does not include automation of lowering the gear and operating landing airbrakes, though.
- fully automated landings including landing roll on the landing runway (defined as Cat IIIc runways, not in operation so far, though)

Transmission in cars Depending on user requirements on comfort (ease of operation) and costs, as well as technical matters such as fuel economy, torque and power, cars are available with a range of transmissions, that convert the engine torque via one of multiple gear ratios to torque on the powered wheels. Over time, at least the following function allocation modes have been used in car transmissions:

- manual gearbox: the driver operates the clutch, and the shift lever all by himself to select an appropriate gear ratio
- hill-hold assist: variation of a manual gearbox, that eases starting to drive up-hill

	reference metrics summaries			**ALFUS Levels**
reference levels:	**MC**	**EC**	**HI**	**User-defined levels using metrics summaries to the left**
10	highest adaptation, decision space, team of teams collaborative missions; fully real-time planning; omniscient, highest level fidelity SA; human level performance	lowest solution/possibility ratio: lowest margin for error, understandability; highest level of dynamics, variation, risks, uncertainty, mechanical constraints*	performing on its own and approaching zero human interaction, negotiating with appropriate individuals	
9	high adaptation, decision space, team collaborative missions/tasks; high real-time planning; strategic level, high fidelity SA	low solution/possibility ratio, understandability highly dynamic, complex, adversarial high risks, uncertainty, constraints*	UMS informs humans; human provides strategic goals, interacting time between 6 % and 35 %;	
8				
7				
6	limited adaptation, decision space, vehicle tasking; limited real-time planning; tactical level, mid fidelity SA	mid solution/possibility ratio, understandability dynamic, simple mid risks, uncertainty, constraints*	human approves decisions, provides tactical goals, interacting time between 36 % and 65 %	
5				
4				
3	subsystem tasks/skills; internal, low fidelity SA	high solution/possibility ratio, understandability static, simple low risks, uncertainty, constraints*	human decides, provides waypoints, interacting time between 66 % & 95 %	
2				
1				
0	simplest, binary tasks	static, simple	remote control	

Figure 5.11: The Autonomy Levels for Unmanned Systems (ALFUS) group at NIST defines Levels of autonomy along three axes: Mission Complexity (MC), Environment Complexity (EC) and Human Independence (HI), see [HMAG07]. Their main setting are military systems.

- semi-automatic gearbox: the driver selects gears manually, but the clutch is operated by a machine system (automated clutch system)
- automatic with different available controls
  - program selection: the driver needs to select a shifting pattern (eco/sports) explicitly or the shifting pattern is guessed from the use of the accelerator pedal
  - the transmission offers an alternative mode, where the driver can quickly shift sequentially and therefore take over by shifting manually (Tiptronic, Speedtronic)
  - context recognition: the transmission control unit with sensors takes into account hills, slopes, or operation with a trailer, all to adjust the shifting pattern

In these cases, we see what a multitude of task-dependent automation there can result, or seen from a different point of view, how more and more tasks can be covered by automation, integrating the individual automation routines.

Summary

There is knowledge in systems engineering, that does indeed define levels of automation in a way that is generically applicable to all cognitive tasks, so it would be applicable to IT management automation as well. These schemes are mainly rooted in academic research, first standardization activities begin, when industry and standardization organizations see the need for doing so in actual applications such as unmanned systems.

Sheridan's eight levels of automation seem appropriate for EIMA in terms of their scale and nicely staged automation levels. Its verbal definition does not overcomplicate matters, and is enough for our purposes. The levels also well align with the MAPDEK loop model.

ALFUS also takes into account additional metrics that aim to quantify mission complexity and environment complexity, they could be used in automation feasibility assessments or comparing systems.

5.4.3 R3.3 "Good/best" automation for a task

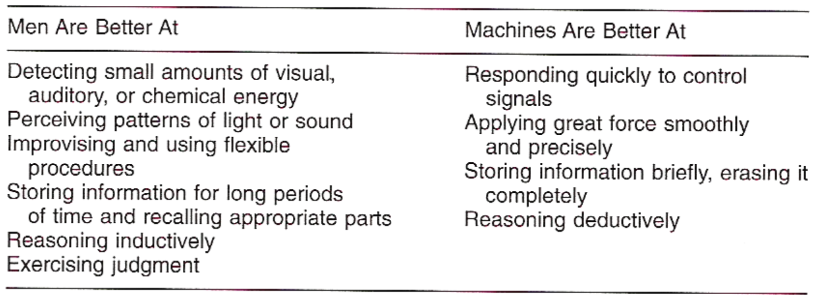
Advice on automation feasibility

Automation feasibility can be decided upon in a number of ways. So far splitting work between humans and machines in human factors engineering is mainly considered from the human side, not the side of technical capabilities. In other words: Human work is considered the default, and automation needs reasons, why it should be implemented. The economic motivation for automation is underrepresented in these considerations.

General academic work

There is a vast amount of literature on Human Factors Engineering / Cognitive Engineering, dealing with automation and its effect on humans. After listing a small subset academic work, we will take a closer look at practical examples, which even better show, that other domains handle automation in a way, so that it is used positively.

Fitt's MABA MABA list, 1951 In pioneering work, Fitt created a list of tasks, writing down men are better at (MABA), and where machines are better at (MABA), see Fig. 5.12. This list also includes physical tasks, not only cognitive ones. Given the period of over 50 years from its creation, machine capabilities have much advanced however, while the gist of the list still holds true.



Men Are Better At	Machines Are Better At
Detecting small amounts of visual, auditory, or chemical energy	Responding quickly to control signals
Perceiving patterns of light or sound	Applying great force smoothly and precisely
Improvising and using flexible procedures	Storing information briefly, erasing it completely
Storing information for long periods of time and recalling appropriate parts	Reasoning deductively
Reasoning inductively	
Exercising judgment	

Figure 5.12: Fitt's "Men are better at, Machines are better at" list (1951), as shown in [She02]

Sheridan, Various work Sheridan describes in (Human centered automation: oxymoron or common sense?) and (Function allocation: Algorithm, alchemy or apostasy?) that a scientific approach in task/function allocation has a hard time to replace features of an art.

Sheridan reasons in [She02] about Fitt's list, which he judges to be a role model in simplicity and understandability. Interestingly, he says that "as computers become smarter, they creep closer to bettering human capabilities. One can say for sure, based on lots of evidence, that human memory prefers large chunks with interconnected and associated elements, prefers relatively complete pictures and patterns, and tends to be less good at details. This empirical fact strongly suggests, that it is feasible and practical, the human should be left to deal with the "big picture" while the computer copes with the details.

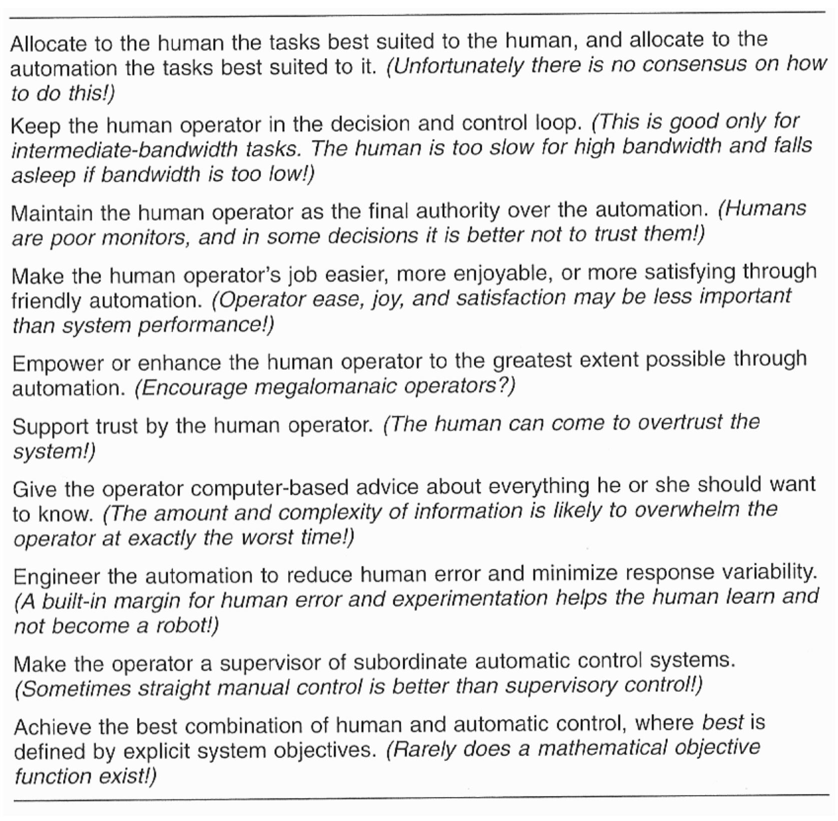
In a whole section on "How far to go with automation", he explains his point of view that all-or-none automation is a fallacy, gives advice on ultimate authority and limits on modeling, prediction and design in human-automation systems. His basic attitude stated again and again is the dependence of function allocation on a whole range of influences, and that there are no simple answers. We will likely have to live with the fact, that function allocation can hardly be a step, which itself is automated unless we can measure and analyze these influences and decide and act based on these analyses.

In Fig. 5.13, which is reproduced here for the comfort of the reader, he lists nominal criteria of human-centered automation and nicely wraps up the goals of function allocation in automation, but at the same time (in italics) lists the opposing influences.

Advances in human performance and cognitive engineering research This is an edited series of volumes, published by Elsevier, covering many aspects of automation. Volume 3 in this series once again tackles automation in airplane cockpits, seeming to be the role model of human factors engineers.

Practical examples: Airliners

The following practical examples show recommendations on automation in other domains. The described procedures are then each analyzed, what they would mean, if we applied them in a similar fashion to IT management automation.



Allocate to the human the tasks best suited to the human, and allocate to the automation the tasks best suited to it. *(Unfortunately there is no consensus on how to do this!)*

Keep the human operator in the decision and control loop. *(This is good only for intermediate-bandwidth tasks. The human is too slow for high bandwidth and falls asleep if bandwidth is too low!)*

Maintain the human operator as the final authority over the automation. *(Humans are poor monitors, and in some decisions it is better not to trust them!)*

Make the human operator's job easier, more enjoyable, or more satisfying through friendly automation. *(Operator ease, joy, and satisfaction may be less important than system performance!)*

Empower or enhance the human operator to the greatest extent possible through automation. *(Encourage megalomanaic operators?)*

Support trust by the human operator. *(The human can come to overtrust the system!)*

Give the operator computer-based advice about everything he or she should want to know. *(The amount and complexity of information is likely to overwhelm the operator at exactly the worst time!)*

Engineer the automation to reduce human error and minimize response variability. *(A built-in margin for human error and experimentation helps the human learn and not become a robot!)*

Make the operator a supervisor of subordinate automatic control systems. *(Sometimes straight manual control is better than supervisory control!)*

Achieve the best combination of human and automatic control, where *best* is defined by explicit system objectives. *(Rarely does a mathematical objective function exist!)*

Figure 5.13: Nominal criteria of human-centered design (and reasons to question them), see [She02]

Airbus Flight Operations Briefing Notes, 2004 In three very short (about 12 pages each) Standard Operating Procedures documents called “Operating Philosophy” [Air06], “Operations golden rules” [Air04a], and “Optimum use of automation” [Air04b], Airbus gives hints to pilots on automation related to the subsystems of an airbus: autopilot, autothrottle, flight management system.

What is key in their descriptions is, that automation in airplanes supports standard procedures of the crew. They put enormous emphasis on pilots understanding the automation systems, their functioning, as well as their non-functioning. They describe, how the automation systems can be coupled, that some of them (autopilot, autothrottle) are not to be overridden, and that a pilot should stepwise disable automation, take over and return to more direct control, when he notices that the automation does not work as expected.They give advice on how to choose the correct level of automation (based on a number of influencing parameters), factors affecting the optimal use of automation, recommendations for optimum use of automation, engaging/interfacing with and supervising automation. With a range of statistical observation, they argue that human error (non-regarding automation) is far more frequent than human error about automation.

CAA: Flight Crew Reliance on Automation, 2004 In [Woo04], the Civil Aviation Authority (CAA) stresses that crews are placing great reliance on the increasing automation in airplanes. They argue, that this turns into dependence and summarize their findings in the following two skeptic observations:

- “First, pilots lack the right type of knowledge to deal with control of the flight path using automation in normal and non-normal situations. This may be due to operators making an incorrect interpretation of existing requirements and/or a lack of emphasis within the current requirements to highlight the particular challenges of the use of automation for flight path control.

- Second, there appears to be a loop-hole in the introduction of the requirements for crew resource management (CRM) training. This has resulted in many of the training personnel and managers responsible for the ethos and content of training programs not fully understanding the significance of the cognitive aspects of human performance limitations."

However, they do not argue to reduce automation, but instead that 1) pilots are to be better trained on the automation systems and their properties, and 2) that "hand-flying skills" should regularly be trained, refreshed and used.

Practical examples: Industrial automation

Takeda, 2006 Takeda [Tak06a] analyzes manufacturing processes, most of them related to assembly of cars, machines, etc. as an application domain of automation. In his recommendations on low-cost automation, he focuses on a few simple steps done by assembly line workers and lists very simple and easy to implement changes of these typical tasks in assembly lines which shall bring down the error rate of human workers and speed up their work.

Obviously, the domain-specific descriptions of these procedures are not of much help in the context of this thesis, but the statement makes aware of an important fact: automation can be done in different ways: comprehensive, which mostly translates to complex and expensive, or focused on a few activities, which can enable quick wins. This is the main argument to implement multiple levels of automation in function allocation. Quick wins can even be realized by a few steps being automated, so iterative automation makes perfect sense. For automation to have the chance to be performed iteratively, a low level of automation should not be fixed with the shipped IT resource bundle, but be open to even further automation.

Setpoint, USA, 2009 Setpoint Inc., a small enterprise dealing with custom automation in industrial settings, has compiled a list of "Top 10 things you should know about custom automation", dealing with the main question for each of their customers: "Does automation make sense (for me)?" Clearly, they put many of the economic aspects first, but also list process issues. Their list is reproduced in Fig. 5.14 in total, so that their recommendations are immediately available to the reader. As can be seen, they mainly list negative aspects so as to warn their potential customers of the pitfalls of custom automation in industrial automation. While some of the items are specific to this domain, we can take over many of the recommendations to IT management automation. There, also defined processes, large groups of identical managed objects, and wise selection of automation tools and depth are key to overall automation success.

Summary

Once we give up the domain borders of *IT management* automation, the stream of available literature on better understanding humans, their strengths and weaknesses, and their interaction with automation, is sheerly endless. The typical application scenarios for cognitive engineers have been: air traffic personnel, pilots, soldiers. What is missing though is an in-depth analysis of IT administrators / network administrators, the staff most relevant in our domain. We believe, that most of the work would be transferable, as far as humans are concerned. Studies in this direction have started at IBM Human Research, but we believe there is much more to do.

10 Things You Should Know When Investigating Custom Automation
Through the years here at Setpoint we have seen automation companies come and go, and here are some of the things that we have learned as an industry leader. Anybody who is looking for automation should know the following 10 things - but if you don't, that's okay; we'll guide you through it.

1. **Automation can be a practical alternative to overseas labor.** Automating processes can decrease labor costs associated with producing products by combining multiple steps into one compact machine, making it cost effective to remain at home.
2. **Costs twice as much and takes twice as long as you think.** When automating processes, the inclination is to look at the big components like robots or electrical control systems, add up their costs, and then factor in some engineering hours. What is forgotten, however, are important items like light curtains, safety guarding, brackets, pneumatic valves, hoses, cables or rails to move the robot back and forth as well as the items that are required to make all components work together. Lead times on robots can be 16-18 weeks, with most lead times on major items averaging around 8 weeks. With the lead-time on components being so long, a project can take twice as long as you are anticipating it will.
3. **Cheap and fast, or good and reliable: which one do you want?** If you don't want to pay a lot and still get a Mercedes Benz product, chances are it will be broken and rusting away in a corner. If you want a fast solution, you're rushing through the design and assembly, risking the chance of excluding some critical safety or quality measures that you need. The good and reliable solution is like getting the Mercedes Benz at the proper cost and in a timeframe that allows for proper lead times and a good design to be built.
4. **Technologies are constantly changing.** Just because you bought something 5 years ago, and it works well, it doesn't mean that it's still the best technology on the market. Think about how many updates and upgrades there are with the typical computer; there is always some upgrade or new version that makes it function better. The same holds true for automation, products are improving and new technologies are constantly emerging.
5. **It's difficult for companies inexperienced in automation to articulate and visualize what is needed.** If you can't articulate what you want, you don't need it. For companies just starting down the automation path, there are a number of processes and capabilities that they do know, but hundreds more that they don't. A good automation company can help suggest better options to make your process run smoothly. Reveals flaws in processes and part consistency — inconsistent parts don't work well with automation.
6. **Processes and tolerances must be tightened up.** When automating a process, tolerances, dimensions, and part accuracy are critical. Processes must be consistent. If not, the automated process won't work and you will hate the machine.
7. **Just because you can build your parts with a hammer and an anvil, it doesn't mean the process can be automated.** Although we would love to be able to automate every process, sometimes the cost of the machine versus the payback makes it unwise.
8. **Chances are that your automation vendor will be out of business within 2 years.** Most automation companies are small businesses that design machines one at a time and then pass on all intellectual property to the customer along with the machine. When purchasing machines, 60-65% of the price is for materials needed to build the machine. If the machine is not paid for until the end of the project the automation company runs the risk of paying for the machine and the design of it until it is complete. This practice, along with taking on machines that are risky for the company because they may not specialize or have experience in certain areas, can put companies out of business. Be prepared to put money down on a custom automation project.
9. **Lean automation still means you need labor.** Automating processes does reduce your labor force, but does not eliminate it. Lean automation still needs someone to run the machine that has the ability to stop the production line if parts are being processed incorrectly or if an error is occurring.
10. **A full automation effort will require maintenance and spare parts to sustain reliability and a high uptime.** All machines need to have routine preventative maintenance performed on them. Spare parts need to be maintained to replace those parts that wear out. Sensors and vision systems need to be adjusted and tightened back into place to maintain the reliability of the machine.
11. **Automation will never call in sick, show up late, or create a sexual harassment lawsuit for your company.** This one just speaks for itself.

Figure 5.14: Recommendation list on custom industrial automation by Setpoint USA, see [USA09]

With respect to the EIMA requirements, we do indeed find much related advice automation (general or domain-specific), but the transfer to our domain is non-trivial.

5.4.4 R3.4 Dynamic function allocation at operation phase

Dynamic function allocation

As soon as automation levels come into play, it's a small step to not choosing one at design time, but instead having the human select from the levels at run-time (adjustable automation) or even making the machine adapt the automation mode (adaptive automation).

Both of them mean changing the trade-off between a low level of automation and a high one, once in a manual way, and once automatically. Again, Sheridan [She02] lists lots of related work on this topic and defines:

> "[Adaptive automation means], that the allocation of functions between human and automation is not fixed but varies with time depending on the human's momentary workload as well as current context. [,,,] A change in function allocation could also be at the request of the human (asking for help) or at the insistence of the automation (telling the human the job is not getting done, or simply taking-over when the human is napping or otherwise preoccupied."

Application: Adjustable autonomy in Space

Clearly, the need for adjustable automation arises first in those domains, where there are technical artifacts that are hard to control. Satellites and space objects fall in this category: on their course, they are far remote from their operators, and there is much uncertainty on the communication link in terms of bandwidth, delay, availability. Still, they need to do tasks and return results. For reasons of bandwidth, results should be information, not raw data, which motivates doing analysis aboard the space object. In a similar way, they must cope with high delays on the control channel, which renders remote real-time control impossible. Only goals/missions can be set on this channel. Even worse, a loss of the control channel may occur, so that local planning and decision is needed. All these are technical necessities.

We can assume, that after the end of the cold war, space agencies also had new requirements on higher efficiency and less operating staff due to budget reductions. So, there was an economic need as well.

Zetocha, 1998 In [Zet99], Paul Zetocha (Air Force Research Laboratory Space Vehicles Directorate) describes a feasible approach for implementing greater levels of satellite autonomy. He lists nine impediments to enhancing autonomy in satellites and answers them one by one. He pins his hopes on stepwise introduction of automation, cheaper COTS software, simulations for human verification and validation, quantitative analysis of automation results, and saving operational costs freeing resources for more capable technology. Limited cooperation between satellite staff and AI staff, as well as fears by workers are marked by him as largely political and psychological and as to be overcome

Kortenkamp, 2001 Kortenkamp [Kor01] describes the concept of adjustably autonomous control systems, that have the ability (1) to be completely in control, (2) supervise manual control, (3) be somewhere in between and (4) shift among these control extremes in a safe and efficient manner.

Regarding the design of such systems, he states the following guidelines:

- Strive for full autonomy where possible.
- Build in appropriate sensing, even for situations in which the human is in control
- Build in capabilities that enable multiple methods for humans and system to achieve goals
- Plan for changes in autonomy as much as possible
  - system-planned changes in autonomy
  - system-initiated changes in autonomy
  - human-initiated changes in autonomy

And summarizes that effective adjustable autonomy minimizes the *necessity* for human interactions but maximizes the *capability* for humans to interact at whatever level is most appropriate for any situation at any time. He argues, that adjustable autonomy must be designed in from the beginning. Often full autonomy is not possible (for technical, political or economic reasons), in such cases he sees adjustable autonomy as the only solution. Finally, he concludes that by asking the "right" questions (stated there, omitted here in the interest of brevity) at design time adjustable autonomy can be safe and practical.

Summary

Dynamic function allocation would take automation one step further. Instead of only implementing one level of automation, which is considered as the "best" by system designers, multiple levels of automation would be offered to the system administrators so that they can choose from them. Or, even more automated, so that the machine switches between them based on its sensors for human (non-)activity.

Two examples from space objects were shown, where adjustable autonomy is considered as a solution to make the operation of remote space objects more efficient.

There exist additional examples in cars, such an emergency auto-breaking system by Volvo called "Volvo city safety", that (if not generally disabled) under certain conditions automatically (based on front radar sensor input) first sounds a warning tone and then brakes the car to a stop, when it assists, that the driver (with his reaction time) would no more be able to brake. In contrast to other passive safety systems (such as airbags) and active safety systems (such as ESP) here the system takes over a function (deliberate full braking), that was previously allocated to the driver exclusively. The system deactivates itself in conditions such as parking.

5.4.5 R3.5 Computer-aided modeling of function allocation

Application to model function allocation and format for data exchange

Allocation of functions is known by human factors engineering professionals as encompassing both a process and a product. As a process, function allocation refers to the sequence of steps involved in establishing the alternate roles, responsibilities, and requirements for humans and machines in a complex human-machine system. As a product, function allocation refers to the end state of the application of the process, the optimal distribution of roles, responsibilities and tasks between humans and machines.

Verbal descriptions in documents The traditional and most versatile format to describe function allocation is natural language. It is the format for requirements documents, which are a main influence factor on function allocation. In critical domains with human operators and automation, such as nuclear power plants, chemical plants and the military, external requirements (policies, guidelines, standards) on function allocation have been described verbally.

Special, proprietary tools For verbal specifications, actually other tools than a word processor are not necessary. Yet, if authors want to improve consistency of the documents when establishing the alternate roles, responsibilities, and requirements for humans and machines in a complex human-machine systems, then tool support will base documents on one common model.

Large vendors with military contracts will likely use their own tools (such as document management systems), we could not identify any particular standard software for function allocation in particular.

As one concrete example, we found a software suite by Carlow Inc., a small company that created special HFE software for handling systems related to the US Navy and Air Force.

In [MH03] they describe function allocation as a matter of policy, practice, procedures and process. This publication in naval engineering well shows the new look at function allocation aiming to reduce crew size on ships. The authors look at function allocation in a way to reduce manning on Navy ships.

There, when the system development objective is to downsize emerging systems as compared with existing systems, the focus of the allocation of function effort changes from an emphasis on optimizing human roles to minimizing human involvement in system functions. In addressing the issue of performing system functions with fewer humans as compared with existing systems, the function allocation strategy is not simply to assign functions to automated or manual performance on the basis of differential capabilities and capacities of the two, as exemplified in the Fitt's List approach. Rather, the strategy is to automate functions to the extent needed to enable the required reduction in workload and manning, with attendant provisions for decision aiding, task simplification, and design in conformity with human engineering standards to ensure adequate levels of human performance and personnel safety. Another change in emphasis when allocating functions for a reduced manning environment humans and machines are not viewed as competing resources to which responsibilities are assigned on the basis of their unique and individual capabilities but rather as cooperative elements of a system interacting and collaborating in synergy to achieve the system objectives.

SysML 1.0, 2007 The increase in systems complexity is demanding more formalized systems engineering practices. In response to this demand, along with advancements in computer technology, the practice of systems engineering is undergoing a fundamental transition from a document-based approach to a model-based approach.

There, a standardized modeling language is considered an enabler for model-based systems engineering. For this purpose, the Systems Modeling Language (SysML) was developed as an extension of UML 2, with version 1.0 released in September 2007.

In SysML,"allocation" means relating certain modeled items (e.g. requirements, behavior/functions, flows, structure, properties) to different modeled items in the system model. "Functional allocation" then deals with relating activities, or functions, or actions to blocks or parts. Blocks or parts are typically hardware and software components. As a result, an allocation table shows which function is allocated to which modeled hardware and software component of the *machine* system.

See Fig. 5.1 on page 132 for an example of a function allocation matrix.

SysML enjoys support from many industrial sectors traditionally associated with the systems engineering branch: car industry, aerospace manufacturers, military applications, tool vendors (e.g. Artisan, IBM, Telelogic). Often former UML tools were equipped with SysML extensions. Due to SysML being a recent language, there are basic modeling tools and drawing applications for SysML, but what lacks are e.g. applications that use SysML function allocation to really work with it, create animations, statistics, visualizations, etc.

Summary

SysML is the promising format to standardize verbal descriptions in function allocation and a transition from a document-based approach to modeling-based systems engineering.

5.4.6 Summary

Systems engineering can offer lots of experience, recommendations, and past investigations of human-machine-systems. Even though the typical application scenarios there involve aircraft pilots/air traffic controllers, space applications, operators of nuclear power plants, many of these share properties with IT administrators and the mentioned systems have similarities with data centers.

Function allocation being done at design time is considered standard in systems engineering, described in a written manner in documents. We have identified levels of automation that also would make sense in our domain. The existing advice on good automation presents different approaches to automation: plain and simple starting small in some applications (low-cost automation), in other cases applying sophisticated methods with proven engineering carefulness and simulations. Dynamic function allocation and the concept of “adjustable autonomy” go hand in hand with a desire for switching between levels of automation to accommodate more situations. Regarding tools and data formats, SysML presents the state of the art in the migration from a document-based approach to systems engineering to a modeling-based one.

5.5 Proposal to deal with function allocation in EIMA

The following recommendations for function allocation in EIMA were derived from the state of the art in IT management automation with the input from systems engineering. They aim to make available the essentials of the experience in systems engineering in the domain of IT management automation.

5.5.1 R3.1 Static binary function allocation concepts

Explicit tagging of roles, whether they are allocated to humans or machines

We should use names for roles in task and loop analysis, that first deliberately leave open, whether a human or machine will later play this role. This allows automation in places it had not been predicted before. For such roles, we can use no prefix, such as in “Planner”, or the prefix “u”, such as in “uPlanner”.

During function allocation, a systems designer in EIMA is supposed to make the essential decision, whether the function is assigned to a machine role or a human role. The roles should be marked with a prefix “h” for a human, and “m” for a machine component. Then, role names such as “hProjectManager” and “mFocusOptimizer” do not leave any doubt about the categorization.

	Machine	Machine and Human	Human
T01 ...	X		
T02 ...	X		
T03 ...		X	
T04 ...			X
T05 ...		X	
T06 ...	X		
T07 ...	X		

Figure 5.15: Allocation matrix, that covers the most essential automation decision.

Scheme for human/machine function allocation and accountability for functions explicitly

In EIMA, accountability for all tasks must reside with humans. This is made explicit with a RACI chart, see e.g. the example in Fig. 5.16. This way, we have a person, that cares for monitoring and resolving automation failures, checking compatibility, and making improvements, all of which a machine role could not do by itself.

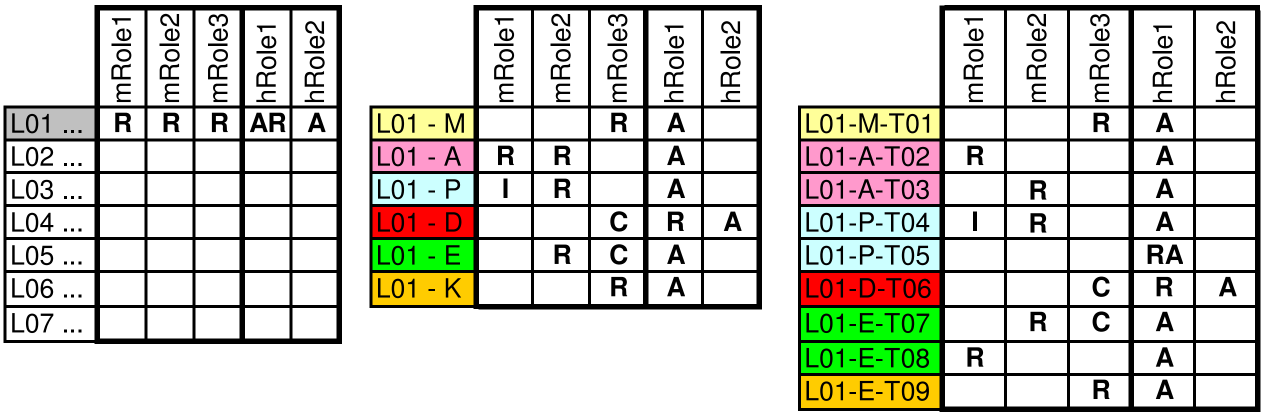
	mRole1	mRole2	mRole3	mRole4	mRole5	mRole6	hRole1	hRole2	hRole3	hRole4
T01 ...	R						A		I	
T02 ...		R						A	I	
T03 ...			R				A		I	
T04 ...		C						R	A	
T05 ...			C		R			A		
T06 ...			C			R		I	A	
T07 ...		C				R		I		A

Figure 5.16: Refined allocation matrix, that covers the responsibility across human and machine roles. (R)esponsible=does the work, (A)ccountable = ultimately accountable for the correct and thorough completion of the deliverable or task, exclusively assigned to humans, (C)onsulted = included with two-way communication, (I)nformed = included with one-way communication.

Function allocation at loop, loop step or function level

Due to the structure of tasks in EIMA, we have the chance to do function allocation at different levels of abstraction: essentially we have the AutoITIL process level (unlikely to be concrete enough for real decisions, but a good start for tendencies), then the level of loops (each loop as a whole), then single MAPDEK loop steps and finally the functions within the loop steps. This variable level of abstraction allows to cope even with large task trees. Tendencies can be agreed upon soon in systems design and then details can be worked out once the task decomposition tree has been defined in more detail.

Regarding documentation of function allocation at the various levels, the solution is simple: In a document-based approach, we can create allocation matrices for those levels: roles on the one axis, and loops/loop steps/functions respectively on the other axis, see Fig. 5.17 for an example.



	mRole1	mRole2	mRole3	hRole1	hRole2
L01 ...	R	R	R	AR	A
L02 ...					
L03 ...					
L04 ...					
L05 ...					
L06 ...					
L07 ...					

	mRole1	mRole2	mRole3	hRole1	hRole2
L01 - M			R	A	
L01 - A	R	R		A	
L01 - P	I	R		A	
L01 - D			C	R	A
L01 - E		R	C	A	
L01 - K			R	A	

	mRole1	mRole2	mRole3	hRole1	hRole2
L01-M-T01			R	A	
L01-A-T02	R			A	
L01-A-T03		R		A	
L01-P-T04	I	R		A	
L01-P-T05				RA	
L01-D-T06			C	R	A
L01-E-T07		R	C	A	
L01-E-T08	R			A	
L01-E-T09			R	A	

Figure 5.17: Function allocation example at loop level, loop step level, function level. The first matrix shows the list of loops, the second one shows the loop steps of loop 01, the third one details the loop steps with functions.

In a model-based approach, we can model all of them. If the tool was scriptable, it could automatically ensure consistency along the task hierarchy. So one could start at function level and it would propagate up to loop steps and loops. Or one could start at loop-level and it would propagate down into the loop steps and functions. SysML tools could allow to have the task hierarchy serve as a grouping operator in the task axis, so that even one matrix would suffice. It is especially nice that such aggregation views clearly improve comprehensibility. We could also add an AutoITIL process level on top of the loops in a similar fashion.

5.5.2 R3.2 Automation levels and standards

There are two ways of looking at automation levels: they can either be defined in a generic manner or they can be defined depending on the individual loop/task.

Generic automation levels Generic automation levels have the benefit, that they are applicable for all loops. So we can quickly and without going into too much detail attach automation levels to loops, e.g. based on their loop names. This assignment can then be a requirement, or a wish, or a likely estimation by an automation engineer on loop automation feasibility.

EIMA proposes to use the revised scheme by Sheridan (see Fig, 5.10 on page 160) for reasons of its simplicity, nice range, and background of the author. In these levels, the major difference is in the Decide step, with small influence on Execute (sending messages) and Knowledge (logging for later reference).

Still, the difference in decision making is essential to the level of automation. That's why the "Decide" phase was included in the MAPDEK loop. Decision making is choosing one option (or no option) from a set of options according to some rating function. Basically, a decision can be broken down to the following sub-actions:

- build the set of potential options (level 2)
- build the rating function
- adjust the rating function
- apply the rating function to all options
- choose among alternatives with the same rating (level 3)
- execute the chosen alternative (level 4, 5)
- tell about it (level 6,7)

The extremes (level 1 = human and level 8 = machine) have been treated before.

Sheridan defined his scheme in a way to assign automation levels at the loop step level. By default however, we believe that assigning automation levels to whole loops will be far more frequent at EIMA's level of abstraction. However, refinement of automation levels to loop steps is also possible in EIMA.

The drawback of generic automation levels is, that the details for the task are unknown. E.g. "allows the human a restricted time to veto before automation execution" does not say what is executed in detail.

Task-dependent automation levels For this reason, task dependent automation levels along the sub-tasks (such as "search for updates, but do neither download nor install", "search for updates and download, but do not install", "search for updates, download, and install" in Windows Update Management) make sense. Obviously however, we cannot expand task dependent automation levels here for all tasks. The obvious property of task-dependent automation levels is that the semantic variety (different meanings of different automation levels) comes along with a syntactic variety (different names of the levels of automation). This can be seen as an advantage to not compare automation levels for different loops, but also as a disadvantage as all loops actually must be known to assess the level of automation from the name of the current automation level.

Application advice: Describe the automation level in a generic and a task-dependent way! A common misunderstanding in automation is on the names of levels of automation. Clearly, "half-automated" does not explain much. It is important that users of the automation can map a generic level of automation, such as Sheridan's level 4 "the computer selects one way to do the task and executes that suggestion if the human approves", to a task-dependent automation level, e.g. "The computer pre-selects the software packages to install according to dependency information from currently installed packages and information about new versions from the package repository. The user can override this pre-selection but the machine will keep the updates and their dependencies consistent. On approval by the user, the selected package updates are installed."

In system specifications, both descriptions, a generic one and a task dependent one should be kept. The generic one helps to quickly assess the overall automation level and therefore the remaining automation potential (in fact, Sheridan's level 6, 7, 8 do not leave much room for further automation benefit), so that this task can be excluded or at least handled with lower priority in the next automation effort.

Via the user interface, the task-dependent information should at least be available on request. Clearly, in central automation manager components, (such as an automation control center for a storage appliance), automation levels cannot be described in task-dependent verbosity. Here, after training of the administrators, the generic levels may be used (a simple integer in Sheridans scheme) with the long task-dependent description being available on demand as described above.

Idea: Creation of automation levels from behavioral modes inspired by current systems

As a compromise between generic levels and task dependent levels, common “behavioral modes” and examples have been sought after and identified, that at least can give the reader a good overview on automation levels.

The behavioral modes are divided into control modes, built-in functions and a table of behavioral modes. They all cover different aspects of a loop’s level of automation.

Control modes We can assume, that management automation systems will have an externally controllable power supply, so that managers express their wish to switch an autonomic system into certain states:

- powerless: Without external power, the system cannot function, it is off because of lack of power. This can be useful (pull the power plug to reset devices without built-in sources of energy) or it can be unwanted (device reset, here prevent this by using an uninterruptible power supply (UPS). In every case, the automation engineer needs to reason about this case and that it does no harm to the system. Administrators resetting systems by power-cycling them, even though this is not recommended, are far too common.
- off: In this state, the loop is powered, but is switched completely off intentionally. This can be useful for temporary complexity reduction in fault analysis (e.g. disable DHCP when troubleshooting network problems). Of course, there must be some way to turn it on again.
- on: In this state, the system in up and running. It should be decided, whether it can be switched off temporarily or permanently. It may also switch off/on after some event, such as a timer event or an interrupt. This also relates to human take-over, e.g. cruise control being suspended in a car, when the driver uses accelerator or break, or system-takeover, e.g. an automatic braking system.

As a side note: Truly autonomous systems would have an internal power source, or would be powered reliably from the environment. Due to the internal trigger, there would be no control from the outside, the system could not be managed. It could only be influenced via the environment, if one understands its reactions. A wild animal in a forest or the sea is a good example of an autonomous system, in IT management automation we can hardly imagine that true autonomy would be of any use for us for the reason of lack of control.

Application advice on the control modes There should be a defined interface to manage a management automation system, and also its loops, at least in an ON/OFF manner.

The default state should be decided upon carefully.

For loop-like automation, it should be decided whether a new iteration of the loop can be triggered externally, such as e.g. the automatic garbage collection process in Java. This makes only sense however for loops with a lower activation frequency than is welcome by the specific application. Finally, “standby” modes with a reduced function set, where some component is waiting for an activation event, can be handy e.g. for saving power or deliberately using the device with less functions.

EIMA engineering needs to take into account events such as power outages and broken communication links. For each of the MAPDEK loops broken communication links to the outside entities (manager and resources) should be taken into account and handled internally, mainly in the Monitor and Execute components, or in a separate Touchpoint entity.

Power specifications for management automation systems created with EIMA needs to include a specification or advice on appropriate reliability of power supply to the automation software. Automation software, that internally handles power outages gracefully and may even work in a stateless manner, will have different requirements on power supply than stateful automation software, that keeps large amounts of data in RAM. As machine capability supporting this, there are databases based on Atomicity, Consistency, Isolation, Durability (ACID) transaction handling, that will at least remain in a consistent state after power outages. The commonly accepted trade-off of the various locking and timestamp mechanisms enabling transactions results in a lower execution speed, however.

Power cycling a management automation system (especially, when it is unresponsive or not functioning properly) should ideally not harm the system and bring it into a defined working state after restart. In fact, remotely switchable multiple socket outlets are one of the most basic remote control facilities in data centers. They may not be elegant (as power-cycling a system prevents incident analysis and getting the necessary information about the error), but when remote terminal (e.g. ssh, serial connection) access to a component no longer works, there is little else an administrator can do than power cycling the device/software component anyway.

Access to remote control on power supply must be well protected from abuse and unintended human error.

The control modes named above are only a rough classification. A state machine with state transitions and a list of available functions in each state will be needed for more complex systems. Of course, the necessary management commands to change between these basic states must be known to the administrators.

Administrators should be trained on the control modes. They should accept, that managing control modes of the management automation system is a primary control interface instead of immediately switching to low-level direct control.

With respect to quickly recognizing automation potential:

- Review all available automation modes and compare with EIMA's proposed levels of automation! Decide whether higher automation levels can be added!
- Review the default automation on/off mode!
- Review if previous low automation was caused by insufficient access control!

Example When adding a management automation system to an upgrade mechanism, it should at least be switchable on and off. Along Sheridans automation levels, it should decided upon user notifications. The most automated mode of "silent updates" will only be welcome by users, if it works properly and also handles corner-cases like compatibility with other software and is performed with no negative effect on the usage function of the system. It needs to be decided on the default state of the system, that is preconfigured when the user has never touched the configuration. When deciding for a trigger of the update loop, it should be possible to deliberately start an update cycle. Communication to a central package repository should be configurable and gracefully handle non-availability, but at least with a log entry or user notification. The update should be performed in a transaction-like manner to work around problems of power outage during the update. Power cycling or access to remote control on the power supply does not apply to this kind of system. Administrators need to know, what the current control mode is (on/off), and they should know how to switch it. If faults in update automation are considered likely, there needs to be a fall-back procedure, so that administrators can update the system manually and reset the update automation system.

Loop-local functions The following non-exhaustive list of loop-local organizational functions can be added to loops in addition to the MAPDEK steps:

Access control to the loop trigger: If triggering a loop is an event that needs to be guarded from unauthorized access, than access control mechanisms need to be applied there. They would typically be added to the management interface of the loop or the Monitor step.

Self-check mode: A loop could compute a checksum on itself to know, that it is unaltered, just like software packages do on the files before they are installed. This would give a certain protection against unwanted mutation/change in systems.

Logging mode: Logging is the basis for better fault diagnosis. It needs to be decided on log level (filtering), log space/time and entry overwrite. As an example, flight recorders are best known for their value in the analysis of flight incidents and accidents.

Explain mode: Documentation and explanations are a key acceptance factor in the phase of switching to systems with significant management automation routines. By tracing output, reporting output, and explanations the system can describe and justify decisions and give reasons, as well as update the admin on progress. Simple examples are database query optimizers: Postgres explain, DB2 visual explain, both of which explain, how they dissect and translate a complex query into atomic actions. Advanced explanations would log all reasons and allow to give reasons for past, and current, potentially also future decisions (when being asked for it) understandable by humans. This is different from logging due to its level of human comprehensibility.

Application advice on loop-local functions Many workflows are cluttered with items like internal non-functional activities like access control, logging and fault handling or other organizational built-in functions. To not make this happen to MAPDEK loops, such functions should be externalized and added to the loops, so that they can be excluded/included in a certain view on request.

All four mentioned internal functions should be considered on all loops and it should be decided on a case-by-case basis whether their use is beneficial. In a class representation for loops, these functions can be added to the Loop class, and be inherited and implemented by each loop.

Access control will be the necessary prerequisite to building secure management automation systems. Self-check mode can be seen more as an optional function, which protects against modifications. It will be used for critical systems.

As non-properly functioning automation is a likely cause for *repeated* faults, there is a high risk that automation will be disabled at the first sight of potential automation flaws. Logging is important to provide in such events the necessary information to trace errors to their most likely cause. For this reason, logging should never by completely disabled. The typically negative performance impact of logging should instead be traded off by choosing a different loglevel or different maximum log-size, both via the management interface. Logging output will be read mostly by machines who process the information and make it accessible for humans. However, a log processor should not be *necessary* to read logs. For this reason, logging output should have a formal structure, but a human-readable explanation. The syslog-ng protocol for distributed logging, and eventing schemes like Common Base Event (CBE) are standard forms for logging.

Explain mode should be added to all loops, where complex decisions are made in an automated manner. In contrast to logging mode, it should be designed specifically for humans, not machines.

With respect to quickly recognizing automation potential:

- Review whether the four functions are present in loops!
- In case they are not present, review whether they are required to improve automation!
- In case they are present, review whether they are implemented in an appropriate fashion!

Example In the same scenario of an update management automation system, access control to the decisions during the update (if needed at all) are necessary to ensure the right qualification of users at this point. Self-check of the update automation system is optional, it should be decided upon whether the update management automation system can update itself as well. Of course, all update actions should be logged to a known place. Explain mode could include information on versions, changelogs, fixed errors, added or removed features, dependencies between software packages. Resource consumption in terms of disk space should be checked before performing an update.

Table of common behavioral modes In Tab. 5.1 you will find a list of behavioral modes that all represent half-automated modes. Several sample applications are mentioned taken from many fields, where management automation routines have matured.

Recognizing automation potential with behavioral modes With respect to quickly recognizing automation potential, there are a number of "natural" transitions in the behavioral modes, where change from one mode to the other increases automation, but as a small, easily implementable step. They may be preferred for their non-radical nature balancing automation benefit with staying in control and accepting the automation improvement:

- From "confirm" to higher Sheridan levels: User confirmations have positive and negative effects. If a user/administrator is indeed knowledgeable about the action to be confirmed and the decision depends on certain influencing parameters, confirmations are a welcome interaction. This changes, when the user does not know how to decide, or if the reaction to confirmation requests stays the same. As waiting for user input in confirmations can substantially reduce the execution speed of automation, it needs to be used carefully and reasonably. Ideally, confirmations are configurable. A multi-level undo/redo feature can often replace user confirmations as the user can easily go back in case certain of his prior actions should be reverted. This also gives the user to preview the effects.
- From "alerting" to "predefined error handlers": Reacting to alerts that had been activated the threshold triggers is a very common action for system administrators. The variety in complexity of the necessary cognitive tasks is high: some alerts require only very simple reactions, such as dealing with a user using more network traffic than appropriate or dealing with a likely infected host in a large network, others are more complex. In many scenarios, the majority of these alerting events can be handled with a small set of automatically performed rules. This only needs system administrators together with their management to standardize reactions in usage policies and to automate the reactions. Blacklists/white lists can provide the necessary exceptions.
- From "mapping" to "explorative": mapping an environment, that is large and complex presents a high effort. If a manager could express the rules that he applies implicitly during mapping, than the system can explore the environment on its own but also pay attention to the rules. Automation

Table 5.1: Behavioral modes of automation in management automation systems – 1/3

Name	Behavior	Examples
direct control mode	Manager turns off the management automation system and works around it. Human take-over.	Usually, by managing at the lowest layer, the manager can be sure, that the management action won't be overridden by the management automation system. Example: In case of suspected fault in a router management automation software, such as Ciscoworks, router administrators will fall back to configuring the Cisco routers via a lower-level interface such as via IOS being remotely used via a serial connection to the router or a ssh connection.
simulation mode, test mode	System repeats every management command, shows internal settings, does not actually perform actions, shows plan how high level goals would be implemented with actions.	A backup plan could allow an administrator a before/after comparison, so that the admin can judge whether the backup plans are ok and data will be distributed in accordance with requirements. CD burning test mode allowed to simulate burning a CD testing source input data rate and buffer sizes, but not actually executing the actual disc writing.
mapping	Manager controls the system, system logs all sensor data and therefore maps the environment it passes through.	Guided/controlled network or service discovery can profit from such a mode, if an admin wishes to explore the resources in a controlled manner, so that the resource discovery does not interfere more than necessary with actual production traffic. This mode is also known from human-controlled reconnaissance vehicles/scanners, and could be taken over to other scanning activities, that need to be done in a controlled way in an unknown environment. Network monitoring/documentation, service monitoring/documentation would profit from such a mode, especially when the environment is very dynamic such as wireless network services.
explorative	Manager does not control the system, system "crawls" through environment based on its sensor input.	Automated discovery systems for networks or services use this mode if an administrator does not wish to control the progress of the exploration.
active probing	System expands its own knowledge by actively making probing measurements, e.g. to test its assumptions or to investigate error causes.	Instead of polling all potential causes for errors in a monitored system, a monitoring system can actively (out of sync of usual polling or expected message reception) probe individual resources to better assess the true effects of resource outages. Reducing the polling frequency (and therefore monitoring traffic) and adding active probing in cases where up-to-date information is instantly needed for a small portion of the infrastructure is a promising benefit of active probing.

Table 5.2: Behavioral modes of automation in management automation systems – 2/3

Name	Behavior	Examples
confirm	The system suggests an action, has the user confirm it.	Example: `rm -i`. Confirmations are the most basic delegations of a decision to an upper entity. They are used in many small decisions in current IT management automation systems already. Eventually they can be replaced with an automated decision paired with an undo capability.
alerting	System monitors input, sends notifications when certain conditions apply.	Alarm systems, watchdog systems, notifier systems in general all use this mode. In cars, reverse alarm obstacle sensor are an example, in airplanes the Traffic Alert and Collision Avoidance System. In IT management it can be found in terms of broken threshold notifiers, SNMP traps, simple self-made scripts monitoring certain objects, update available notifiers and many more places.
predefined action handlers	System monitors input, starts handler routines on known events, notifies manager of unhandled events.	Watchdogs with fix handlers, SMART in hard disks, most current IT management systems, procmail (predefined rules and actions) are examples for this mode. It is much advanced over the alerting modes, as here the reaction is also machine-executable in terms of handler routines.
guided	System is given a goal state, system computes optimal route from current state to goal state, during approaching this goal system monitors progress, gives advice for future user behavior to reach the goal.	Today's navigational systems in cars. This mode could likewise be used in IT management for migration planning of resources/services, or service recovery.
waypoint autonomy	System performs actions automatically towards a certain goal given by the manager, but continually listens to changes to the objective criteria. Here, the human is out of the loop, but remains in control.	Adaptive cruise control in cars, or waypoint following algorithms in autopilots are examples for this mode. In IT management automation, this mode can be used for fix goals that need to be reached on a path needing temporary overriding. Examples include migration of composed services or service recovery.
training, user feedback	System monitors sensor input and notifies user of issues it does not have any rules for yet, learns (builds rules) from user reactions.	This mode is known from e.g. human error correction mode in OCR programs handling unclear cases after automated OCR. It would well be applicable in IT management in detecting and handling unwanted entries in configuration files, detecting and handling unwanted change, mass operations on files/users/quota, and more application scenarios.

Table 5.3: Behavioral modes of automation in management automation systems – 3/3

Name	Behavior	Examples
immune	System applies fine-tuning to user input and adapts to environmental disturbances. The human can rely on system to take them into account.	Prominent inspirational examples are flight control systems, stabilizers in ships, anti-slip control in cars. In IT management automation, we could think of adding immunity to applications, services, resources so that outside bad influences have less effect on their desired functioning. Other negative influences include congestions and delays on links, or availability of local resources like CPU time and memory.
corrective	System ignores user actions that are out of safety range or actively triggers actions, system take-over.	In the vehicle domain, auto-braking such as with Volvo CitySafety, and electronic stability control are corrective assistant systems. In IT management automation, this could be used for reducing administrator errors, when they do critical actions such as deleting files, stopping services, shutting down machines
follow the master	Manager controls a master device, several slave devices take over the management actions from the master.	In the vehicle domain, electronic drawbars were invented to virtually assemble road trains out of trucks. Likewise, agricultural machines like tractors can be operated in a “follow the master” fashion to process large fields with many machines simultaneously but with less operating staff. It will soon be used for formation flights of unmanned aerial vehicles. This mode can be used in IT management automation for group operations, e.g. treating many VMs in parallel via a multicast terminal session, or for simultaneous changes in a resource environment, such as changing VLAN IDs in a switched network.

benefit increases with size of the unknown environment and frequency of re-exploration. Only if re-exploration is automated it will be done periodically, and result in an updated view of the environment.

Another way of using the table of behavioral modes is, that the reader is inspired by the list and will guess for himself, whether certain of these behavioral modes are useful in a certain automation scenario. Different modes can be combined or chosen alternatively and then evaluated (e.g. in a simulation) whether they meet the individual requirements or not. The list of automation behavior modes serves as this source of inspiration.

Example Again, imagine an update management automation system for a resource pool. Confirmations (or more general: user input) may be considered necessary for the following decisions:

- deciding on which update repository servers to connect to
- deciding on the polling frequency to the update repository servers
- deciding on whether to allow access to these update servers at all
- deciding on which software packages to update and on which resources
- deciding on when to download the new packages
- deciding on when to install the updates

Clearly, there is an automation potential in both, dealing with the decisions. All confirmations could essentially be replaced with preconfiguring the answers for all cases, but this is often impractical. Therefore, we can also apply the following more automated reactions:

- probing a list of update repository servers, or asking a broker to forward the request to an appropriate one
- using an adaptive polling frequency based on the time since the last update
- this is policy issue, the typical reply however can be learnt from user input
- an update mechanism can pre-estimate the likely user selection based on metadata, such as changelogs, recommended/optional update
- an adaptive scheme can download small packages instantly and only have large downloads confirmed, the threshold can be estimated by download time on the bandwidth of the link
- a mechanism can either refer to a configured maintenance window or assess the activitiy of the updated services itself. It can then use periods of low typical service usage to update a certain application providing a certain service.

Alerts may be triggered by the following events. According to the recommendation above, we would need case-specific error handlers in order to mask many of them. The automated predefined error handlers themselves are scenario-specific and therefore not covered above (or by EIMA), but it is easy to come up with ideas for them:

- non-available update repository servers, communications errors, such as broken connections : timeouts, retry, resume of partial downloads
- certificate errors: excluding such servers from the list, only alert, when the list of update servers is empty

- a certain download volume of update data or share of bandwidth being exceeded: built-in download limits and traffic shaping capabilities
- non-validating digital signatures of update packages: fetch package from a different server
- updates failing on certain or all resources: retry a defined number of times
- non-available local disk space: free temporary own resources, that are no longer needed, e.g. old downloaded update packages, old temporary data such as snapshots. If this is not enough, invoke a central disk space manager, as lack of disk space will cause many more management mechanisms to fail.

Using the table as source of inspiration, the following mode can be added to the update management automation system:

- "follow the master": especially useful for long update management wizards with many decisions to make, this can then be turned into unattended updates for the rest of the resources, once a certain master resource has been updated
- "guided" navigation in a complex service upgrade: In a complex service upgrade, the state before is known, and the state to be achieved is known as well. What is missing, is a path from the current to to the desired state that is optimal to some utility function, such as service outage time or temporary resource consumption. A service update navigation system can pre-compute paths and show the administrator the available routes and their implications. In choosing one path (an update plan), an administrator can include many additional considerations, that cannot all be configured in a system based on his experience.

5.5.3 R3.3 "Good/best" automation for a task

EIMA should give advice on automation feasibility!

Automation feasibility input is a task and a set of machine capabilities, with the aim to find an appropriate machine capability for a given task. Then, automation feasibility assessment needs two steps: (1) matching a task (a process, loop, loop step, function) with a candidate set of machine capabilities, that are likely to solve the task and (2) an in-depth evaluation whether this task can really be done by this method based on many criteria. Given the universal, generic approach of EIMA, we will care about (1) only, as (2) is too dependent on the scenario.

Matching tasks with candidates of machine capabilities In EIMA, the workflow input is prepared for automation feasibility assessment by the steps mentioned before to expose the automation relevant properties of the task: (1) create task decomposition tree, role hierarchy, automation knowledge table, (2) create MAPDEK loops.

Now, the default automation guidelines based on experience and use of methods today are:

- Monitoring tasks: VERY GOOD automation feasibility. In IT management, monitoring is almost always automated, in contrast to other domains, that deal with physical objects in the real world where the lack of sensors is a major obstacle for automation. In IT management, raw data is already digital, there are no analog to digital conversion losses. As long as a monitoring interface exists and is documented, it can be used.
- Analysis tasks: GOOD automation feasibility. There are many available routines, even though IT management only slowly recognizes their potential. What helps us is capable, available machine

power, e.g. in clusters, grids, clouds, that could well be used also for analysis in IT management automation, instead of other sciences which already make use of this source. In IT management, not the classical algorithms from other domains are important, such as 2D/3D image analysis, speech recognition, motion analysis, but instead analysis tasks often cover combinatorial problems and reasoning.

- Planning tasks: MEDIUM automation feasibility. Planning in IT management (and in general) quickly suffers from state space explosion for real-world problems. So, the problem must be small and limited to few actions to try. So far, automated planning is hardly known in IT management (in contrast to other domains such as robots), except for very specific purposes, such as archival planning in archival solutions.
- Decision tasks: LOW automation feasibility. Decisions are simple to automate, if they are based on a few objective, quantitative criteria. However, as decisions almost always also contain strategic and tactical interests, as well as mainly undocumented policy matters, and there is a very low acceptance for wrong decisions, decisions are the tasks that most likely stay with humans.
- Execution tasks: VERY GOOD automation feasibility. The complexity of execution tasks in IT management is generally low. However, we need a control/management interface to send it to a certain resource. Such management interfaces are available for most data-center grade hardware, but they may cost an extra amount. We have few "real-world" problems as there are no digital to analog conversion losses, and we can rely on mature communication links.
- Knowledge tasks: VERY LOW automation feasibility. Knowledge handling, modeling and processing is among the most difficult tasks to automate, so that a human typically beats a machine by far by his means of abstraction, generalization, intuition, inference and comprehension. Ontologies and machine learning try to get a grip on better automation around knowledge tasks, but this is only a start. If available methods do match the problem, they do apply, but often lack properties like robustness, performance and flexibility for more demanding tasks.

With this general rating, machine capabilities can be looked up in the catalog of machine capabilities (see Ch. 6) to know a good portion of the candidates on machine capabilities. The output of EIMA function allocation are better knowledge, on what tasks to assign to machines, what tasks to assign to humans, or how to combine both of them. When doing function allocation, also be inspired about the behavioral modes of automation mentioned above. The human-performed mapping has the benefit, that a human can better assess new methods that in turn can quickly be associated with a certain designed entity in the automation system.

Not in EIMA: Scenario-specific in-depth investigation Sheridan describes function allocation as an art, and leaves little hope, that function allocation itself can be found deductively.

While there are objective criteria, such as frequency and necessary response time limit of the task both exceeding capabilities of even the best-performing humans, as well as legal requirements (hardly known in IT management, in contrast to e.g. airplanes), the main problem is, that the correctness of all assumptions is unknown. Subjective criteria include policies, human wishes, human performance under stress in different ages, human education, experience, and differing mental capabilities.

All in all, we do not make this assessment EIMA's business. As stated in the non-requirements in Ch. 2, we also refrain from doing other influencing assessments:

- cost-benefit-analysis
- risk assessment and forecast of potential performance

- influence on human administrators
- compatibility issues

Such assessments of good/best automation going beyond automation feasibility, and including the compromises made are known as “Human Systems integration” or “Human Machine (Workflow) integration” in systems engineering. This is a research topic on its own, that so far has been performed only for a few critical systems, such as military vehicles, devices and equipment purchased by the US Department of Defense (DoD). It is on-going research for other domains, where expensive, rare devices are controlled by humans.

What is very important in function allocation: We need to gather the *reasons* for using a certain type of management automation routine plus the date, and should re-evaluate reasons after a certain period of time to keep pace with the state of the art in methods, while humans change more slowly.

As stated before, making function allocation a fully objective matter is very unlikely to be successful, as there always will be influencing factors, that may promote or hinder automation.

Function allocation should be reiterated from time to time to adapt automation to its users’ acceptance. The scheme for task analysis already contains a column for operator or machine ID, so that the reiteration itself resembles an improvement cycle aiming to achieve the most reasonable level of automation. Basic roles include local human managers, remote human managers, local machine managers, remote machine managers, so we have many options to allocate tasks to.

5.5.4 R3.4 Dynamic function allocation at operation phase

With systems getting more and more complex in terms of scale, heterogeneity and change dynamics, we need to find ways to abstract from this complexity and manage via a more high-level human machine interface. Therefore, we need ways to abstract from resource management being done via monitoring resources in a systems/network monitoring system and then reacting via ssh or telnet sessions.

From our point of view, the level of automation of built-in management automation loops is a property at a good level of abstraction to be managed by a human administrator. This would mean going from management of the managed system to managing the management system, i.e. managing the automation loops.

Shift of function allocation from systems designer to system admin

We cannot expect, that IT administrators will build automation profiles for every task according to their wishes/requirements. Much know-how is needed on software and systems development, which cannot be taken for granted with system administrators. It is also inefficient, if for a common type of resource, each customer would build his automation from scratch, instead of the system developer to at least offer an optional automation package. So, general automation design is indeed a matter of systems design. But in contrast to implementing a fixed level of automation, multiple, carefully chosen and documented automation levels should be implemented and function allocation should be left to the administrators in the operational phase to be managed. As we know from experience, the *default* configuration is to be chosen reasonably, as it is likely to be generically applicable to most customers. So designers should pre-select one way of automation (default level) but leave it to the admin to change the automation. The overall idea is to start at a level of automation, the customer feels most comfortable with, and then to allow change in cases of overtrust or undertrust.

Reasons for switching between multiple levels

A typical problem with automation systems are both undertrust and overtrust in automation at runtime. On the one hand, undertrust will make administrators/companies make use of the full automation potential. On the other hand, overtrust can make administrators blind for the weaknesses of the automation system. Offering multiple levels of automation and switching among them at runtime would enable administrators to adjust the level of automation to their trust in the system. As this trust is impossible to measure and changes over time, a dynamic level of automation for complex tasks makes sense.

The transition from the lower levels of automation to higher levels requires people (foremost administrators and users) to get in touch with, operate, test, and run IT management automation systems. As long as the automatic nature of these new systems is not taken for granted, systems should allow outside control over the level of autonomy. Benefits of systems with adjustable autonomy:

- Partial autonomy where full autonomy is not possible or desired.
- Lower cost: difficult-to-automate parts of the system can be left for (human) managers.
- Safety and reliability: human experience brought to bear when needed.
- Operator acceptance of management automation systems.
- One more important control for coupling management automation systems.

However, also see the skeptic view of Sheridan in Fig. 5.13 on page 166, that these optimistic goals may not always actually be achieved.

Idea: An automation manager user interface

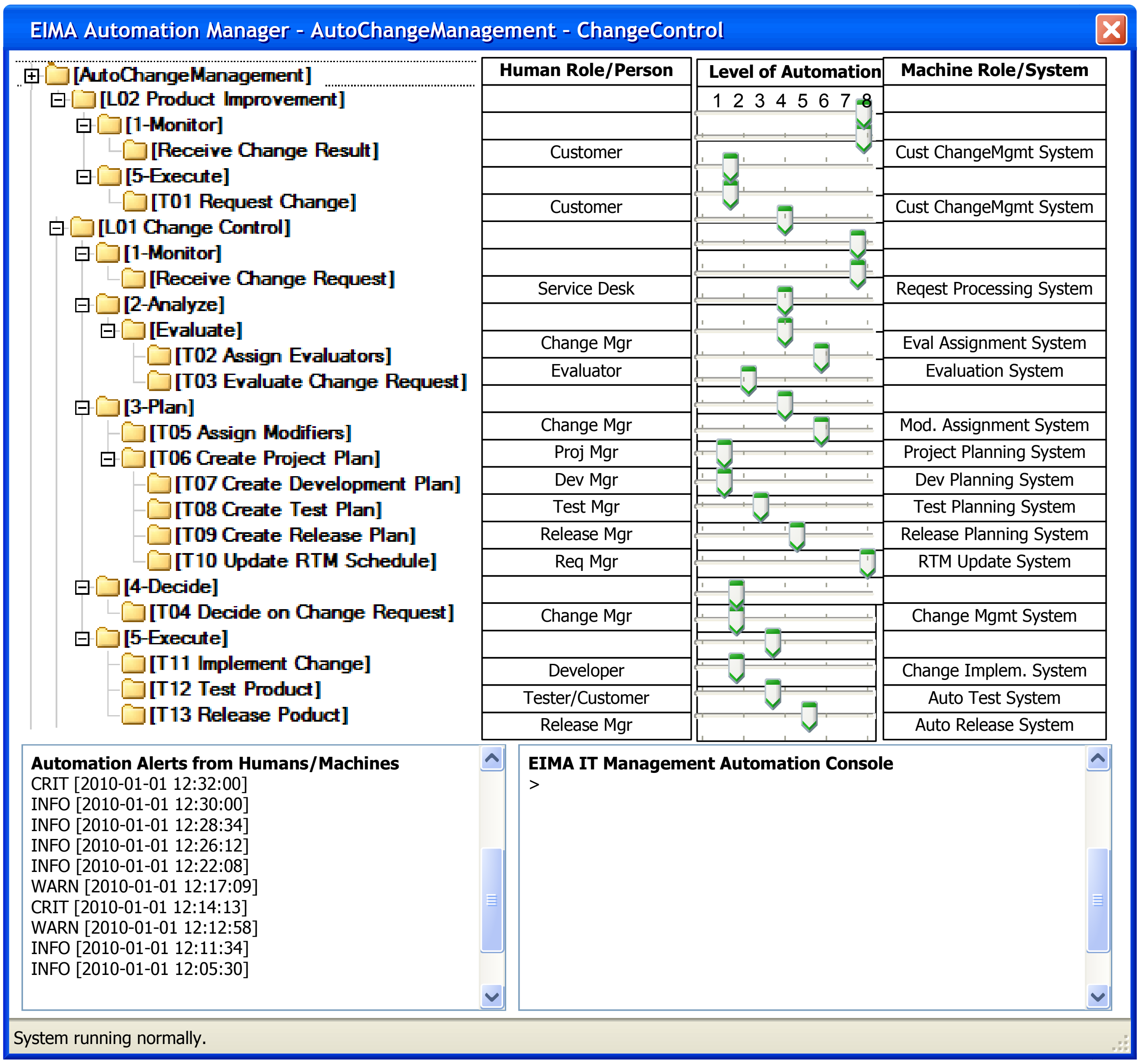
In a fictive central “Automation manager” matrix view in a management automation system (see Fig. 5.18) a human administrator could monitor the status of the overall automation and manage the management automation system by changing the level of automation for individual loops or loop steps. This view would also include responsibilities for the individual processes/loops/loop steps/functions and interaction capabilities with the loops.

Going even one step further, adaptive automation, i.e. automatic, context-sensitive adaptation of the level of automation would be possible, where the system takes over in urgent cases or the human takes over in case of automation inadequacies or disabilities.

Usage Example

Fig. 5.18 shows, how dynamic function allocation can be used in a tool. The view extends the scheme used in the tables in Section 5.5.1.

Generic view Taking a generic view on this figure, it uses the task decomposition tree (see Ch. 3) at the left-hand side for the tasks. There, the loops and loop steps can be expanded or collapsed so that a variable level of abstraction can be used. There is a slider for each of the loops, loop steps, function to change the level of automation along Sheridan’s list of automation levels. Doing so at the loop level or loop step level will interact with the sliders below in the task decomposition tree. In the same way, changing the level of automation below in a function will interact with the level above in the loop step


EIMA Automation Manager - AutoChangeManagement - ChangeControl

```
[AutoChangeManagement]
  [L02 Product Improvement]
    [1-Monitor]
      [Receive Change Result]
    [5-Execute]
      [T01 Request Change]
  [L01 Change Control]
    [1-Monitor]
      [Receive Change Request]
    [2-Analyze]
      [Evaluate]
        [T02 Assign Evaluators]
        [T03 Evaluate Change Request]
    [3-Plan]
      [T05 Assign Modifiers]
      [T06 Create Project Plan]
        [T07 Create Development Plan]
        [T08 Create Test Plan]
        [T09 Create Release Plan]
        [T10 Update RTM Schedule]
    [4-Decide]
      [T04 Decide on Change Request]
    [5-Execute]
      [T11 Implement Change]
      [T12 Test Product]
      [T13 Release Poduct]
```

Human Role/Person	Level of Automation	Machine Role/System
	1 2 3 4 5 6 7 8	
Customer		Cust ChangeMgmt System
Customer		Cust ChangeMgmt System
Service Desk		Reqest Processing System
Change Mgr		Eval Assignment System
Evaluator		Evaluation System
Change Mgr		Mod. Assignment System
Proj Mgr		Project Planning System
Dev Mgr		Dev Planning System
Test Mgr		Test Planning System
Release Mgr		Release Planning System
Req Mgr		RTM Update System
Change Mgr		Change Mgmt System
Developer		Change Implem. System
Tester/Customer		Auto Test System
Release Mgr		Auto Release System

Automation Alerts from Humans/Machines

```
CRIT [2010-01-01 12:32:00]
INFO [2010-01-01 12:30:00]
INFO [2010-01-01 12:28:34]
INFO [2010-01-01 12:26:12]
INFO [2010-01-01 12:22:08]
WARN [2010-01-01 12:17:09]
CRIT [2010-01-01 12:14:13]
WARN [2010-01-01 12:12:58]
INFO [2010-01-01 12:11:34]
INFO [2010-01-01 12:05:30]
```

EIMA IT Management Automation Console

```
>
```

System running normally.

Figure 5.18: User interface mock-up—related to workflow example 2 in Chapter 3—to manage a management automation system, here by dynamic function allocation. The tree view shows the task decomposition tree. There are sliders for function allocation along the Sheridan levels. For each task, human roles and machine roles are named. At the bottom are two views: 1) a list of escalations, and 2) a console interface to manage the system via commands.

or loop. By this hierarchical decomposition, we can manage many loops, but have it broken down when needed.

For each function, there is a human role, being responsible, and a machine role, that can either support the human in various ways or even take over his work. In the Figure, only role names were added, in a real scenario, the names of actual staff and actual system could be added. The slider is arranged in a way so that its position reflects, how close it is to human work (on the left) and how close to machines (on the right).

Therefore, a very quick assessment of automation potential is immediately possible, either at design time (when a mock-up may be used), but also at run-time in the application: sliders being close to humans obviously leave room for more automation. And we can quickly investigate this at process, loop, loop step and function level. When a slider is far to the right, the human role will mainly supervise the automation and only interact with it in case of automation failures. Here, the potential of shifting work from a human to the machine is small, but there can be high potential in improving the machine function with a better implementation in terms of efficiency or effectivity.

The typical person actually *using* this interface may differ from scenario: (Auto)ITIL process managers can use it to get on overview on allocation of tasks in "their" managed process, upper management may use this for strategic decisions on workforce planning as well as planning new IT management automation systems, the actual resource administrators may use it to see their tasks in context.

Please note, that the automation manager view shall not replace resource or service monitoring and management tools. Instead it should provide information on the links between staff and management automation systems.

Scenario-specific view Now, we take a look how the example itself has changed, when compared to the figures at the end of Ch. 3 and 4. First, we can see, that role names have been adapted to ITIL, where applicable (e.g. the requestor becoming a customer, and the service desk receiving the request instead of the change manager). Second, we have now added names of potential management support systems or management automation systems for each task. They are intended to be systems that do a great part of the human worker's job. For example, an automated evaluation assignment system could indeed apply rules to select evaluators based on objective, human-supplied criteria and forward the evaluation requests with minimal input (such as only a confirmation) by the responsible human role, a change manager. In this manner, we believe that for every single function, there are ways to come up with a management automation system used as a tool by the human role, or replacing the human role (apart from supervision). In this view, we arranged the sliders in a view, so that—as mentioned before—monitoring tasks' automation potential is actually used, analysis is supported by tools, planning will benefit from automation with mixed results, decisions will likely remain with humans and execution is automated in case not much creativity is needed to perform a previous built plan.

This way, we can quickly reduce overhead in the former workflow: simple processing of messages and requests can be done automatically, assigning evaluators and modifiers can be based on objective criteria, much of change planning can be done with planning support, development/implementation of the change will likely stay with humans (unless we had model-based design, where a change in a model would propagate into the next release), testing and release can be machine-supported.

Application advice on dynamic function allocation

The known, traditional way of static function allocation happens at design time of a system based on many criteria that are predicted for the operations time of the system and assumed to be correct over the life-time of the system. Static function allocation is therefore suitable for scenarios, that indeed can be predicted in a reasonable manner. Less flexibility at operations time needs more known properties, which at all enables planning automation based on these properties.

In contrast, dynamic function allocation postpones decisions about assignment of tasks to either humans or machines to automation managers at operations time. Such automation systems would allow a more dynamic allocation of functions. Both ways of function allocation can be used with EIMA.

Table 5.4 concludes the criteria and serves as a checklist of automation scenario properties to assess, which type of function allocation can be considered more suitable for a certain scenario.

It can be seen in the table, that none of both is generally "better" than the other one. A major difference between both types of function allocation can be witnessed, when requirements on automation change at operations time. With static function allocation this means that an IT service provider needs to wait for another release of the management automation system, which respects these new requirements (such as a changing level of automation, changing managed resource types, or a change in the number of managed instances of a resource/service type). In contrast, with dynamic function allocation, the system designer and implementer have implemented multiple automation levels, configurable function allocation and evt. scalable capabilities. For this reason, the automation is flexible enough to allow adapting to changes in automation requirements by reconfiguration of the management automation system.

Allow an administrator to manage responsibilities Another major problem that can arise around automated or remote management is lack of responsibility by system operators in case of machine errors, once roles are mapped to automated managers or remote human managers.

Having experienced bugs in all kinds of software, firmware and devices created to assist them, administrators and operators will initially demand control on certain aspects of management automation systems. With an adjustable level of automation they can get accustomed with the autonomic management solution, experience its behavior, build up trust and will step by step demand less and less control over it. There is a more detailed discussion of the relationship of trust and control in [CF00].

They can use the management automation system in certain cases only for a short time (say, at night, at weekends or in cases that they cannot handle themselves), while disabling automation features in cases of failure analysis to reduce complexity.

In general, management automation systems should not assume, that they are always right, instead they should respond to management actions at all time with fine-grained re-start or reset functions. They should be able to test themselves and (ideally) to repair themselves.

To link the human operator's world and the management automation mechanisms and to prevent confusion about responsibilities, the automation manager should therefore also feature the additional known concept of RACSI matrices (Responsible, Accountable, Consulted, Supportive, Informed), for the individual tasks. Existing RACSI diagrams, that cover only local human roles, would therefore be extended to include machine roles and remote human roles in addition to local human operators.

Despite automation and remote management, local human operators or managers should remain responsible and accountable. We could assign "Accountable" only to humans, while "Responsible", "Consulted", "Supportive", and "Informed" may be assigned to machines and humans.

Table 5.4: Influence factors on choosing static or dynamic function allocation for a certain automation scenario

Pro static function allocation	**Pro dynamic function allocation**
Human-related factors	
known capabilities of IT system administrators and IT service managers, standardized professional training	training of staff unknown/unpredictable upfront, eventually varying based on customer
known long-running contracts for workers	unknown availability of staff over time
known working hours, night operators	job-sharing, flextime, flexplace work positions
fixed attitude of staff/top management on automation	changes in attitude towards automation likely
Policy-related factors	
known legal requirements or fixed long-lasting corporate policies on function allocation	unpredictable changes and dynamics in either corporate policies or law with quickly needed reactions
Machine-related factors	
known capabilities and maturity of IT management automation systems	unclear machine capabilities of management automation systems, either on the negative side (systems does not meet expectations) or on the positive side (system profits from new capabilities and performs better than predicted)
known or predictable resource environment	dynamic resource environment
predictable tooling, as well as predictable compatibility and cooperation of tools	variable tool landscape, loose coupling of management automation systems
Service-related factors	
predictable number of service and service instances, predictable quality of services	varying number of service instances per type, varying requirements on service quality

At first, RACSI diagrams in management user interfaces may be documentary (as used today in process descriptions), once automated management routines exist, changing the RACSI settings may represent a real hand-over of control, like a pilot enabling autopilot linked to the Flight Management System for a certain period of time. Approaching more and more machine-machine interaction, RACSI with machine roles may clarify conflicts between automated management loops. Of course, the implementation for the RACSI levels requires much work on protocols and models. This way, the features of management automation system could be reached: Just enough management by human administrators on a human-understandable level of abstraction.

Keep administrators fit in low-level management Automation should not be taken for granted to work exactly as expected at all times. So, like pilots, administrators should train regularly to detect automation defects quickly before they can do much harm. For the most likely automation disabilities they should be trained on how to react and revert changes without much automation tool support.

Otherwise, in a similar way to pilots who regularly fly on autopilot coupled to a flight management system being prone to loose hand-flying skills over time, low-level administration skills might be affected or get lost with system administrators. It much depends on the scenario, whether this is considered as harmful to overall operations or not. We have investigated some of the pro's and con's to this matter before. For IT administrators the situation is a little different because of their non-standardized professional training and widely varying mental skills.

A global undo/redo function for automation (common in desktop applications today) would be most welcome but seems only for a small number of real-world cases. There are some operations that cannot be undone (e.g. erasing data without having a backup, or changing a password that then is necessary for changing it back), such operations should be done with special care, just like a pilot who drops fuel tanks in order to reduce the risk of fire of a potential crash landing cannot get them back to e.g. reach a certain landing strip.

5.5.5 R3.5 Computer-aided modeling of function allocation

As before with tasks and loops, *engineered* IT management automation in the sense of EIMA means, that we model all artifacts in a machine-interpretable format in a suitable application. Doing so, scalable and flexible computer-aided modeling better copes with growing models and replaces pen-and-paper or verbal descriptions.

Proposal on doing function allocation in EIMA

We assume that the following prerequisites exist from previous chapters:

- Task decomposition tree (see Fig 3.8, example in Fig. 3.14) to get a list of tasks. Each task has at least a task ID and a task name.
- Tasks are associated with loops and assigned to MAPDEK categories (see example in Fig. 4.19 and 4.20).
- List of roles, include both: likely human roles and likely machine roles, but do not tag them with "h" and "m" yet to give an automation engineer more options to do function allocation. Each role has a role ID, and a role name. If applicable, create a role control hierarchy (see example in Fig. 3.13).

- A task table lists basic information about each task relevant for automation (see example in Fig. 3.15).
- The catalog of machine capabilities (see Ch. 6) is available for reference.

The following list of steps now describes how to do function allocation in EIMA:

1. Create a basic function allocation matrix (see Fig. 5.15) that only contains the decision on mapping tasks to either humans, or machines, or both. Doing so, first try to match the tasks with machine capabilities in the catalog. It is structured along the EIMA structure of AutoITIL processes, loops, loop steps, which all serve as an index to identify machine capabilities quickly. In general, when doing function allocation, keep in mind the general advice on automation feasibility in section 5.5.3, which says which MAPDEK categories are more likely to be automated than others.

2. Refine the function allocation matrix and break the basic allocation to concrete human/machine roles. This can be done along the task decomposition tree (see Fig. 5.16 and 5.17) in a top-down manner first for processes, then refined for loops, then refined for loop steps, then refined for functions. If an automation engineer feels more comfortable doing function allocation bottom-up, this of course is also perfectly ok. Here, allocation of composed tasks can be derived from allocation of sub-tasks. Involvement of roles can first be marked with simple "X" check marks in the matrix.

   When doing this step, treat each column in the basic function allocation matrix differently:

   a) Function allocation to machines (first column checked): First, identify the involved machine role(s). If appropriate machine roles do not exist in the original workflow or system design, create them. Finally add an accountable human role.

   b) Function allocation to humans and machines (second column checked): identify the involved machine and human roles. Likewise create roles, that are needed but do not exist so far. Make one human role accountable.

   c) Function allocation to humans (third column checked): identify the involved human roles, make one of them accountable. Check that no machine role can be responsible in this case, but they can be consulted or informed. To express that this is a human task, the human should not rely on the machines, however. The human should be able to accomplish the task even without machine support.

   Tag each role explicitly with one of: human, machine (see section 5.5.1.). A simple way to create and tag roles is to derive an hRole and an mRole from a certain Role, such as an hTestManager and an mTestManager from a TestManager role. For better comprehensibility and more variation in names and staying in line with current role names, the hTestManager can also remain as TestManager (implicitly assumed to be human role) and the mTestManager can be named as Test Management System (implicitly assumed to be a machine role).

3. Refine all "X" entries in the matrix to entries along the RACI scheme. Add additional RACI entries if considered necessary.

4. Optional for *dynamic* function allocation: For all tasks assigned to humans and machines, decide on whether to allow dynamic function allocation and decide on an appropriate automation level.

Please note: Doing function allocation is never a completely objective matter. When in need for additional information on tasks, use the task table or refer to people knowledgable in the tasks.

When doing function allocation, in parallel document the *reasons* for deciding this way for the particular scenario. This can be notes, calculations, referring to requirements on e.g. duration of task execution, task frequency, corporate policies or laws, input by IT management staff, or even intuition of the automation engineer. It is not so much important to have an objective way of justifying each decision, but it is important that later those being justified by certain causes can be easily identified e.g. when doing function allocation in a next iteration of a continuous improvement cycle. In such cases, the causes can be checked, whether they are still valid, or whether they may need to be replaced.

Application example on Workflow 2 from Chapter 2

Step 1 "Create a basic function allocation matrix"

	Machine	Machine and Human	Human
T01a Decide on Change Request			X
T01b Send Change Request	X		
T01c Receive Change Request	X		
T02 Assign Evaluators	X		
T03 Evaluate Change Request		X	
T05 Assign Modifiers	X		
T06 Create Project Plan		X	
T07 Create Developm. Plan		X	
T08 Create Test Plan		X	
T09 Create Release Plan		X	
T10 Update RTM schedule	X		
T04 Decide on Change Request			X
T11 Implement Change		X	
T12 Test Product		X	
T13 Release Product		X	

To do basic function allocation, the basic tabular scheme was created first. Tasks were taken from the result of Chapter 4, where they have been aligned to two MAPDEK loops. Function allocation in this scenario is justified as follows:

- T01 Receive Change Request has been split into three parts to separate the decision to submit a change request from the actual message submission and reception. A human decides/approves to have a change requested (as this may involve terms on contracts, money), the actual message transfer happens automatically, as a communication systems can easily be used to submit form data. This can later be implemented via e.g. a web service, a HTTP post request, or even an e-mail with a standard format.

- T02 is automatic because we assume distribution based on either a fair basis (making this a simple round-robin scheduler for example), by availability (easy to automate), or tags/keywords, that also appear in the change request (automate with a classifier).

- T03 is semi-automated, because human evaluation is unlikely to be completely automatic. The machine part however can take care of simple steps to automate first, such as checking for duplicate request, deadlines, authorization.

- T04, see T02. Here, we also assume that there is not much knowledge involved, as it is typically clear whom to assign certain tasks to based on job function or availability.
- T06 to T09 are planning activities. They can be supported by planning tools.
- T10 is a simple update that will result from the plans. As there does not seem to be much strategy involved here, we can automate this simple update.
- T04 is the central decision. A change manager will review the whole change request, and also will be able to see all the plans to enable it and its effects on the ready-to-manufacture date. We leave this to a human to decide, because it's hard to express all the involved policies, constraints, strategy to a machine.
- T11, T12, T13 are semi-automated as implementers, testers, release staff will likely be supported by machines. The extremes are highly unlikely.
- T11 Software implementation is a field on its own, which we will not investigate here.
- T12 Testing is likely to be automated with unit tests, component and system tests.

Step 2 "Refine the function allocation matrix and break the basic allocation to concrete human/machine roles"

	Customer Change Management System	Request Processing System	Evaluator Assignment System	Evaluation System	Modifier Assignment System	Project Planning System	Development Planning System	Test Planning System	Release Planning System	RTM Update System	Change Mgmt System	Change Implementation System	Product Test System	Product Release System	R01 Customer	R02 Change Manager	R03 Evaluator	R04 Project Manager	R05 Development Manager	R06 Test Manager	R07 Release Manager	R08 Requirements Manager	R09 Modifier	R10 Service Desk
T01a Decide on Change Request											X				X									
T01b Send Change Request	X										X				X									
T01c Receive Change Request		X									X					X								X
T02 Assign Evaluators			X								X					X								
T03 Evaluate Change Request				X							X						X							
T05 Assign Modifiers					X						X					X								
T06 Create Project Plan						X					X							X						
T07 Create Developm. Plan							X				X								X					
T08 Create Test Plan								X			X									X				
T09 Create Release Plan									X		X										X			
T10 Update RTM schedule										X	X											X		
T04 Decide on Change Request											X					X								
T11 Implement Change											X	X											X	
T12 Test Product											X		X							X				
T13 Release Product											X			X							X			

Now, we need to refine function allocation to actual roles. The following reasons support function allocation as performed in the figure:

- Machine roles were created for all tasks that need machine support (all, except the two human decision tasks). Here, we see how AutoITIL offloads humans but structures its own system organization along the known human process.

- The requestor was renamed to customer to reflect the position of the requestor in ITIL.
- A service desk role was added as this the more likely point in ITIlL, where a change request will be received first.
- The assignment of tasks to machine roles follows directly from the basic function allocation scheme.
- Being part of the AutoChangeManagement process in AutoITIL, a change management machine role was added that acts as a central information system for this process and is involved in all tasks.
- At this point we change one aspect in T12: the customer/requestor of the change is no longer responsible for testing the change, because it seems unbeneficial to wait for his approval for time reasons. Instead, we assume that he has submitted test criteria with his change request, so that the tests can be done on-site and at the own pace of automation.
- T13 was moved from "modifiers" in the original workflow to the release manager. Modifiers seemed to be personnel implementing changes, so people who write code. Along ITIL however, release management is better done by a release manager, who in EIMA is supported for the routine tasks with a release management system.

Step 3 "Refine all X entries in the matrix to entries along the RACI scheme"

	Customer Change Management System	Request Processing System	Evaluator Assignment System	Evaluation System	Modifier Assignment System	Project Planning System	Development Planning System	Test Planning System	Release Planning System	RTM Update System	Change Mgmt System	Change Implementation System	Product Test System	Product Release System	R01 Customer	R02 Change Manager	R03 Evaluator	R04 Project Manager	R05 Development Manager	R06 Test Manager	R07 Release Manager	R08 Requirements Manager	R09 Modifier	R10 Service Desk
T01a Decide on Change Request											I				RA									
T01b Send Change Request	R										I				A									
T01c Receive Change Request		R									I				I	A								R
T02 Assign Evaluators			R								I					A								
T03 Evaluate Change Request				R							I						RA							
T05 Assign Modifiers					R						I					A								
T06 Create Project Plan						R					I							RA						
T07 Create Developm. Plan							R				I								RA					
T08 Create Test Plan								R			I									RA				
T09 Create Release Plan									R		I										RA			
T10 Update RTM schedule										R	I											A		
T04 Decide on Change Request											I				I	RA								
T11 Implement Change											I	R											RA	
T12 Test Product											I		R		C					RA				
T13 Release Product											I			R	I						RA			

Finally, we change the "X" check marks to RACI categories. At this many changes are mechanic with the following few exceptions:

- The customer/change requestor needs to be informed on the progress of his change request. Such information were added to the reception of the change request, the decision about its approval and the final release. We believe, this set is small enough but still keeps the customer informed enough. Of course, these notifications can easily be sent automatically.

- In T12, the customer/requestor is again included to be consulted during testing. We believe that this optional feedback is a good compromise between 1) totally relying on the customer to test, waiting for his approval and 2) not including the customer at all.

Step 4 "Decide on dynamic function allocation" Refer to Fig. 5.18 for an illustration of implementing this scenario with dynamic function allocation. As can be seen, in this view many sliders exist that have been set to automation levels similar to the one in static function allocation. The major difference here is that change in the distribution of control itself can be controlled via sliders and changed at operation time. The level of automation that is considered most reasonable by the customer can therefore be found over a certain time frame. Overtrust and undertrust in automation can quickly be counteracted with changing the level of automation. It can also be used for troubleshooting automation or changing function allocation over periodic time intervals.

Information model

An information model for static and dynamic function allocation is depicted in Fig. 5.19. It shall serve as an model about the necessary information, that is intentionally kept free from implementation details.

In this model, roles can be refined to be either associated with humans or machines.

Data on static function allocation is essentially matrix data, that expresses an association of tasks and roles. Entries in these matrices can be binary (checked, unchecked) or from the RACI (responsible, accountable, consulted, informed) set. When role names are given, then static function allocation deals with assigning the roles to either machine or humans. Static allocation table entries are accompanied by metadata that lists reasons, why function allocation has been decided to be done as it is modeled.

Data on dynamic function allocation assigns two corresponding roles to a task: a machine role and a human role, as well as "slider" to gradually assign automation more to one of both extremes.

Current information models in IT management cover only a part of this information. The Common Information model (CIM) has concepts on users, administrators, roles, organizations, but no tasks, no allocation. Likewise, the Shared Information and Data Model (SID) lacks tasks and allocations.

Data model and applications

Appropriate tool support The general requirements on a tool for function allocation alone are low. One could use any spreadsheet application, or even a text editor. Where an application for function allocation really can support the automation designer is in consistency checks, group operations, and different views on a modeled distribution of functions. In EIMA with its layered concept for AutoITIL processes, tasks, loops, loop steps, functions, there are dependencies between these composites that should be taken into account during modeling.

At system design phase At system design phase, a SysML tool can be used to create information on function allocation, especially when modeling the actual IT management automation system.

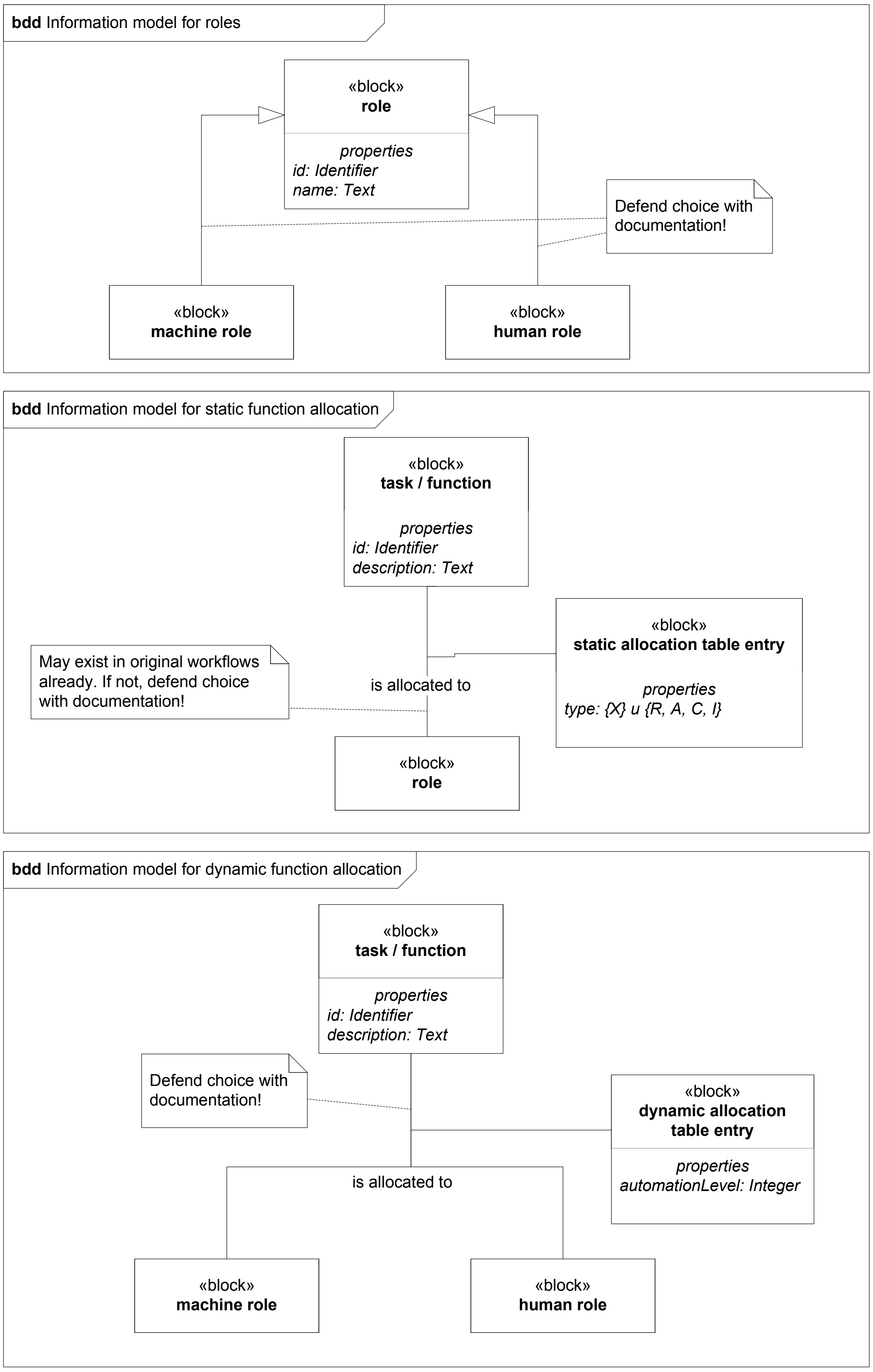


Figure 5.19: Three diagrams describe the basic information model for function allocation in EIMA

SysML allows modeling function allocation seemlessly inside a number of its diagrams: activity diagrams, block definition diagrams, internal block definition diagrams can all be used to model function allocation. SysML tools can create tables/matrices from these models to give a concise, compact overview without further detail. Unfortunately, a "function allocation" table itself is not part of SysML (see [FMS09], Chapter 13). Instead, we can use the `matrix` diagram type to express relationships between model elements. For the annotations we can use "note symbols" in SysML. As with BPMN and as an alternative, SysML also allows modeling with swim lanes to express function allocation.

While SysML does indeed cover some aspects of function allocation, even the simple form that is described in the information model leaves things to be desired, such as RACI marks instead of simple on/off allocations. With the matrix element having a standardized graphical representation but no common representation in XMI or MOF, it will also not result in models interchangeable between current SysML applications. Both may be attributed to the relative youth of SysML, and future versions of both language and tools may likely improve this point.

For dynamic function allocation, multiple levels of automation would have to be designed. In general, at systems design phase, system designers should work together with later system operators to create automation levels most useful in actual practice. One could also think of systems, where extra automation levels are offered as updates or add-ons. This can be done for marketing reasons (creating product diversity in features and pricing), but it could also be done to shorten time-to-market, or to wait for machine capabilities to catch up with the resource demand of certain automation methods. And remote services or maintenance plans can be add-ons that add the remote administration site in the list of roles to delegate tasks to. With support for function allocation being limited, dynamic function allocation will necessitate extensions of current system design tools or the system designer modeling it, and later developers to have it implemented. Fig. 5.18 shall serve as an inspiration towards this goal, which could be used at design time as well.

At operation phase Clearly, static function allocation hardly leaves room for changes at operations time. Instead, changes in function allocation in such preplanned environments will likely be handled as incidents (e.g. management automation systems failing and humans having to take over). Changes in static function allocation will make necessary a new implementation of the system.

Therefore, dynamic function allocation is more relevant at operations time. To repeat this: Changing function allocation at run-time is not a mandatory design decision in EIMA. Instead, it is an additional feature, if there is high uncertainty on function allocation at system design time (see Tab. 5.4) and the system designer can predict that potential customers will want to influence function allocation at run-time.

For dynamic function allocation at operations phase, EIMA proposes to use a tool called "automation manager", that was briefly presented before (see Fig. 5.18). It would allow administrators or service managers to manage function allocation at run-time. As stated before, the creation of the automation manager is a matter of systems design. At run-time it is meant for switching between the automation levels and to monitor automatic switches by adaptive function allocation.

In its basic version, the automation manager tool itself would be the same for all customer environments, it is operated in. To make an automation manager component even more specific to the customer environment, several additional functions could be added:

- mapping human roles to actual people in an enterprise
- mapping machine roles to external equipment (e.g. a database or an LDAP server) outside of the actual delivered IT solution

- a change calendar: Change is a major contributor to complexity, but many changes affecting function allocation are predictable and can be documented in a calendar, similar to a forward schedule of changes. For example new resources, resources going out of service, new management systems, management system going out of service, newly hired staff, ill staff, dismissed/retiring staff, and so on. Also maintenance windows are an exception from normal operations, where typically function allocation changes. A calendar on these matters would allow in-advance planning of function allocation, responsibilities, automation levels.

- historic function allocation data: a calendar does not only have to cover the future, for distinct reasons it can also be necessary to know function allocation of the past

- showing relationships of the loops to the managed resources, i.e. integrating a view of the resource pool and overlaying the tasks and loops.

With this extended function set, it might interfere with (1) work planning and organization tools, (2) worker presence tools and (3) resource/service monitoring. While exactly this cross-connection point was EIMA's intention to address, it—again—depends on the situation of the customer, whether this interference fills a gap, that long needed clarification, or whether it is seen opposing existing service, process and resource management organization and tools.

Format for data exchange about function allocation Ideally, an interchange format would exist, that allows to export and import function allocation data to and from modeling applications, CASE tools and management automation systems.

As we have seen above, EIMA function allocation can itself start simple, in fact basic function allocation matrices, automation levels, and responsibilities can all be modeled as matrices. Function allocation tables, automation levels, responsibility matrices can therefore be created in a spreadsheet application, or relational database, or in a text file format (e.g. CSV) or XML, but none of these would be standardized. Instead, EIMA proposes to use the SysML "matrix" diagram type. As shown in Fig. 5.17 on page 174, this way tasks at all EIMA abstraction levels can be mapped to roles. Despite of their shortcomings, SysML matrices are at least intended for this kind of mapping between modeled items.

As a practical alternative for the time that SysML applications are not yet ready for EIMA's type of function allocation, spreadsheet applications may be used.

EIMA proposes to use the same format both at design time as well as runtime, so that the data (and potentially the function allocation history) of many customers from the operational phase can be used for the next iteration of system development.

Clearly, this basic format could be extended to include comments, describing why function allocation was performed the way it has been done. One could also think of graphical views, where the familiar swim lane view would be recreated for visualization for humans. Or where the loop hierarchy is shown, with roles or people attached. However, here the limitations are much more in the application, than the format.

To keep historic data on function allocation, again there are a range of more or less equivalent ways to store the data: multiple text files, a database, XML, multiple XMI files. Each of them would have its advantages and disadvantages and the "best" way would more depend on the situation, than it makes sense to propose one specific way here.

Using the models

Ideally, we could draw conclusions/predictions about the real world after modeling a certain automation scenario. To do so, function allocation at system design time could be extended by basic usage frequencies, event frequencies and duration and therefore basic timing models. Once this data was added, we could assess whether a certain human role or machine role does even have a chance to do the work properly, or whether there are obvious reasons that the function allocation under review cannot work at all e.g. for time constraints or scalability constraints. Making theses estimations early at design time would positively effect system design as the necessary function set of machine tasks would be better known in advance.

To give an example of the state of the art, NoMagic, a company creating a SysML modeling applications (MagicDraw with SysML plug-in) lists the support of the following uses of its mostly graphical models when used with other third-party products or products by the same software company (reproduced verbatim from SysML Plug-In description at `http://www.nomagic.com/`):

- ParaMagic plug-in offers an expansion of the power of SysML parametric simulation, allowing users to integrate Microsoft Excel, MATLAB/Simulink (The MathWorks, Inc.) and Mathematica (Wolfram Research, Inc.) into their MagicDraw SysML models and run simulations from the earliest stages of system design.
- Cameo DataHub allows the user to import, export, synchronize, and make references between SysML Requirements and text based requirements in tools like Cameo Requirements+, Telelogic DOORS, Rational RequisitePro and Microsoft Excel (other tools are to be supported soon).
- fUML MagicDraw plug-in for executing activity models.
- UPDM and DoDAF plug-ins for systems modeling within Systems views in military architecture frameworks.
- SYSMOD profile provides a toolbox for Systems Modeling Process support.
- MARTE profile adds capabilities to SysML for model-driven development of Real Time and Embedded Systems (RTES).
- Xholon runtime framework transforms MagicDraw models into executable Java code for simulation of composite structures, interactions through ports and hierarchical state machines.
- SinelaboreRT generates C / C++ code from MagicDraw state machine models for embedded real-time and low power application developers.

We can learn from this example, that a matter as abstract as function allocation seems not as of yet to be on the agenda for tool makers. While integration of a whole tool eco-system indeed takes place, function allocation across system boundaries (e.g. to humans or from humans) or simulations at the function allocation level are apparently not at the level of abstraction of current users of SysML modeling application.

5.5.6 Summary

EIMA recognizes that function allocation, i.e. to decide whether a certain function (base-task) is performed by a human or a machine, has different aspects to take care of. Simple models where for each task an allocation tendency is documented are the first step that more or less document intentions or goals that need to be refined. To keep large models manageable here, EIMA introduces function allocation at the levels of abstraction, that have been defined before: processes, loops, loop steps, and functions. This hierarchical breakdown allows grouping and simple inheritance where applicable.

The simple model of black or white allocation to either man or machine is then extended with automation "shades of gray", thus modes, where human(s) and machine(s) cooperate to jointly solve a task. EIMA sees use in generic and function-dependent automation levels and specifies patterns so that an automation designer can at least get an idea of this human machine cooperation.

Regarding the best automation for a task, EIMA splits automation feasibility into finding relevant machine capabilities and then assessing them in a certain scenario. Based on input from systems engineering we drop the second half as this is unlikely to be solved within this thesis based on previous experience and the subjective influences on this matter, such as human preference, regulations, policies, which all may hinder a purely technical answer. Instead, we give a more general scheme aligned to the MAPDEK loop, that assigns qualitative marks to the individual steps.

To address this on-going conflict which level of automation is right in a certain situation, EIMA recommends, but does not require dynamic function allocation, where the automation level can be changed at run-time by the administrator or by the machine. This also includes documented responsibilities, which need to in sync with decisions on function allocation and automation levels.

Finally, EIMA proposes to use SysML `matrix` constructs to model function allocation, automation levels and responsibilities instead of other non-standard approaches. This simple construct leaves room for improvements but at least it is a standard way and can easily included in the systems engineering of IT management automation systems.

5.6 Conclusion

Function allocation is the heart of automation. It is the step where the tasks are distributed to men and machines, and therefore a fundamental but also non-trivial task to do. Requirements on function allocation in EIMA have been set high to really achieve an increase in the domain of IT management automation. This domain itself totally lacks function allocation as being a scientific way to task distribution and by its definition of RACI charts so far neglected machine roles in IT management, concentrating on human processes. The careless and unworried handling of existing automation in this domain surprised us, and proved that IT management automation has not yet the standing in this domain, that automation in general has in other domains.

As planned, input from systems engineering helped a lot at this point to benefit from concepts and insights into function allocation in systems like airplanes, air-traffic control, cars, ships, industrial automation. There we could find good results on a number of requirements, which were transferable to our domain despite all differences.

A proposal on function allocation in EIMA finally applies the available knowledge onto IT management automation, creating a concept how to introduce more automation in a sensible way in IT management.

6 Step 4: Matching tasks aligned to the EIMA scheme with machine capabilities

Contents

6.1 Motivation for a catalog of machine capabilities

Chapter 5 of this thesis described function allocation to be the “heart” of automation. This is only true, as long as we speak of automation *planning*. Of course, what really matters in an actual automation scenario is automation *implementation*, which in turn needs machine capabilities to perform tasks.

Therefore, function allocation cannot end as a “wish list” about the distribution of tasks. This happens far too often with requirements specifications, which may disregard the trading-off between reusing existing mature system components even if they marginally disrespect requirements vs. implementing new modules costly and with unknown maturity only to meet some special requirements, which may not even have been very severe.

Example: A distributed resource management scheme for a composed service by multiple partners may have the requirement of distributed knowledge and no access to resource information among the partners. This can be a privacy requirement that is not caused by law but simply have been added due to lacking initial trust among the partners. Clearly, such a requirement disallows most central automation schemes that rely on global knowledge. However, such central schemes with global knowledge have been applied very successfully in related scenarios. In other words: The partners must trade-off between their privacy requirements (resulting in a probably inefficient scheme) and their automation requirement (effectively and efficiently improving the management of the composed service). In a requirements-oriented approach, the privacy requirement would rule out all solutions, that do not fulfill it. In EIMA we want to look into the feasible alternatives for automation, and also question requirements whether they needlessly disallow automation.

The risks of function allocation (especially in its static version) for an automation engineer are twofold: 1) missing automation opportunities that do exist but are not known to the automation engineer (this is related to undertrust in automation at design time) and 2) overrating automation opportunities and allocating tasks to machines, that they cannot perform (related to overtrust in automation at design time).

For these reasons, EIMA needs to look into the subsequent steps that follow after function allocation on the path to an actually implemented IT management automation system (see Fig. 6.1). In this figure, the top section represents the results of Ch. 3-5: tasks having been been broken down and categorized, MAPDEK loops having been defined and associated with AutoITIL composite tasks.

Then, for each of the tasks/functions in the loops function allocation applies a lot of influence factors to trade-off between an allocation to human roles vs. an allocation to machine roles. Here, there are push/pull factors for each of both sides, as well as reasons such as task frequency, task duration time, general policies, applicable law, and a good portion of intuition, which had all been explained in Ch. 5. This results in a *preferred* function allocation at this point, as the actual machine capabilities have not been taken into account yet.

So, subsequently machine capabilities need to be identified to actually implement the machine tasks. At this point, we will witness a filtering effect: On the one hand, existing machine capabilities will attract tasks towards the automation side, on the other hand missing machine capabilities will cause a reallocation of the task to the human side. We silently assume at this point that humans can do all tasks, even if they may be slow or imperfect. In practice however, there will be a reallocation effect from humans to machines as well, if humans simply cannot cope with certain tasks.

Clearly, such reallocation of tasks should be done as early in the system design as possible to avoid confusion in the system design and system implementation process, as well as with future administrators and users.

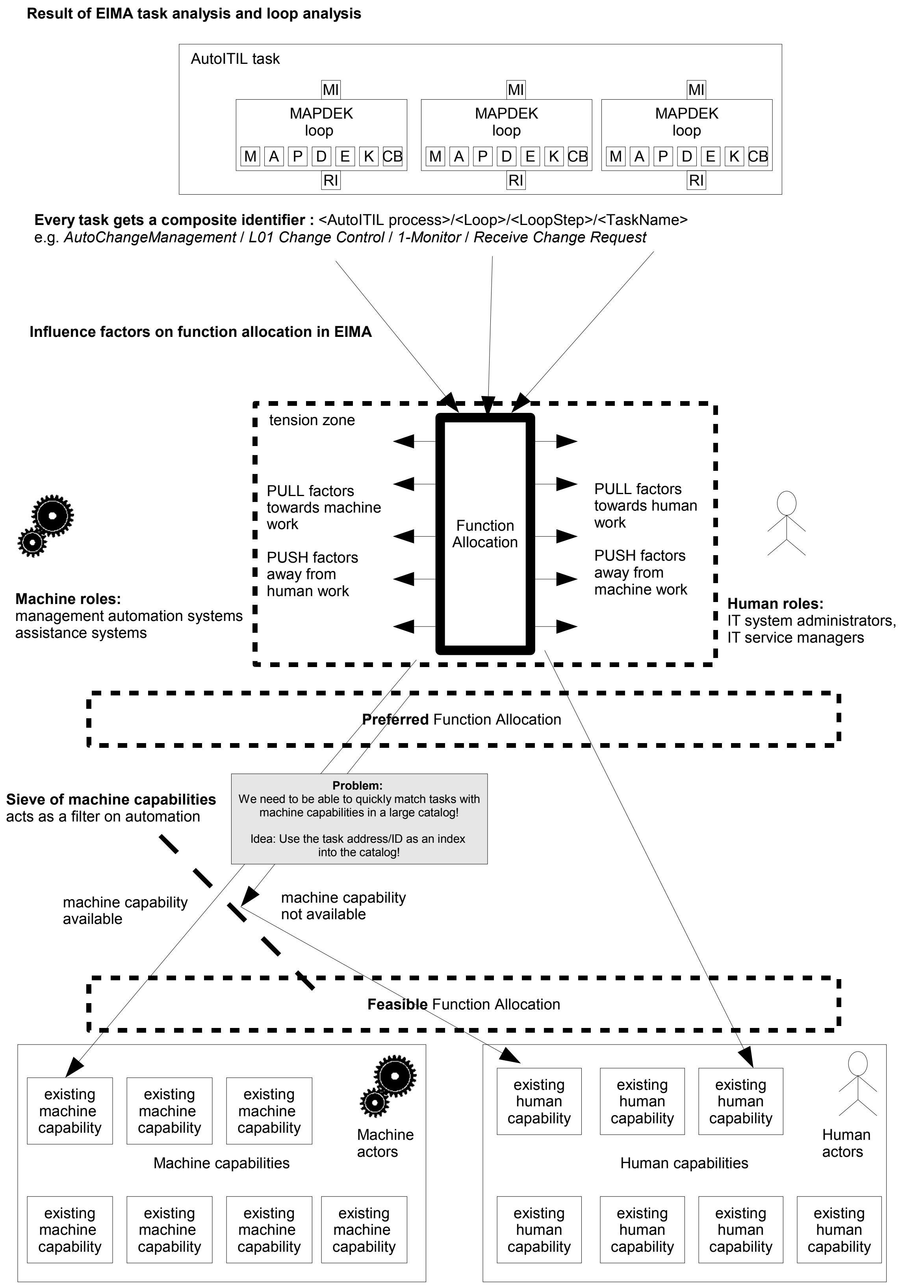

Result of EIMA task analysis and loop analysis
AutoITIL task
MI
MAPDEK loop
M A P D E K CB
RI
Every task gets a composite identifier : <AutoITIL process>/<Loop>/<LoopStep>/<TaskName>
e.g. AutoChangeManagement / L01 Change Control / 1-Monitor / Receive Change Request
Influence factors on function allocation in EIMA
tension zone
PULL factors towards machine work
PUSH factors away from human work
Function Allocation
PULL factors towards human work
PUSH factors away from machine work
Machine roles: management automation systems assistance systems
Human roles: IT system administrators, IT service managers
Preferred Function Allocation
Sieve of machine capabilities acts as a filter on automation
Problem: We need to be able to quickly match tasks with machine capabilities in a large catalog!
Idea: Use the task address/ID as an index into the catalog!
machine capability available
machine capability not available
Feasible Function Allocation
existing machine capability
Machine actors
Machine capabilities
existing human capability
Human actors
Human capabilities

Figure 6.1: Function allocation assigns *preferred* roles to individual tasks. To achieve a feasible function allocation, machine tasks need to be associated with actual machine capabilities. Otherwise, they will be bounced back to humans actors. Matching tasks with machine capabilities is therefore an integral part of the design of management automation systems.

Three ideas, that are based on each other, originate from the problem of unknown machine capabilities:

Idea 1: A catalog of machine capabilities specifically for IT management automation We would much profit from an overview of machine capabilities in IT management, and especially in the AutoITIL subset in EIMA. Fitt's list (see Fig. 5.12) gave a rough general estimate on human/machine strengths, but it does not contain the improvements in the last 50 years, it is not specific to IT management automation, and way too abstract for a task by task decision on automation feasibility. The catalog will be presented in sections 6.5 to 6.7.

Idea 2: Matching tasks and machine capabilities along the EIMA structure We can predict at this time, that the list of individual tasks (as results of task analysis and leaf nodes of the task decompositions tree) will cover a wide range. In the same way, the list of machine capabilities in the scope of IT management automation will be enormously large.

Semantic matching of tasks and machine capabilities would therefore be inefficient and tedious, if both of these sets were organized as simple plain lists. What makes matching tasks and machine capabilities (even at a coarse level and not taking into account implementation details such as performance or resource demand) non-trivial here, is the lack of structure. So what remains as a question in this "sieve of machine capabilities" is how to speed up the matching, without constraining both sets.

At this point, we can make use of the EIMA structure to categorize the set of likely machine capabilities into: 1) machine capabilities associated with the AutoITIL processes (see 6.5), 2) machine capabilities associated with loops (see 6.6), and 3) machine capabilities associated with loop steps (see 6.7).

This way, we can reduce the matching problem in the following way: To later find the matching categories, in EIMA, each task has an identifier, that can be derived from the EIMA structure: as each task is assigned to a loop step, then each loop step assigned to a loop, and each loop assigned to an AutoITIL process/composite task, we have the composite task ID: `<AutoITIL process>/ <Loop>/<LoopStep>/<TaskName>`. This way, in addition to the `TaskName`, we have three category names, that can be used individually as indexes into the catalog or as compound index. Thus, we immediately find three categories and can use the `TaskName` for finding appropriate machine capabilities within the categories (with support of human semantic matching).

Example: For a task with task ID `Process:AutoIncidentManagement / Loop: Inci/ dentCategorization / LoopStep: Analyze / Task: Assign Incident to Re/ sponsible Unit` we can immediately match the three following categories: (see Fig. 6.2 for an illustration)

1. Index 1 "AutoIncidentManagement": all machine capabilities in AutoIncidentManagement, with the "Analyze" tasks being highlighted or ranked first.
2. Index 2 "IncidentCategorization": in an opportunisitic approach, one can consider all loops, as the loop name "IncidentCategorization" is none, that occurs as or relates to a loop function. The automation engineer/designer can then discard the loops, as they are not applicable for this task. A pessimistic approach would not even investigate the loop functions if names do not match.
3. Index 3 "Analyze": add all machine capabilities in the "Analyze" category to the candidate list. These are general analysis methods.

The number of methods in the candidate set will still be large, but at least the structure offers a clear reduction from the set of all methods and a priorization from more specific methods for a task to more general. With the actual task name (and much background knowledge) the automation engineer can

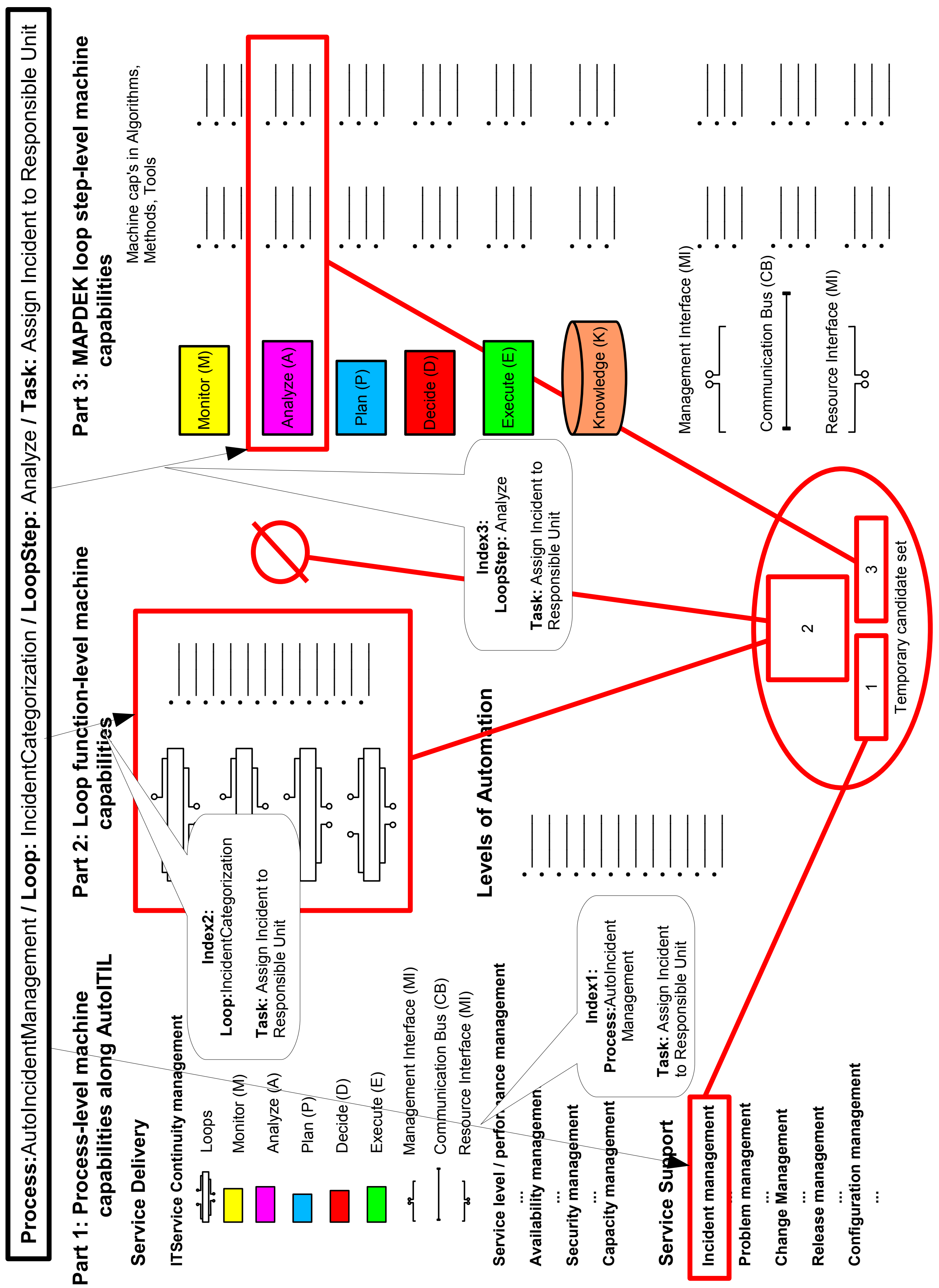

Process:AutoIncidentManagement / Loop: IncidentCategorization / LoopStep: Analyze / Task: Assign Incident to Responsible Unit
Part 1: Process-level machine capabilities along AutoITIL
Service Delivery
ITService Continuity management
Loops
Monitor (M)
Analyze (A)
Plan (P)
Decide (D)
Execute (E)
Management Interface (MI)
Communication Bus (CB)
Resource Interface (MI)
Service level / performance management
Availability management
Security management
Capacity management
Service Support
Incident management
Problem management
Change Management
Release management
Configuration management
Part 2: Loop function-level machine capabilities
Levels of Automation
Part 3: MAPDEK loop step-level machine capabilities
Machine cap's in Algorithms, Methods, Tools
Monitor (M)
Analyze (A)
Plan (P)
Decide (D)
Execute (E)
Knowledge (K)
Management Interface (MI)
Communication Bus (CB)
Resource Interface (MI)
Index1: Process:AutoIncident Management Task: Assign Incident to Responsible Unit
Index2: Loop:IncidentCategorization Task: Assign Incident to Responsible Unit
Index3: LoopStep: Analyze Task: Assign Incident to Responsible Unit
1
2
3
Temporary candidate set

Figure 6.2: Creation of a temporary candidate set of machine capabilities for a specific task

find out, that e.g. a classifier (a known concept in AI) is very suitable for this task. To know, how to design a classifier in turn will require to refer to AI literature. In cases, where knowledgable developers are available, the same task can be delegated to software development: “Build a classifier for incident classification into a list of categories.” This way, the large automation problem can be split into smaller software modules. This way, IT management can make use of much domain theory extrinsic to IT management so far, such as statistics and AI. Access into these domains is provided by keywords.

Idea 3: Describe machine capabilities at software development’s level of abstraction
The importance of an appropriate level of abstraction when describing machine capabilities can be seen in the following example. At the lowest level of abstraction, the machines (servers) used as basis for management automation systems in IT management, have only three basic capabilities: computation, data storage and communication. But if we described machine capabilities at this level of abstraction, it would be hard to match them with tasks in IT management automation. We thus need concepts at a higher level of abstraction, and therefore have a look at software development. For the catalog of machine capabilities to actually be useful for system design and development, we need to identify how management automation systems are built from other systems, components and theory, to know at what level of abstraction we need to identify machine capabilities.

Fig. 6.3 shows the steps of software development, that happen after function allocation. In this figure, an automation engineer gives advice on feasible function allocation which breaks down the functions into human-performed functions and machine functions. The human-performed functions serve as input for human training and user documentation but are of no further immediate interest within EIMA.

The machine functions—together with most of the input originating from the original automation scenario—form a system requirements specification as input to system design and development. In the figure, we show two alternate ways of software development: (1) “classical” software development and (2) model-driven software development.

The first one (1) is based on a conceptual system design phase, that creates a system architecture along architectural design patterns and decomposes the system to its necessary functions. As today, there’s hardly any software that does not use external libraries and tools, already a software architect considers how to reuse such existing components. Then in an implementation/coding phase, developers take care of individual system components and transform module of function specifications into code. They use design patterns at a lower level of abstraction, also reuse components such as tools and libraries and use algorithms to cover functionality.

The alternate way (2) of model-driven software development uses the approach to model the system in a platform-independent way first. A system modeler can reuse existing models, such as architectural models, behavioral models, or existing functionality. Occasionally, model-driven tools for analysis, simulation and testing are applied on the models to improve them with respect to their requirements. When needed, a code generator translates the model into a general purpose language.

Finally, the code is compiled to machine code or interpreted on the target system. This way, executable management logic is created, that relies on software sensors and actors, most of them being to send and receive messages. Hardware is not considered in EIMA, we simply assume that it is there, reasonably reliable and delivers the needed performance.

Among the two alternate ways of software development, EIMA is essentially model-based. In fact, processes, loops, tasks are all models. However, we do not know of any machine capabilities described and readily usable as models, e.g. a machine-interpretable model for a load balancer, a scheduler, or an SNMP sensor/actor. So, for the lack of modeled machine capabilities, we need to use the approach of classical software development, as the artifacts serving as input there, do exist.

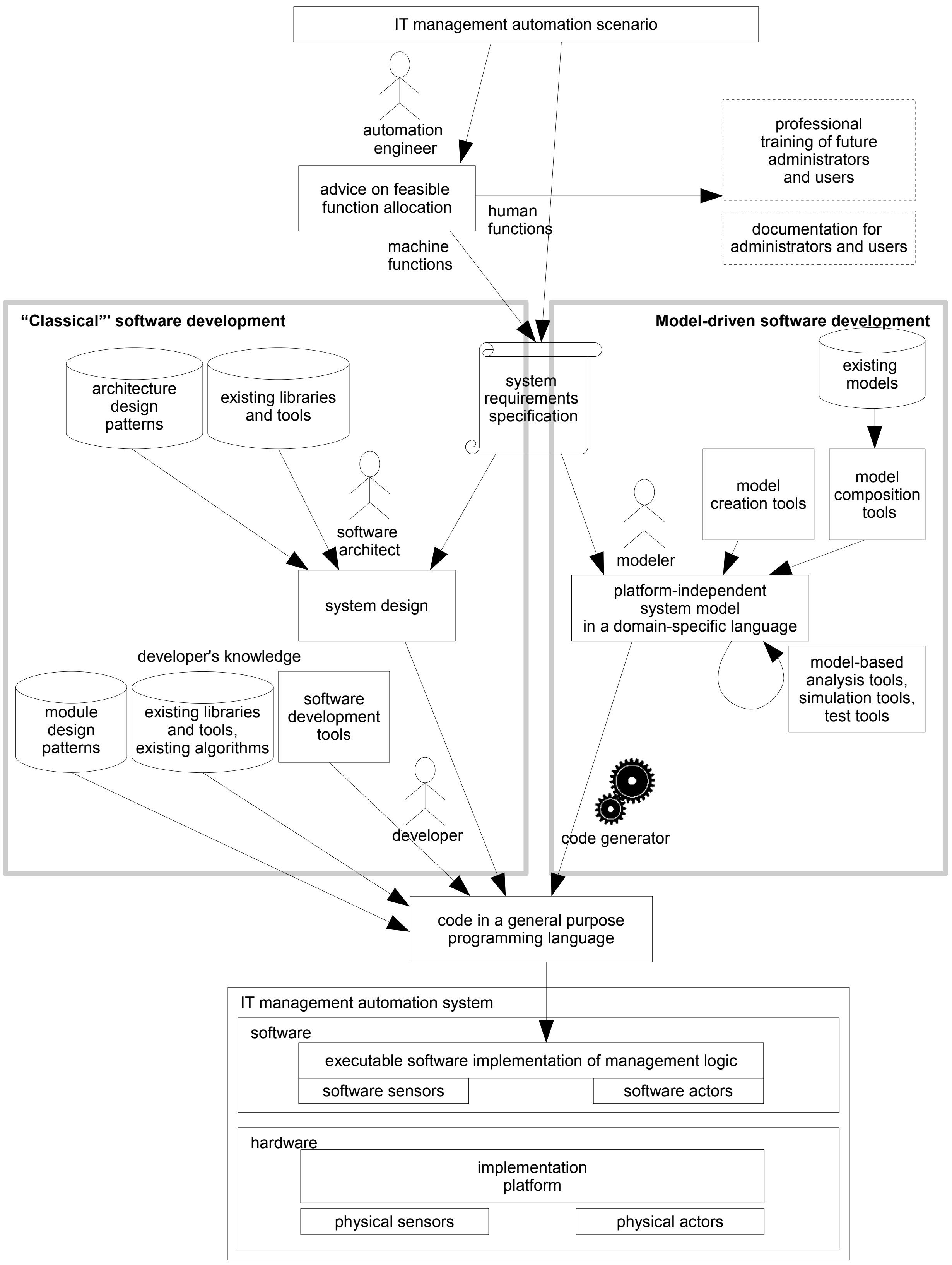

IT management automation scenario
automation engineer
advice on feasible function allocation
human functions
machine functions
professional training of future administrators and users
documentation for administrators and users
"Classical"' software development
Model-driven software development
architecture design patterns
existing libraries and tools
system requirements specification
existing models
software architect
model creation tools
model composition tools
modeler
system design
platform-independent system model in a domain-specific language
developer's knowledge
module design patterns
existing libraries and tools, existing algorithms
software development tools
model-based analysis tools, simulation tools, test tools
developer
code generator
code in a general purpose programming language
IT management automation system
software
executable software implementation of management logic
software sensors
software actors
hardware
implementation platform
physical sensors
physical actors

Figure 6.3: Function allocation is followed by software development. In EIMA we want to anticipate this phase and give hints on implementation as a result of functions being allocated to machines. In other words: A machine function is not included as a wish, but is accompanied by a proposed machine capability candidate as an inspiration to the software development staff. They are free of course, to look for other implementations, but at least they have a candidate for implementation.

Coming back to the question, at what level of abstraction to describe machine capabilities: software development uses knowledge at the following levels of abstraction: algorithm, library, tool, as well as whole existing systems. In fact, monolithic software development is very rare, instead it typically relies on existing components. Existing algorithms are implemented, existing libraries are either linked to statically or dynamically, existing tools are packaged and executed, and in distributed systems, remote components are coupled loosely via middleware.

We can assume, that (especially in the world of open-source software) software development looks for the most directly usable entity first (an existing management automation system), and then follows down the decomposition path of software. So first, existing systems are examined whether they provide the necessary functionality, then individual tools, then libraries, finally algorithms and design patterns typically published in books or academic papers. The rationale behind this approach is to benefit from existing software to a great extent and to limit necessary reimplementation work.

Looking back at Fig. 6.1, we thus avoid that for tasks allocated to machines there is not enough advice on implementation and the task being reallocated to humans. In Fig. 6.3, we see that this way with the machine capabilities software architects and software developers are supported with at least an idea on how to implement the function.

6.2 Requirements and non-requirements

The general goal of EIMA is to quickly identify automation potential in a given IT management scenario. This can be done in two ways:

1) “Top-down”: Starting with tasks looking for machine capabilities. For a certain list of tasks, we would ideally look them up in a structured knowledge base (the catalog of machine capabilities) to find machine capabilities that would enable assistance or automation of these tasks.

2) “Bottom-up”: Starting with machine capabilities, looking for tasks. This direction is especially useful, when for example progress in certain machine capabilities has been made, and there are now ways to use a method that before was not feasible in terms of resource use, slow algorithms, or quality of results. The introduction of host virtualization software in data centers of IT service providers is such an example with a major impact on IT management automation.

To achieve this, we need a catalog with the following features:

- The catalog should be structured along the EIMA approach (see Fig. 6.4). After EIMA task analysis and loop composition, we know the tasks nature in terms of AutoITIL processes, loops, loop steps (MAPDEK). These are orthogonal and can be composed. As for large IT management automation systems, such abstract entities play a central role in the communication between system operators and systems designers, also the catalog encourages alignment of the abstract system architecture to these common building blocks. For the actual implementation of the machine capabilities, it still leaves enough room for customization.
- The catalog should have a scheme, that allows to include machine capabilities at multiple levels of abstraction: algorithms from theory, methods from AI, software libraries, IT administration tools, general tools. Such a scheme would encourage reuse and cross-disciplinary knowledge transfer. In most cases, this also means better maturity. Today, new implementation methods are used far later than they are invented. Instead, new implementations are created far too often even though they are not better than existing ones.
- The catalog should separate management automation knowledge and interfacing to resources.

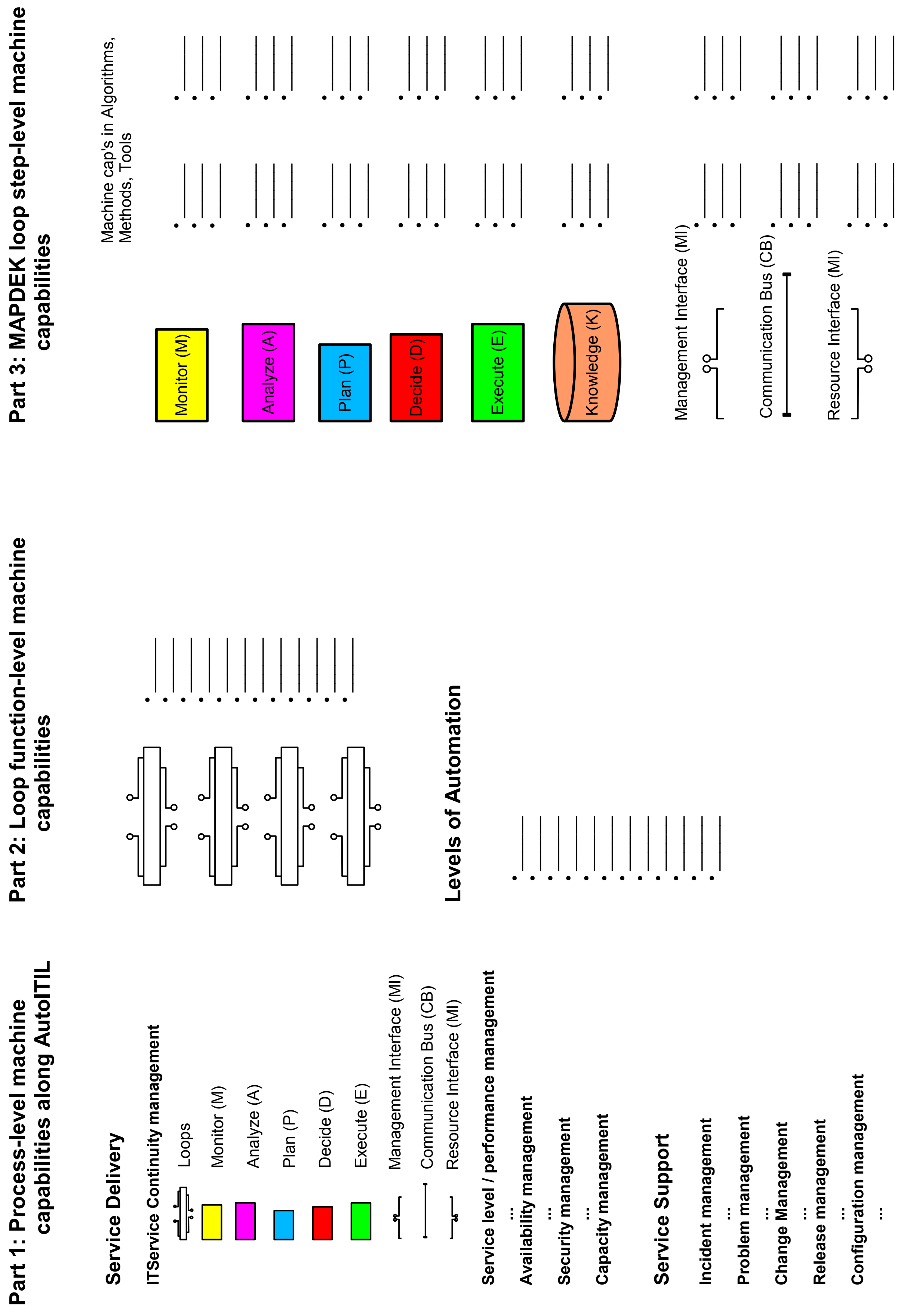


Figure 6.4: Structure of the catalog of machine capabilities

For example a pattern matching application (management automation knowledge) that does work in the dissection of configuration files for a certain program (interfacing) will likely also work for other configuration files. So, it is the method that is important, and evt. the type of input (configuration files) not the configured program itself. By design, this has already been fulfilled by the MAPDEK loops. There MAPDEK covers the steps, the management interface and resource interface are independent from it.

The machine capabilities catalog however also comes with at least two non-features caused by the independence from certain scenarios and the vast range of methods and tools:

- As said in Ch. 5, EIMA cannot provide automated function allocation. For the reasons listed there, function allocation will remain dependent on the case, economic influences (e.g. wages, costs of automation, business size), availability of qualified personnel, hopes, expectations and fears by staff, and many more influences. The catalog should be understood as motivation to learn about the listed machine capabilities and to investigate them in one's own projects, it should not be understood to serve as a recipe book.
- The catalog makes no claim on completeness. Undeniably, a catalog at the level of abstraction demanded here can never be complete. Instead, it lists components and methods that can be found in automation systems in one or the other way. Many of the entries such as methods and algorithms are timeless, while entries on tools would need periodic updating to catch up with latest developments. A machine approach would use a knowledge base to keep the information maintained.
- The catalog cannot duplicate knowledge about the machine capabilities. Instead, it must be concise with short method names. There are many books on the individual functions, methods, algorithms, there is documentation on tools and software libraries. They cover all the details, but many of them hardly relate back to IT management automation. Therefore, all we need in EIMA is a forward reference to the machine capabilities, which should be investigated by an automation engineer during the mix of function allocation, prototype design, systems design. Once we know the names of the machine capabilities, we can easily look up the details, e.g. capabilities in application scenarios, elsewhere.

6.3 Proposal to use the catalog in EIMA

6.3.1 Applicability to IT management scenarios

To give the reader a better impression about the usage of the catalog: The intended target scenarios of EIMA are those at IT service providers, that are aligned to an ITSM process framework and also need IT management automation to be able to offer services with competitive prices and service quality to their customers. Example application scenarios include but are not limited to *management automation systems* in:

- grid computing / high-performance computing
- cloud computing
- large database installations
- large business applications
- web hosting providers
- network service providers
- essentially every IT solution that has grown to a certain size (automation potential in the large number of similar/equal resources)
- essentially every IT solution that is built

in large quantities (many similar/equal instances)

• and others.

In such scenarios, the automation potential results from a large number of equal entities (e.g. equal resources, equal jobs).

6.3.2 Step-by-step application of the catalog in function allocation of management automation systems

Input We assume, that the workflows in the automation project has been treated in the way, that EIMA proposes and that task analysis and loop composition have successfully been performed by an IT management automation engineer.

At this point, we have:

- results of task analysis
  - a task decomposition tree
  - a role hierarchy
  - an automation table
- results of loop composition
  - a hierarchy of MAPDEK loops, associated with AutoITIL categories (see Fig. 6.1)

Matching of machine capabilities For function allocation we need to know candidates for machine capabilities for the MAPDEK steps, loops, and processes. Here we can search through the catalog of machine capabilities in a top-down fashion:

1. **Examine the involved AutoITIL tasks for relevant machine capabilities at process level, and allocate loops or loop steps to machine capabilities!**

   Reason: Matching machine capabilities at AutoITIL process level will likely lead us to the most relevant management automation systems most quickly, as they already relate to the process to automate. Here, we can assume, that terms of that process are used, e.g. for tasks and roles. This means, that we have to use very little abstraction to use the machine capability.

   Example: If we look for machine capabilities in capacity management, and find a capacity management system with the desired functionality in terms of machine capabilities, we can assume that it provides a well matching solution to our problem. In a certain sense, this is a trivial case. It gets more likely, that a management automation system exists, if we do not expect too many machine capabilities. The more specific automation needs we have, the less likely it will be to find a matching application and we will have to investigate matters at a lower level of abstraction.

2. **Examine the loops' general function! Consider whether one of the listed loop functions in the catalog fulfils a certain task or parts of a task!**

3. Also for loops, consider the loop levels of automation (in Ch. 5) and look for automation potential by raising the level of automation for certain loops!

   Reason: Loops are at an intermediate level of abstraction. They have the nice property to express automation along the Sheridan levels, so that loops can be run automatically with less (open loops) or none (closed loops) necessary human intervention.

Example: Taking again a capacity management example, we may not have found a capacity management system that enables a certain function such a preview function for the capacity demand and supply in case of a certain decision on a capacity management policy (such as a policy on user quotas). In this case, an additional effects forecaster loop function would be needed, which exists in the loop functions list.

4. **Examine the loops' interior!**

   a) **Examine the MAPDEK steps (the actual functions) inside the loops for matching machine capabilities in the MAPDEK part of the catalog!**

   Reason: MAPDEK steps are the essential part of each loop. Monitoring and Execution are classical functions that are needed to interface to the managed resources or lower loops. In many cases, the machine capabilities here are quite obvious. The actual challenge is in finding machine capabilities for analysis, planning and decision.

   Example: A simple notifier for different kinds of incidents could be composed of: monitoring machine capability - an IMAP receiver to read e-mails with standardized incident notifications, analysis machine capability - a classifier that puts the mails into categories, planning machine capability - none needed, decision machine capability - a simple threshold mechanism, execution machine capability - an SMTP sender that sends a mail with the notification to the upper loop as an escalation or status information, and resets the internal message counter for that category.

   b) **Find machine capabilities to use at the *management interface* of the top-most MAPDEK loops that interface to human roles!**

   Reason: These interfaces/capabilities must be chosen in a way, so that future system administrators feel comfortable with them. Typical implementation here is threefold: an application programming interface (API) allows external upper machine managers. The interface for system administrators, graphical user interface (GUI) and command-line interface (CLI), can then be based on the API.

   Example: For a capacity management automation system, policies could be used to pass administrators' preferences to the system.

   c) **Find technologies to use at the *resource interface* of the bottom-level MAPDEK loops!**

   Reason: This interface transmits information back and forth to managed resources or lower loops. Of course, it must match the available management interfaces of the resources to be managed. If no touchpoint is used (a translator for the Monitoring and Execution steps), then the Monitoring and Execution steps directly interact with the resource interface and must "match" it in terms of protocol/interface.

   Example: A management loop that needs to manage network switches will need access to them. Based on the capabilities of the switches, either an SNMP library will suffice for SNMP access, or a software library that wraps a client for access to the e.g. Cisco IOS console via telnet or ssh.

   d) **For each loop decide on one or more suitable levels of automation! See Sec. 5.5.4. "Dynamic function allocation"! This also has implications when investigating machine capabilities!**

   Reason: As shown before, there are often multiple levels of automation worth to be considered and let administrators choose at operations time.

Example: When configuring switches, a management automation system may fail because it cannot get access to the switch because of misconfigured authentication information. If we would now lower the level of automation and hand over the execution step to an administrator, then a management automation system could type out the commands to send, and the admin can use e.g. the a serial console interface of the switch (which may work without identification) to configure it. So, in this case a manual fallback actually exists. It could be used for matters of policy.

e) **Find a technology inside the MAPDEK loops to connect the MAPDEK components with an internal communication bus!**

   Reason: Of course, the MAPDEK components need to interact with each other and with the two interfaces of the loop. This needs a communication facility. For loose coupling the same machine capabilities can be used as for inter-loop communication (see communication bus). Coupling can also be achieved by means of inter-process communication (files, shared memory, etc.) or the knowledge base.

   Example: For the switch management automation system could internally be based on a relational database covering all the detected (or configured) network switches as one of the knowledge components. Communication with this database may be based on ODBC or JDBC, the MAPDEK components themselves could all be implemented in a single program and use the program language's own way of data exchange (method calls or function calls) for reasons of simplicity.

5. **Select a technology/technologies to connect the loops with a communication bus!**

   Reason: The loops will need to communicate with each other.

   Example: Web services can be used as middleware to connect distributed components in a management automation system.

In parallel during the search for machine capabilities the automation engineer can also use the role hierarchy to sort the roles, and to decide on implicit human and implicit machine roles and make them explicit. During function allocation, for each loop he can use the previously collected knowledge such as lines in automation table. Also in parallel, the machine capability catalog can be updated with additional new information input if discovered during this particular project.

Output We achieve a structure that attaches candidates of machine capabilities to the loops and loop steps, where applicable. From this basis, we can also derive an educated guess for black/white function allocation for "human", "machine", "human+machine", both at loop-step level and at loop level.

Postprocessing and reiteration This is essentially, what EIMA can achieve to this point. Now we have a rough estimate on a feasible way to allocate the functions. At this point the whole structure should be searched for opportunities of consolidation of loops, or another round of refactoring to drop unnecessary loops or add additional ones that may have been forgotten in the original workflows. The reiteration shall essentially improve system design with the goal of restructuring the problem and introducing more automation capabilities while at the same time paying attention to external constraints:

- detailed automation assessment
  - automation necessity assessment along task frequency, task response time
  - automation profitability assessment
    - finding wages for workers

    * finding actual implementations/code, and associated costs

  - laws and policies that decide function allocation
  - simulations of the system to see the overall orchestration. For this, typical flows through the loop hierarchy need to be defined, as well as task frequencies, durations, buffer sizes. With a queuing model, now we can size the individual loops to know for example how many instances we need to cope with the average request rate or peak rates.
  - include comments/wishes by future system administrators

- additional constraints/requirements
  - include considerations on safety, performance
  - include considerations of market research on product variability, features, costs
  - check against requirements of the automation project
  - include comments by system development on implementation

It should terminate based on the intuition and experience of the automation engineer, who must assess when no more automation potential can easily be identified. As said before, function allocation to machines is hardly a fully objective matter, so we have to rely on human judgment here.

To reduce human judgment and make matters more objective, simulations can be used. Once set up and parameterized, they would allow to quantify the automation effects. In such cases, there would be additional rounds of refactoring an simulating several alternative designs. The aim of the overall process would be to have an up and running model of the management automation system, where in the simulation we can see its effects. Only such a model is then talked about with development people on implementation.

6.3.3 Application example on "ChangeControl" workflow from Chapter 2

We used the methodology shown before to apply it to the running example in this thesis.

1. **Examine the involved AutoITIL tasks for relevant machine capabilities at process level, and allocate loops or loop steps to machine capabilities!**

   This example is a workflow that is associated mostly with (Auto) Change Management but also with communication to (Auto) Release Management. At this point, we can already say, that a change management system will generally accompany all the steps involved here. Its main purpose will be to serve as a central instance in documentation of the progress of the change requests. For the individual sub-tasks it will likely either contain functional modules or communicate with other management systems.

2. **Examine the loops' general function! Consider whether one of the listed loop functions in the catalog fulfils a certain task or parts of a task!**

   In the example we have only refined the task decomposition tree to two loops, an upper one with the request for change, and a lower one with change control. At this level of loop refinement, the loops have multiple functions, so that we will consider the loop functions for the individual tasks.

3. Also for loops, consider the loop levels of automation (in Ch. 5) and look for automation potential by raising the level of automation for certain loops!

   Levels of automation for dynamic function allocation in this example have already been considered in Ch. 5. There is an example on dynamic function allocation in Fig. 5.18 on page 188.

4. **Examine the loops' interior!**

   a) **Examine the MAPDEK steps (the actual functions) inside the loops for matching machine capabilities in the MAPDEK part of the catalog!**

   When doing so for this example, the result from function allocation in Ch 5 was used to get an overview of the machine roles. From the final function allocation matrix there, the part on machine roles was extracted and transposed. Then machine capabilities were added along the catalog. See Fig. 6.5 for the result.

   The selection of the machine capabilities was done based on the following reasons:

   Customer Change Management System: This system submits the change request based on locally available data. For this purpose it only needs to send a well-formatted change request to the lower loop.

   Request Processing System: We use a WS receiver to attach to the communication middleware. As two of the typical tasks in monitoring, WS has included mechanisms for source format validation as well as authentication, so we do not have to add such functions explicitly.

   Evaluator Assignment System: The overall idea of this step is to assign an incoming change request to one or multiple human or machine evaluators. Unfortunately the original example workflow made no statement on the "depth of cognitive processing" in this task. Therefore, multiple machine capabilities were listed, that start very simple with round-robin assignment, other scheduling and load balancing techniques. These would aim at a fair distribution of the incoming requests, but not actually matching requests and appropriate evaluators.

   A little more sophisticated and context-sensitive, pattern matching algorithms can make the assignment based on extracted keywords. We could also use statistical analysis or classifiers (from AI) at this place, that could use a training set of past assignments to learn how to distribute the incoming request well to evaluators. Finally, an expert system could be used to have experts model the rules, how to assign evaluators.

   As we can see, in this example, choosing well applicable machine capabilities can much depend on the scenario. What is important though is the knowledge, that there are machine capabilities that all could essentially do this task.

   Evaluation System: Again, we do not know how objective the evaluations are. If they are well formalized and based on numerical evaluations, then a utility function could be used to calculate a certain score for each change request. On the more sophisticated side, an expert system can be used for this type of ranking/scoring based on more criteria. As a minimum machine support tool though, a document management system should be used to keep track of the evaluation sheets, the results, versioning, authors and so on. This way, human evaluators can be off-loaded from the well automatable tasks and concentrate on their cognitive work: human evaluations.

   Modifier Assignment System: Here, the same line of thoughts applies as with the Evaluator Assignment System. Probably, the machine capability used there can even be reused.

   Project Planning System: Also with change planning, total automation is unlikely. So we choose planner assistance systems: project planning tools such as MS Project, ERP tools like SAP (if software development for change requests is modeled as a business process

System Component	(D) T01a Decide on Change Request	(E) T01b Send Change Request	(M) T01c Receive Change Request	(A) T02 Assign Evaluators	(A) T03 Evaluate Change Request	(P) T05 Assign Modifiers	(P) T06 Create Project Plan	(P) T07 Create Developm. Plan	(P) T08 Create Test Plan	(P) T09 Create Release Plan	(P) T10 Update RTM schedule	(D) T04 Decide on Change Request	(E) T11 Implement Change	(E) T12 Test Product	(E) T13 Release Product	**Applicable machine capabilities**
Customer Change Management System		R														(E) send message via (CB) between loops: use e.g. Web Services, use general purpose programming language to implement WS client, source of the change request data: web page, database, ticket system
Request Processing System			R													(M): use a Web Service for reception, e.g. in Apache Axis, basic validity checks are included (XML schema validator).
Evaluator Assignment System				R												(A): round-robin assignment, scheduler, load balancer, keyword-based assignment (pattern matching), statistical analysis, classifier, expert system
Evaluation System					R											(A): document management system, utility function, expert system
Modifier Assignment System						R										(A): see Evaluator Assignment System
Project Planning System							R									(P): project planning tools, ERP tools, planning algorithms
Development Planning System								R								see before
Test Planning System									R							see before
Release Planning System										R						see before
RTM Update System											R					(P): simple database update, evt. an additional notifier
Change Mgmt System	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	(D): change management system for general progress documentation. Decision on change request is a human activity, supported with e.g. effects forecaster, constraints checker.
Change Implementation System													R			(E): general software development tools, CASE tools, IDE, build tools
Product Test System														R		(E): test automation tool, unit testing, system testing
Product Release System															R	(E): automated updates routines, automatic software distribution, release management system

Legends: R = Responsible, I = Informed

Figure 6.5: Applicable machine capabilities for the example from Ch. 2, Workflow 2 “ChangeControl”

there). Only if the planning can be based on known models (task durations, involved people, capabilities) then a planning algorithm may actually be used to automate the planning work.

Development Planning System, Test Planning System, Release Planning System: These are all planning systems. We can reuse the choice made for the project planning system.

RTM Update System: When all the planning was performed, and is trusted to be correct, then updating the ready to manufacture date results from the plan. So it can easily be derived. A notifier can be used here to make humans aware of the change to the RTM date, who can then intervene.

Change Mgmt System: The Change Management System will overlook all activities in this loop and mainly document progress and involved staff. For T04 "Decide on change request" it could support the human decision maker however. It is unclear from the example, whether such additional support is desired.

Change Implementation System: Except for trivial changes, we're still far away from machines doing work in change request implementation, this means actually altering code or models. Therefore, machine capabilities at this point will support software developers with development tools, computer-aided software engineering (CASE) tools, IDE's and build tools.

Product Test System: Testing is a double-edged sword. Here, we apply test automation, unit tests and automated system tests to cover all the test cases, that can be modeled in a test suite and be run with no human intervention. The test results will then feed back to the software developers. However, testing will typically also involve human testers that are far less systematic but more creative in either invoking bugs or recognizing other pain points with a certain release candidate.

Product Release System: The foundation of this activity will be a release management system. For easier distribution and patching of existing installations, automated update routines (pull) or software distribution (push) can be used to speed up user adoption of new releases.

b) **Find machine capabilities to use at the *management interface* of the top-most MAPDEK loops that interface to human roles!**

   In this example, the upper loop is already existent, so that the change request and change done notification are the interface to communicate with the lower loop. The upper loop in this example is not considered here, as it is outside the scope of the automation engineer in this case (loop performed by requestor).

c) **Find technologies to use at the *resource interface* of the bottom-level MAPDEK loops!**

   This example does not directly deal with resource management, but with change control. In principle, we could imagine actors directly modifying the code of the application to change, but it is highly unlikely that this will succeed for all changes. So, here the lower loop is an open one, where software developers do the actual implementation of the change but are supported with assistance systems.

d) **For each loop decide on one or more suitable levels of automation! See Sec. 5.5.4. "Dynamic function allocation"! This also has implications when investigating machine capabilities!**

This step was skipped here for the reason, that we have no further detail on the example. We can imagine though, that sending and receiving messages will be automated, all analysis and planning steps will have at least one level of automation where a human can intervene plus an automated one, the execution tasks will be machine-assisted but hardly be fully automated.

e) **Find a technology inside the MAPDEK loops to connect the MAPDEK components with an internal communication bus!**

   We have much freedom here. As the loop steps all involve other potentially distributed systems, we could use a middleware such as Web services. Another feasible choice would be to used the change management system's data store as a common entity for knowledge exchange and a simple signaling protocol from a central master to start the individual loop steps in the correct order.

5. **Select a technology/technologies to connect the loops with a communication bus!**

   As this is a distributed system, all applicable middleware can essentially be used. We believe that web services are a good choice here for their general applicability and platform independence and human-interpretable messaging, even despite of its overhead.

6.3.4 Conclusion

The catalog of machine capabilities lists many tasks, methods, and tools in IT management automation. To use it, the type of AutoITIL process, loop, and loop step provide three indexes into the catalog. As could be seen in the example though, finding relevant machine capabilities and reasoning about their feasibility or superiority over other machine capabilities is not an objective matter. The automation engineer must use his knowledge on the individual capabilities to select from the lists those considered more appropriate for a certain task. The lists however give a welcome overview and are an inspiration for in-depth investigation of those capabilities yet unknown to a particular automation engineer.

6.4 Related work and comparison

When creating a catalog of machine capabilities in IT management automation, there is the risk to overlap with state of the art in (1) literature and (2) tools. Therefore, current related work on IT management related literature as well as tools has been examined to compare both and identify the novelties of the machine capability catalog in EIMA. Publications in systems engineering hardly had covered IT management automation as an application domain, so they were left out of this comparison with the exception of books on industrial automation, sharing the explicit view on automation.

6.4.1 Comparison with current literature

OGC: ITIL V2, 2000 and V3, 2007 The most obvious sources for automation of ITIL processes, ITIL V2 and V3, have only few references to machine capabilities that could help in IT management automation, instead they focus on listing organizational best practices in IT management.

Hegering, Integrated Management, 1999 This book investigates the technical foundations of existing management systems and concepts to integrate them. It describes management models and management protocols (OSI and Internet Management). It only slightly covers human IT management processes.

Review result: While it describes some concepts like Management by Delegation in OSI, it rather focuses on integration of management systems, instead of the automation of formerly human-driven processes itself. The integration of management systems relates to existing management architecture and tools, the book does not have a cross-domain view, that includes algorithms, methods from other domains such as AI, decision support systems, or systems engineering, and new assistant systems from the current point of view.

Burgess: Analytical Network and System Administration, 2004 Burgess manages to present many theoretical methods rooted in mathematics in combination with IT management related examples which he regards as human-computer systems. While he describes many basic concepts the chapters themselves are self-contained. Overall this book is absolutely recommended for an IT management automation engineer to just enough deepen the understanding for the individual methods but conceptually staying in the network and systems administration domain in terms of examples, and without turning to a theory book with all its proofs, lemmas and lack of application examples.

Review result: The book uses a theoretic bottom-up approach. It starts simple with basics in mathematics and logics, some statements have a philosophical focus instead of an engineering style approach. Then, it covers aspects common in IT management (e.g. task management, policies, system architectures, knowledge) and includes the human perspective in several places. It nicely covers the individual topics at an appropriate level of detail for an IT management automation engineer, but it lacks recommendations on combining the methods and building management automation systems along them. It also lacks alignment to e.g. system administrator's education in process frameworks.

Design Patterns in System Development (various sources) Design patterns in software engineering have been covered in a number of ways.

Foremost, in 2009 there was the 19th conference on "Pattern languages of programming" (PLOP) that covers patterns regularly and provides proceedings with new patterns, as well as critique to existing ones. The series of these proceedings essentially forms a large proposal for a pattern catalog, being reviewed and updated annually. There, each software pattern, described on several pages, is a contribution and is reviewed and improved. Clearly, this thesis is not intended to compete with it. Instead, EIMA intends to include an application-specific collection of machine capabilities to aid in function allocation of IT management tasks instead of a catalog of design patterns that each propose a solution for a certain software design problem in a general way.

There are also books on design patterns [GHJV99, FFBS04], which focus on object oriented programming in general, and not IT management automation. And there is literature on analysis patterns [Fow96], which however in this particular case was derived from derived from the health care and financial areas, areas that IT management automation does have big overlap with.

In contrast to design patterns already collected in pattern books, the four concepts enumerated in the EIMA approach all cover the specific aspects of IT management automation on the way to IT management automation systems. Information along these concepts should be included in system documentation of such systems to make system function and system management better distinguishable from each other. Once this information is added to user manuals, system administrators and system users can learn what abilities they give up, what they keep, and what they gain to manage or at least influence the system. Neither current design documents with implementation details of the complex systems, nor marketing documents/white papers currently serve this purpose.

Schnieder: Methoden der Automatisierung, 1999 This book is mostly theoretical with a view on modeling systems with Petri networks and state diagrams. It includes examples from traffic control, electrical engineering, manufacturing, along the approach: Take a simple enough problem, then apply modeling (with at most a couple of variables or states) to formalize the problem. The book stops, once the problem is more formal (expressed as a state machine or a Petri network).

The presented methods seem to be not abstract enough to recognize a relationship to IT management. The sought after methods could overall not be found. It is unlikely, that the named concepts (automata, Petri networks) at their level of abstraction would serve as good means of communication between IT system administrators and developers of IT management automation systems.

Love: Process Automation Handbook, 2006 This book describes process automation and uses chemical and process industries as application domains. It claims to be written to "bridge the gulf between theoreticians and practitioners in process automation in domains such as oil, petrochemicals, agrochemicals, fine and specialty chemicals, [...], nuclear processing, etc." Therefore, the sections on application domain-specific matters are non-relevant here.

The book does not contain a general automation method, so it cannot contribute to the approach. Instead, the value of this book for our work is in the sections on automation theory and techniques, where a mathematical yet well comprehensible way was found to present knowledge on control theory, automation techniques as well as general automation matters on acceptance, safety and simulation.

Lunze: Automatisierungstechnik, 2007 Lunze mainly considers the aspects of modeling and simulating automation systems. Many examples are shown, most of them small and comprehensible from a number of application domains from traffic engineering to production systems.

This book well shows the application of models suitable for automation as well as simulation systems. It lacks however automating cognitive tasks (as IT management is), instead it keeps at the level of continuous systems and time-discrete control systems in industrial settings, which are pre-planned and allow little flexibility at run-time. One sign for this position is that he does not include methods for analysis and planning both being cognitive tasks based on potentially unstructured data.

Academic papers Work on patterns in management automation/self-management is relatively rare. The majority of academic contributions presents new frameworks for specific cases along a "Using *method x* for *management task y* of *resources z*" approach. We could get an index from such accordingly structured paper titles, if a parser/classifier identified the three phrase tokens. The value of such an automated compilation would depend on the reference set of papers used and the quality of the automatic compilation. To show the general applicability of this approach however, dissertations from the research team of this thesis' author (the Munich Network Management Team) have been included in the catalog machine capabilities to show that it is able to categorize a broad set of academic work on particular solutions in IT management.

In contrast to the point solutions, there is work like [Wil04, PKGV06, Vas09], that indeed presents patterns on machine capabilities in IT management automation systems. The first two describe patterns along the loop concept which is used the DASADA loop model. The third one defines an "Autonomic System Specification Language" which is intended to be used in formal specification, validation, and code generation for autonomic systems. The author uses it in research on NASA swarm applications [HV08], which again brings back the aspect, that management automation / self-management seems to be more desired in domains, where external communication and control would also suffer from predictable temporary technical limitations.

Conclusion The cited works all relate to the topic, but none of them puts its focus on a good overview on machine capabilities applicable to IT management automation. Most of them lack associations to IT management (process) frameworks (with ITIL as an exception, of course), that would give us a first category to categorize their applicability in this domain. If they describe IT management, then they lack a focus on building management automation systems, if they cover automation, then they do so at a lower level of abstraction suitable for process control in industrial automation. None of them bridges machine-level capabilities with applications in IT management automation including a systems engineering approach (task analysis, loop formulation, function allocation) and human-factors related influences. They are, however, all good additional reading for concepts related to EIMA in various directions.

6.4.2 Comparison with current IT management tools

A catalog of machine capabilities in IT management automation cannot only be seen from the literature point of view, but also the point of view of available IT management tools. In fact, if there was a software suite, that would cover IT management automation at the EIMA level of abstraction, then machine capabilities would be built into it and usable as building blocks. In effect, such a tool would also serve as a catalog of machine capabilities. It would even serve as one that actually contains *models* usable in modeling IT management automation systems.

For this purpose, available IT management automation tools were reviewed and contrasted to EIMA's approach of a catalog of machine capabilities and its desired properties.

ITSM tools Current ITSM tools are mainly intended for use in process specification, process documentation and as groupware standardizing communication between *people*. With workflows and forms, they aim to ease human processes, databases for items like incidents and changes simplify information access between multiple processes to the same data, or between linked items (such as a release linking to change requests which link to a problem linking to a number of incidents). Clearly, automation is one aspect of such tools, and standardizing and automating human communications may have well measurable economic effects, but technical automation is not the main aspect of these tools.

Service management and resource management are still seen as split worlds, as service management software rarely involve managing resources and vice versa. Large vendors, such as IBM, do work on integrating these two and involve their own offered resource management systems and resources, which however also comes along with a certain vendor lock-in. A critical point in such tools is their specification of the workflows and processes. Customers would ideally like reference processes, that are easy to certify against the standard, but at the same time keep the flexibility to change them or design their own from scratch.

Resource management (automation) tools Commercial resource automation tools like

- IBM Tivoli Suite (Systems Automation, Enterprise Workload Manager, ...)
- Microsoft System Center, before: Microsoft Operations Manager
- HP Server Automation Software, HP Network Automation Software, HP Client Automation (previously Opsware)
- Ciscoworks LAN Management Solution

are designed to create automation products for a special purpose: automating particular management tasks for particular resources. With their resource orientation they typically cover parts of multiple of the processes in IT process frameworks, to which they are not aligned however. On the positive side the vendors do task analysis (evt. also loop formulation) and function allocation, but the general aim is not on creating flexible automation products with levels of automation and dynamic function allocation, but rather to tackle most of the standard tasks in a vendor-defined manner.

Custom low-level automation with small tools and scripts Another approach on (mostly systems) management automation, especially for small environments, is to use small and simple "on-board" means such as typical command-line tools supplied with an operating system and to compose them masterfully via self-made scripts.

UNIX-like operating systems typically include command line tools, such as: make, cron, mail, find, cut, cat, grep, awk, sed, date, and by using a shell and its full capabilities, custom scripts can be created quickly and flexibly for many kinds of every-day tasks. The administrators value, that they have made the automation themselves, and so are likely to understand, maintain and change it. In addition, their reliance on low-level standard tools means less dependencies on complex management frameworks which themselves can be the source of errors, low performance, and unnecessary complexity.

Doing so, and by writing the scripts themselves they also keep trained in these low-level tools, so that even if portions of their home-built automation fail they will likely know how to fix it or do it manually, until they have fixed their custom automation routines themselves. The more such custom-made solutions grow, there are potentials like being customized very well for a particular task and environment, but also pitfalls as essentially all the knowledge on these routines is with specific administrators, making them not subject to notice, and who may have spent years to set it up in the way it is, which can be regarded as low efficiency.

On the negative side, they rarely take the point of view of their employer, who would aim at standardization, certifiability, managing more IT equipment/services with less administrators, documentation of tools and processes, generally available knowledge and a certain flexibility in staffing.

Conclusion Current tools in IT management all have a solution character that covers one “layer” in IT management instead of a cross-layer character from algorithms/methods to IT management processes. The individual tools at IT service management, IT resource management, and low-level tools tackle their particular topic, but those at service management level have limited interaction with resource management and command-line tools and vice versa.

Resource-level and lower-level management tools are not aligned to process frameworks and have limited capabilities in explicitly assigning human responsibility and accountability, which makes it hard to estimate how they will be regarded in process certifications. Higher-level tools make internal assumptions on automation levels and give limited advice on the use of automation, its defects and dynamically taking-over.

6.5 Catalog of Machine Capabilities Part 1: AutoITIL process level machine capabilities

The catalog of machine capabilities is divided into three parts as stated in the introduction of this chapter. The main structure is provided as stated in Idea 2 at the beginning of this chapter: AutoITIL processes, loops, MAPDEK loop steps. In the same way as tasks have ID’s, the catalog is organized. Both structures match to speed up finding a set of appropriate candidate technologies for a certain task.

Part 1: AutoITIL process level machine capabilities This part of the catalog lists machine capabilities in the AutoITIL processes. The compilation is based on the experience of the author of this thesis and has been collected and compiled from personal experience, interviews with data-center administrators, conference proceedings and talks, projects, available literature and web sources. Listing all these sources would have had no additional use to the reader and is therefore omitted.

The catalog does not make any claim to be complete. However, it was taken attention to the property, that all methods are current and that there are automation products, algorithms, software tools, that indeed implement the machine capability. No claim is made on maturity of the tools, even though it was tried to list mature ones that are widely accepted and used. Open source tools was given a preference for their good suitability for research, improvement, integration, and availability.

The intention of this list is to show the enormous list of tasks, where in AutoITIL processes automation tools can be applied going beyond simple workflow tools, that only structure and order human communications with forms and databases. Instead, special attention was put on including the automation of cognitive tasks in IT management, such as analysis and planning.

The vision of EIMA is, that along the lines of these machine capabilities, ITIL will get an AutoITIL layer of machine capabilities off-loading work from humans to machines (see Fig. 6.6). The original ITIL specification documents regard ITIL as a framework to regulate *human* organization and procedure, only in very few places (such as the CMDB) they explicitly mention tools to be used.

In contrast, management automation systems along AutoITIL are to be seen not only as a communication and data store layer for ITIL, but as an assistant layer. Like a pilot can hand over certain tasks to an autopilot, AutoITIL management automation systems shall take over human tasks. In an implementation, AutoITIL management automation systems would get the same names as the ITIL human roles, and they would essentially work along the same procedures. This is to ensure, that humans, who understand ITIL human processes also understand the overall organisation of AutoITIL machine processes. When the AutoITIL automation fails in some point, the human can take over and do a job, that he would have done without AutoITIL anyways.

As with pilots practicing their hand-flying skills regularly by deliberately disabling/not enabling autopilot, IT administrators should stay trained in doing their tasks without AutoITIL management automation systems where possible and appropriate. AutoITIL can still help them learn the processes, have proposals for intermediate results (such as plans or calculations). The main objectives of AutoITIL are more effectiveness (meaning less downtime, better service levels) paired with better efficiency (less money/time spent on management to achieve this higher effectiveness mentioned before).

Abbreviations In the descriptions, we use the following abbreviations for the parts in a MAPDEK loop.

- MI: management interface of the loop
- M: Monitoring component
- A: Analysis component
- P: Planning component
- D: Decision component
- E: Execution component
- K: Knowledge component
- CB: Communication Bus
- RI: Resource Interface
- L: Loop

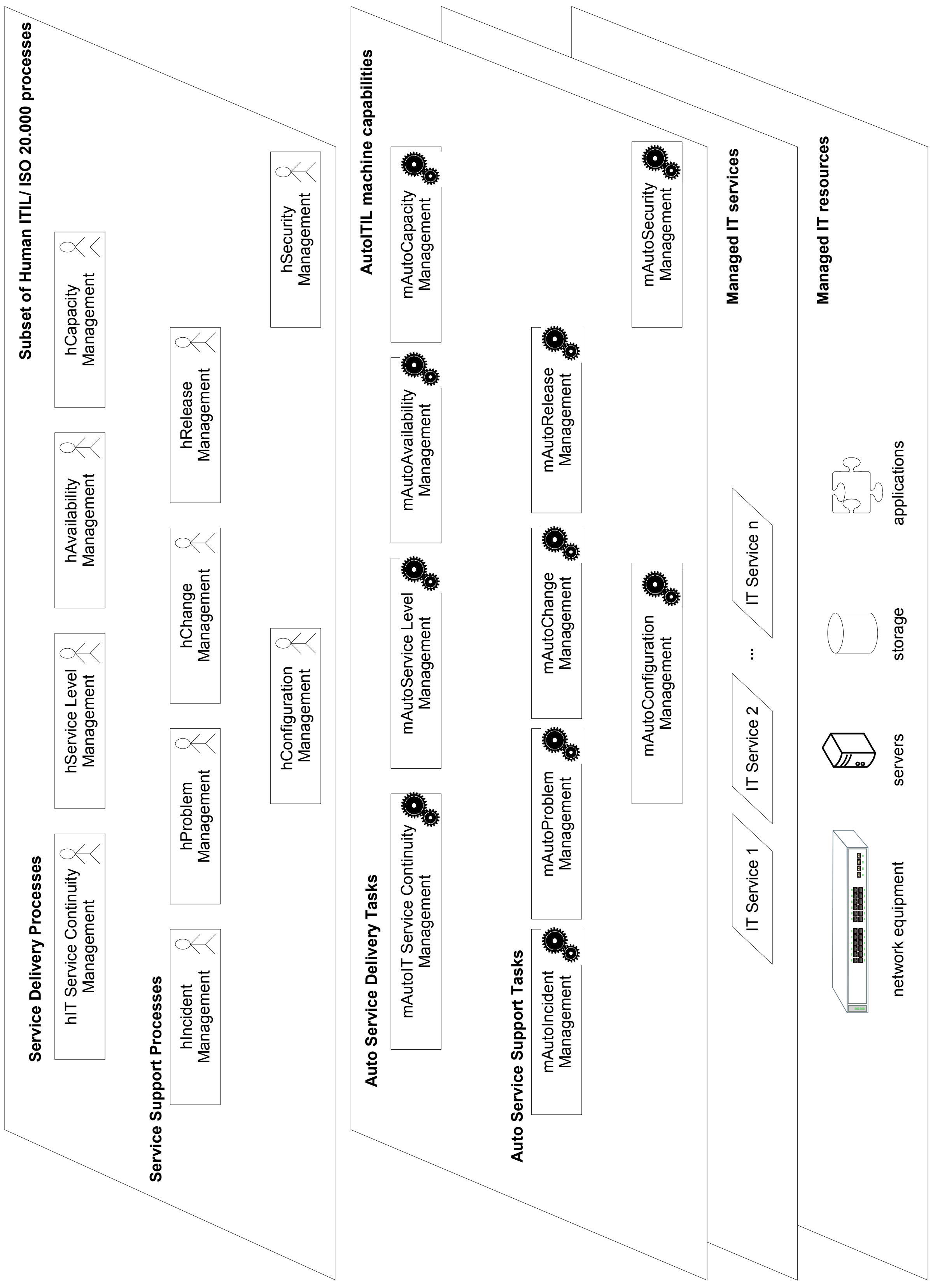


Figure 6.6: Cooperation of ITIL and AutoITIL

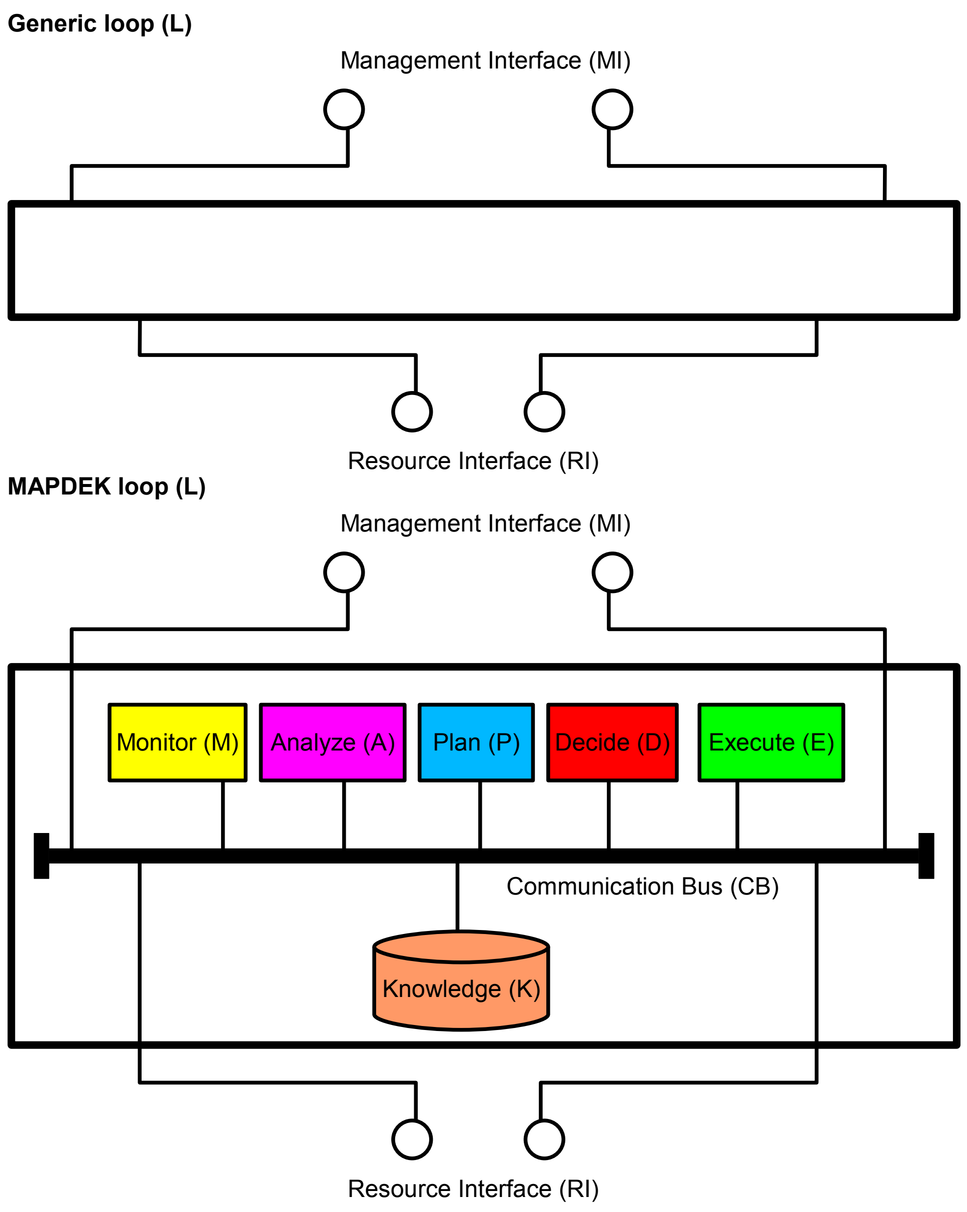
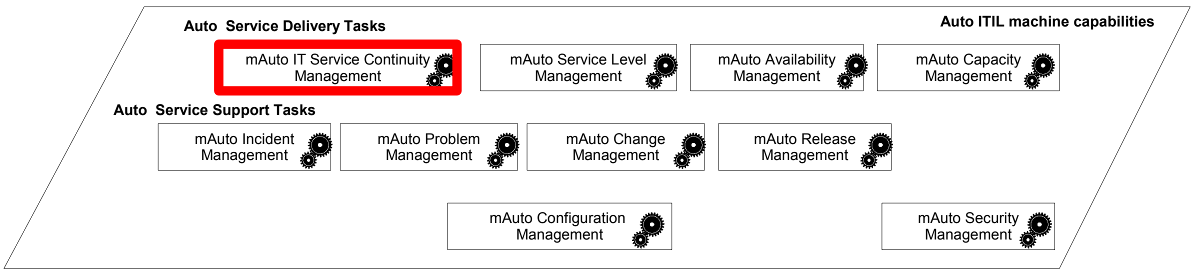


Figure 6.7: Legends for generic loop and MAPDEK loop

6.5.1 Machine capabilities in Auto IT Service Continuity management


Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

Common

- MI: downtime limits/penalties in service level agreements
- MI: ready-made policies for legal requirements
- A: risk analysis / business impact analysis
- K: resources, dependencies
- P: service/disaster recovery planning

IT Service Continuity management for services

- L: checkpointing and replay
  - RI: VM snapshots
  - RI: application checkpointing mechanisms
- RI: remote sites for service migration

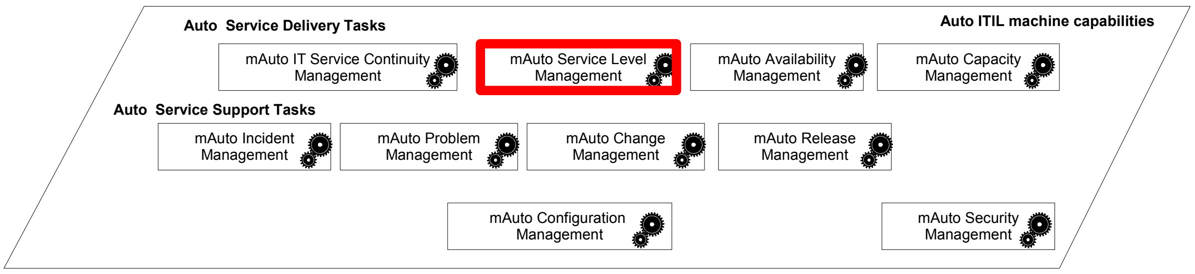
IT Service Continuity management for data

- L: backup
  - MI: backup policies, frequencies, expiration dates, size limits
  - MI: support for regulations and laws, policies (there should be prebuilt backup plans for existing standards, such as Sarbanes Oxley Act, ITIL V3, regulation on data protection and archival)
  - A: expiration checks
  - A: data comparison
  - P: back-out planning for failing backups
  - P: resource planning for storage and network (also capacity management)
  - P: backup prioritization, scheduling
  - E: auto-verify
  - E: compression
  - E: automatic tape libraries
  - K: backup calendar, log
  - RI: file system
- L: restore
  - MI: user-self service portal with authentication

  - MI: special virtual sub-directories in user folders “Yesterday”, “Last week”
  - D: overwrite existing files?
  - E: restore access rights
  - RI: file system

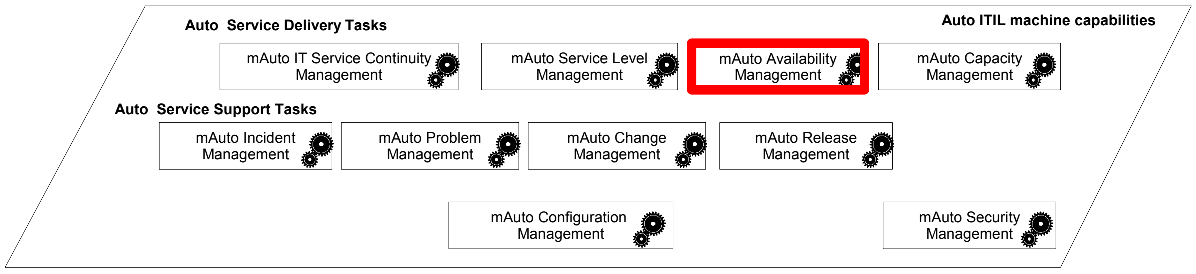
- L: examples: Apple Time Machine, ADSM / Tivoli Storage Manager
- L: checkpointing and replay
  - RI: file system snapshots
- RI: remote storage for data disaster recovery

6.5.2 Machine capabilities in Auto Service level / performance management


Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

- MI: service level / performance requirements/goals/penalties in service level agreements
- MI, K: SLM modeling, see [Sch08a]
- M: customer self-service for ordering and to configure a service with an SLA
- M: performance/service monitoring (local/remote, end to end see [Yam09])
- A: statistics
  - performance/service level trend analysis
  - heuristics
  - SLA aggregation, breakdown
- A: optimization
  - genetic algorithms (find high-performance configuration)
  - bin packing (reduce resource fragmentation)
  - constraint solvers
  - load-balancing (min-max-optimization)
- A: SLA checking/verification
- P: automated resource (re-)allocation
- D: priority decision among jobs competing for resources
  - `nice` levels
- E, MI: performance/service level reports
- K: service catalog management
- improvement cycle: using past real SLA measurements to specify new SLAs for new customers
- issues:
  - automation overhead may impact raw performance

6.5.3 Machine capabilities in Auto Availability management


Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

Common

- MI: availability requirements/goals/penalties in service level agreements
- A: component failure impact analysis
  - tasks
    - ∗ at design time: with planned failure probabilities
    - ∗ redo at run-time with measured failure rates
    - ∗ compare with service level agreements
  - fault tree analysis (FTA)
  - failure modes and effect analysis (FMEA)

Hardware

- L: error detection and correction (fault-tolerance)
  - L: parity checks
    - ∗ simple, but weak protection
    - ∗ ECC RAM
  - L: Forward Error Correction (FEC)
    - ∗ Reed-Solomon-Codes used on DVD, CD, DSL, DVB, WiMAX
- L: high-availability by reasonable redundancy
  - L: fail-over configurations
    - ∗ cold-standby
    - ∗ hot-standby
    - ∗ RAID (redundant array of inexpensive disks)
    - ∗ tandem configurations of servers
    - ∗ heartbeat
  - L: Cyclic Redundancy Check (CRC)
    - ∗ simple XOR and repetition codes, are computationally efficient
    - ∗ high cost in terms of bandwidth (ongoing cost, when paying for traffic)
    - ∗ relatively weak in terms of error protection
    - ∗ example: blocks on hard drive
  - L: redundant hardware components (mask hardware failures)
    - ∗ power supplies, fans
    - ∗ network interface cards
    - ∗ CPU, memory
    - ∗ ideally: hot-swappable to reduce downtime for reboots and power cycling

- L: network failover
  - spanning-tree protocol in switched networks
    - Rapid STP (IEEE 802.1w)
  - gateway (would be single point of failure!)
    - Cisco Hot Standby Router Protocol (HSRP)
    - Virtual Router Redundancy Protocol (VRRP)
  - dynamic routing protocols in routed networks
    - OSPF (in Autonomous Systems), BGP (across Autonomous Systems)
  - Layer 4/7-switches with load distribution strategy
- M/E: multiple booting and start-up configurations
  - work around misconfigurations: one minimal start configuration to start loader, then loader to select from multiple start configurations
  - typical examples: mainboards with dual bios network elements with multiple nvram configurations
- reasonable overprovisioning
  - add reasonable amount of spare resources/redundancy to the device (more RAM, more capable processor, multiple communication interfaces, more storage) to tolerate temporary load peaks and prepare for growth (see also capacity management)

Software

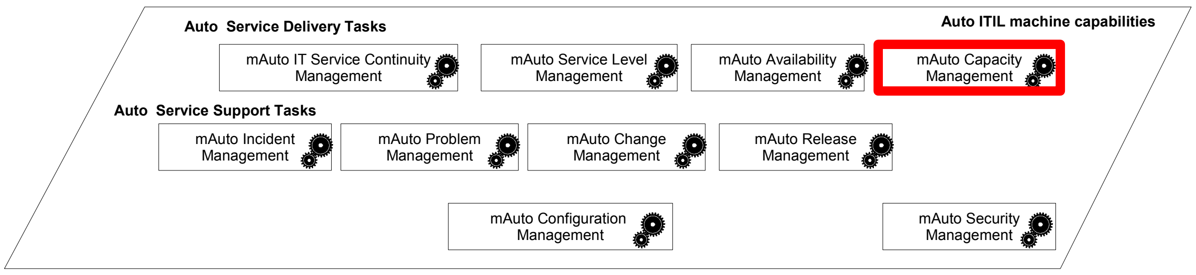
- L: error detection and correction (fault-tolerance)
  - L: Cyclic Redundancy Check (CRC)
    - example: packed archive files
  - L: Forward Error Correction (FEC)
    - "self-healing" storage
      - GridBlocks DISK `http://gridblocks.hip.fi/` (Distributed Inexpensive Storage with K-availability)
      - BitMountain `www.fht.byu.edu/prev_workshops/workshop06/abstracts/6-Hathaway-BitMountain.pdf` `http://hathawaymix.org/Writings/BitMountainPaper.doc`
    - properties
      - computationally more expensive
      - one-time cost for more CPU power necessary
      - ongoing cost in terms of higher energy consumption
      - usually much better error protection
      - reliability computations
- L: self-restart/rejuvenation (intentional short life of objects)
  - threads
  - forked processes
  - micro-reboots
    - "Contracts" in Sun Solaris
  - planned reboots
  - UC Berkeley/Stanford: Recovery-Oriented Computing (ROC)
- L: fail-over configurations

  - heartbeat
  - VM high availability by migrations

- L: high-availability by reasonable redundancy
  - L: distributed file system
    * Andrew File System (AFS)
    * Google FS
    * Hadoop FS

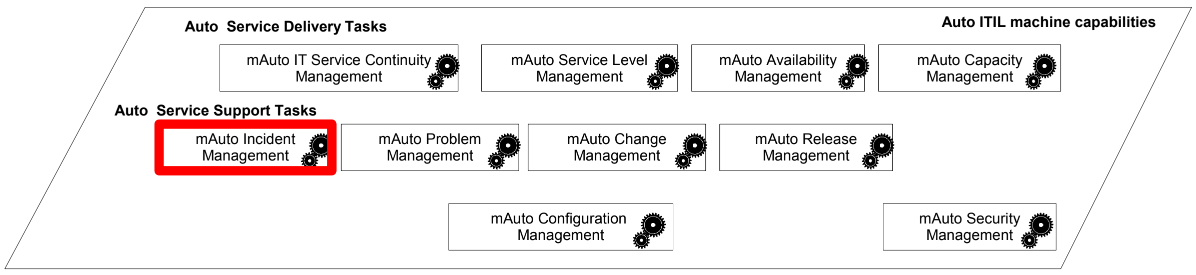
- L: network failover
  - round-robin DNS

6.5.4 Machine capabilities in Auto Capacity management


Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

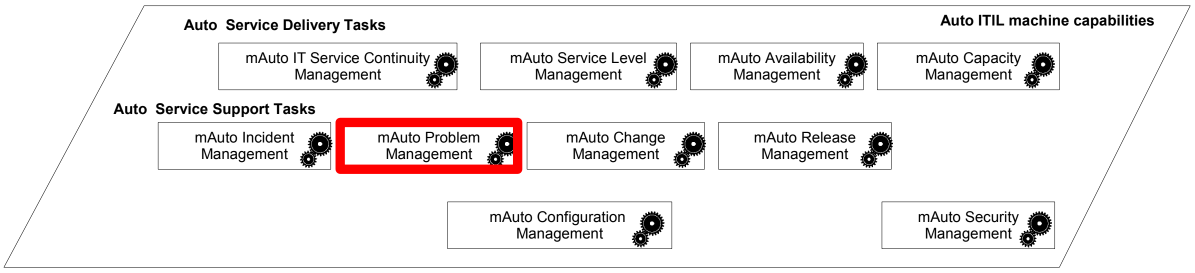
- M, E, RI: server capacity management
  - partitions of CPU/memory by VMM
  - partitioning of multi-core CPUs by assignment of cores
  - scheduler priority change tools / `nice` levels
- M, E, RI: storage capacity management
  - SAN storage
    - flexible extension
    - partitioning by LUNs
  - quota mechanisms (e.g. file system)
  - deduplication
  - archival to other media (e.g. tapes)
  - compression
- M, E, RI: network capacity management
  - QoS mechanisms like IntServ, DiffServ, RSVP, see [Yam09]
  - VLAN priorities
  - MPLS
- L: traffic policing, traffic shaping, prioritization
- A: capacity / demand trend analysis
- P: capacity planning, demand planning
- overprovisioning
  - add reasonable amount of spare resources/redundancy to the device (more RAM, more capable processor, multiple communication interfaces, more storage) to be prepared for future changes, and to reduce procurement and provisioning cycles

6.5.5 Machine capabilities in Auto Incident management


Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

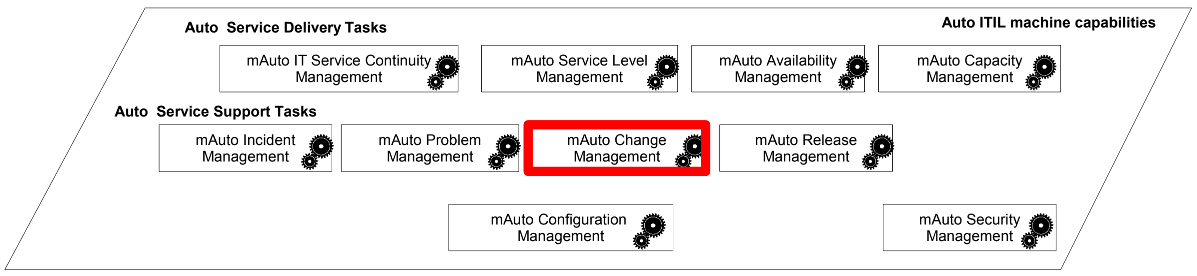
- M/E: phone, mail, chat (input by users/customers)
  - service desk, computer-assisted telephony, call-center equipment
- RI: polling systems/service monitoring systems (input from machines)
- A: incident classifiers to sort into categories, e.g. tree-based voice dialog systems
- A: statistics, e.g. aggregation, thresholds, see “Analyze” for more
- A: intelligent trouble-shooting assistants (decision tree-based, classifier)
- A: pattern matching, nearest neighbor
  - reasoning, whether the incident is known or not
  - if known, then look for solution
  - if it exists (known error), apply
- E: active probing (polling suspect hardware)
- K: known error database
- K: for better finding related incidents: synonym database, keywords, case-based reasoning
- L: user remote probing (user can probe own equipment)
- L: automated error reporting
  - e.g. by Windows, Firefox, KDE, HP OpenView
  - include context, such as: version, configuration, date, time, ID’s, stack trace, dumps
  - then analyze collectively, find clusters, forward to development
  - evt. give user a hint on new drivers or patches
- L: customer feedback tools (for improvement notes and comments, that are not incidents), e.g. in MS Office
- L: user self-help portal
  - M: service/resource monitoring access for users/customers
  - M: trouble indication for the public or customers, e.g. via WWW
  - A: incident statistics and log
  - K: frequently asked questions (FAQ)
  - L: password reset
- CB: groupware for communications support
  - workflow management systems, mainly structure human communication with forms
  - trouble-ticket systems, e.g. Open Ticket Request System (OTRS), Remedy ARS

6.5.6 Machine capabilities in Auto Problem management


Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

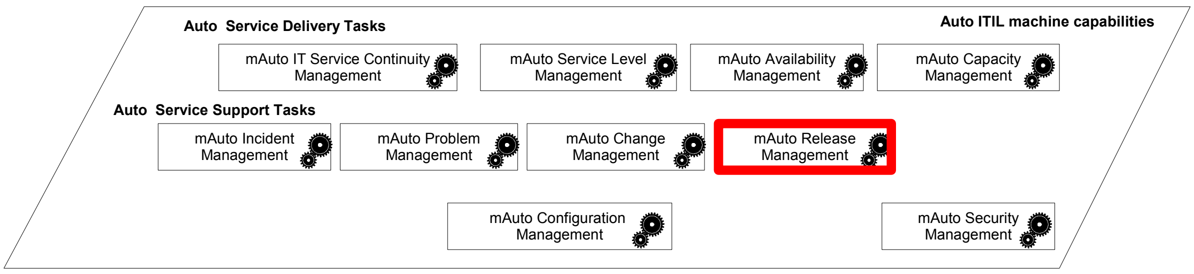
- M: analyze incidents for clusters or classes with high risk or high impact, and for reproducible errors
- A: expert systems
- A: dependency analysis
- A: statistics
  - trend analysis
- A: diagnostic aids
- A: deductive databases
- A: automated reasoning, inference
  - case-based reasoning, see [Han07]
- K: FLORA-2: An Object-Oriented Knowledge Base Language, see [Sch08b]
- K: dependencies of services and services, services and resources, resources and resources
- difficult: recreating the situation, here VM images can help for problems with application software, this allows to wrap a whole VM into a transferable file, better reproducibility by problem management staff
- IBM Red Book: Problem Determination Using Self-Managing Autonomic Technology

6.5.7 Machine capabilities in Auto Change Management


Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

- M: input from request for change (RFC) database
- M: diff tools to detect differences quickly in structured files
- M/E: versioning tools
  - svn
  - cvs
- A: change filtering
- A: change classification
- A: statistics
- A: change prioritization
  - utility, penalty functions
- P: change planning
- P: change scheduling to create forward schedule of changes
- P: back out planning
- D: change authorization
- D: voting mechanisms (change authority board)
- E: change status monitoring (execution tracking)
- E: automated review
- example: CHAMPS (Keller), Change management system with automated planning and scheduling, [KHW^{+}04], now part of Tivoli Provisioning Manager, Intelligent Orchestrator

6.5.8 Machine capabilities in Auto Release management

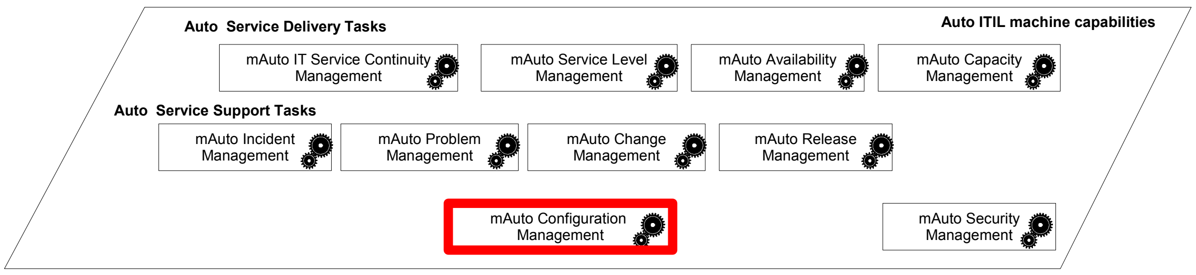

Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

- M: input requests from RFC database, that have been accepted for a new release
- A: RFC clustering/categorization to affected software packages, modules, files
- A: RFC prioritization
- P: release planning
- P: release scheduling
- D: acceptance decision support
- E: automated build tools
  - GNU build system
    - autoconf
    - automake
    - libtool
  - make, Makefile
  - ant (Java)
- E: component test automation
  - unit tests
    - `http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks`
- E: rollout
  - P: rollout planning
  - P: rollout scheduling
- E: software distribution/deployment
  - update management (usually pull mode)
    - most self-management capabilities are determined by software, therefore have a defined and capable process for updates/upgrades
    - updates and packages
      - Open Systems Gateway Interconnect, e.g. by Firefox, Eclipse
      - Java Webstart
      - Windows Update
      - Linux: rpm, yum, apt
  - software distribution (usually push mode)
    - e.g. to many thousands of clients, hierarchical
    - HP Smart Framework for Object Groups (SmartFrog)
  - alternatives to software distribution and local installation
    - terminal server solutions: graphical terminals provide graphical remote access

        - remote shells allow text-based remote access: ssh
        - WWW, browser acts as generic graphical client, plug-ins (installed locally or dynamically) for extended capabilities such as rich media and local system access
    - VM template mechanism

- definitive hardware store
    - inventory management
- definitive software list
    - user self-help portal
    - automatic license management
    - software IDs, checksums to prevent mutation/change

6.5.9 Machine capabilities in Auto Configuration management

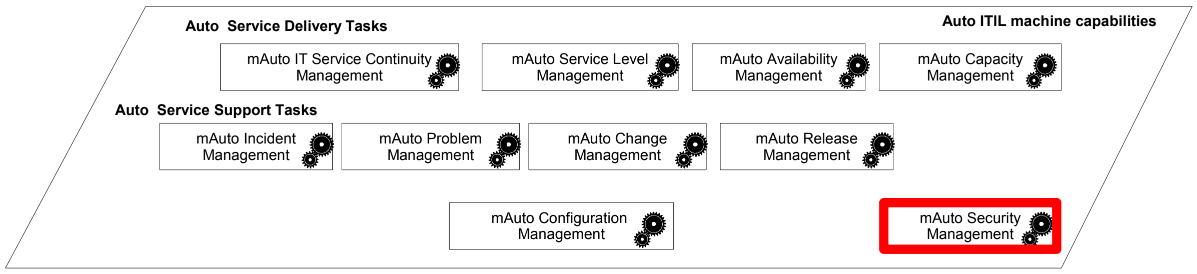

Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

- M: asset management tools
- M: discovery protocols
  - network discovery
    - Cisco Discovery Protocol (CDP)
    - Link-Layer Discovery Protocol (LLDP)
  - service discovery
    - Universal Description, Discovery and Integration (UDDI)
    - Common Internet File System (CIFS)
- A: configuration browser, visualization
- A: statistics
- A: diff tools to find differences
- A: versioning tools
- A: configuration checks, audits, verification (desired state and observed state)
- A: automated calendars for predictive maintenance, preventive maintenance
- K: Configuration Management Database (CMDB)
- K: synchronization of CMDBs, federated CMDB
- L: clean up
  - expiration mechanisms
    - aging
      - application examples:
      - time to live, time out, e.g. for ordered services
      - leases, e.g. DHCP for IP addresses
      - aging passwords, to stimulate change
    - limit the life-time of an object
      - e.g. user accounts, certificates, firewall rules
      - this necessitates periodic reviews of configuration data
    - periodic cleaning, restart, or reboot
- L: negotiation protocols
  - DHCP
    - network configuration of clients
  - ZeroConf
    - Apple protocol to find net resources such as printers, file shares

- UPnP
  - protocol to find networked home equipment, e.g. printers, DSL modems

- tools
  - L: cfengine: policy-based, describes desired state, that is periodically enforced by fix-point operators (scripts, that repair the matters, but that do also work on previously correct configurations, which they leave correct)
  - L: puppet: declarative language to describe configuration, resource abstraction
  - `http://en.wikipedia.org/wiki/Comparison_of_open_source_configuration_management_software`

6.5.10 Machine capabilities in Auto Security management


Auto Service Delivery Tasks
Auto ITIL machine capabilities
mAuto IT Service Continuity Management
mAuto Service Level Management
mAuto Availability Management
mAuto Capacity Management
Auto Service Support Tasks
mAuto Incident Management
mAuto Problem Management
mAuto Change Management
mAuto Release Management
mAuto Configuration Management
mAuto Security Management

- MI: rules from ISO 27001, corporate security policies, audit policies
- M: authentication
  - password-based
  - 802.1x (in access networks)
  - SSL (connect to servers)
  - tokens: smart cards, one-time key generators
  - biometric: fingerprint, retina eye scanner
  - physical tokens (e.g. keys)
- M: public-key cryptography
  - certification authority
  - certificate chaining
    - see [Rei08]
  - certificates for servers/users/services
  - digital signatures
  - SSL for secure connections to servers
- A: authorization
  - LDAP
  - access control lists (ACL) e.g. for file system access
  - XACML
- A: trust and reputation management
  - see [Bou09]
- A: intrusion detection system (detect)
  - snort
  - Grid IDS, see [gF08]
- A: audit tools
  - password format enforcers, crackers, periodic change enforcers
  - configuration audits
    - penetration testing tools
  - port scanner
    - nmap
  - sniffing tools
    - kismet
- K: identity management / directories

- LDAP
- ActiveDirectory
- federated IM, see [Hom07]
- VO management, see [Sch07b]

- L: artificial immune system

  Burgess98 Computer Immunology, M. Burgess, Proceedings of the Twelfth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA), page 283, (1998)
  - Ishida, Immunity-based systems

- L: intrusion prevention systems (react locally with safety measures)

- L: intrusion reaction systems (counter-attack the intruder)

- L: NAT gateways

  - LRZ Nat-O-Mat

- L: firewalls, packet filters

  - netfilter, iptables
  - RI: Layer 2 packet capturing

- L: mail / spam filters

  - rules
    - procmail
  - static rules, whitelists/blacklists, Bayesian filtering
    - spamassassin

- L: scanners for malicious code (viruses, worms, trojans, ...)

- L: virtual private network (virtually separate networks by IDs)

- L: automated software and firmware patching (also see Release management)

6.6 Catalog of Machine Capabilities Part 2: Loop-level machine capabilities

Loop-level machine capabilities cover loops and their behavior in general without making assumptions on loop implementation. This concept wraps two loops: control loops inspired by control systems and MAPDEK loops. The interior is not relevant at loop behavior level, but only later at loop step level.

6.6.1 Loop architecture

Loop interfaces When using a black-box perspective, then a loop essentially consists of two interfaces and an internal loop function, that is performed either continuously or in a triggered fashion, either periodically or event-based. MAPDEK loops and control loops match this structure as shown in Ch. 4, other implementations such as a rule engine, or a state machine would also match it. The resource interface does not depend on the implementation type of the loop function, but the management interface does. This is shown in the following example:

- If the MAPDEK loop steps were the internal model of the loop this would result in a MAPDEK loop, that was already described in Ch. 4.
- If a control loop (heavily used in engineered systems, e.g. flight control systems, heating control systems) was the internal model, then input at the management interface would be the goal input or desired objective value. Management output / status information would be the transduced value. A control loop can be based on analog input (e.g. analog voltages) or digital input.
- If a rule engine was the internal model of the loop, then input at the management interface would be rules and priorities, the loop function however at the resource interface may work just as well as with the control loop.
- If the internal model of a loop was a state machine, input would typically be a clock and an input value. The output would depend on the input and current state.
- In this thesis, we only regard MAPDEK loops. Control loops and state machines have been covered in the automation books listed above. Rules engines have been covered in policy-based management.

The two loop interfaces have been covered in the Autonomic Computing Initiative already [IBM06], we take over these ideas. The upper interface (management interface) allows to manage the loop by an external superordinate loop or human. The lower interface (resource interface) is a sensor/actor interface so that the loop can manage the sub-ordinate loop or resource. A touchpoint connects these two interfaces and provides mapping and translation services. The touchpoint is not explicitly included in EIMA though, instead we assign protocol mapping and translation to the Monitor and Execute components.

A consequence of the two interfaces for a loop is an overlay effect of external management via the management interface and the internal loop function. As long as there is no direct control from the manager to the managed system, the system can take both influences into account: varying management input and varying sensor input and keep its functions.

Things get more complicated, when we allow side-chain management (direct influence from the manager to the managed system). This would lead to a multiple manager problem and would have to be resolved in some way, as usually each manager has the assumption to be the only manager, which frequently leads to inexplicable errors.

An example for this is using `yast` (a configuration management application for Linux reading and writing configuration files) and in parallel manually editing of the same configuration files. Here, we would need conflict resolution mechanisms, e.g. priorities or policies how management commands override each other (last applies, order of events) which is made known to both human/machine entities running yast and an editor in parallel. Or we would need a file locking mechanism.

Loop function The loop function determines behavior (offered service) of the loop, and its "service guarantees" and metrics will relate to this, and not the management aspect. For example, a stabilizer loop will be measured in terms of how stable it achieves to keep a certain output parameter despite outside disturbing influences. A load balancer loop will be judged upon how well it balances the load on a range of servers and keeps the derivation of max and min load small. But it also will be judged how quickly it reacts to load changes, and it is hard to say there what is "best". The most important point here is, that loop behavior is a black-box property independent from the actual implementation of the loop.

6.6.2 Example list of loop functions

Many of the machine tasks in various domains are described to be distinctive to their respective domain, e.g. a stabilizer that keeps a cruise ship stable even in troubled water is associated with ships, marine vessels, and eventually control systems. Having a closer look however, the domain aspect of a cruise ship stabilizer is mainly applicable to the monitoring and execution interfaces between management systems and sensors/actors (here wave detectors, antiroll tanks), while the MAPE steps can be similar to other stabilizers. In fact, research in ship stabilization (at least analysis and planning, as well as concepts of control algorithms or fluid dynamics) could also be applied to e.g. stabilizers that keep large airliners stable in windy weather conditions.

While these two examples were in the domains of ship and aircraft engineering, an application in IT management is also imaginable. For the application of control theory in IT management, [HDPT04] have shown control loops in managing buffer sizes in large applications for example, or keeping the response time of an application below a certain threshold. For such abstractions at control loop level the SASO (stability, accuracy, settling time, overshoot) properties can be used for general comparison.

In essence, at loop level we need to pay attention to two types of implementations: control loops (most of them closed loops), but also open loops, many of which implement a subset of the MAPDEK steps.

To match machine capabilities with the following list of loop functions, we recommend the following approach:

For those loop functions that resemble MAPDEK loops with discrete steps: search the class libraries of programming languages for existing solutions or the set of predefined entities in modelling applications. We list example machine capabilities with the loop functions.

For the closed feedback loops that resemble control loops an implementation can be quickly found with control loops modelled in Matlab or Scilab. These applications contain libraries for control loops with different properties regarding stability, accuracy, settling time and offset. They have been used extensively in control theory and all its application domains.

The function of loops with management automation routines often falls into one of the categories listed on the next few pages in this section. The lists are by no means complete, but shall give the reader an idea about many examples of loop function. Interestingly, they do occur over and over in all kinds of management automation systems in all technical domains.

Loops that interface to (human) managers

Configurator (also called wizard) A configurator groups parameters into logical groups and provides a configuration interface at a higher level of abstraction. During the configuration of a certain task (e.g. software installation or software customization) it takes into account dependencies between parameters and options, and offers only compatible choices and adaptively determines default values. Configurators are used in applications such as making configuration parameters more accessible in a step-by-step to a user e.g. service configurators.

Machine capabilities: constraint solvers and rule interpreters form the basis of sophisticated configurators. Example implementations: Drools Rules Engine.

Known applications: product configurators for computers (e.g. by Dell) or car configurators (all major car manufacturers). The concept is also well known from software installation in Windows. We will need the same for the composition of IT solutions, IT services etc.

Solution set creator A solution set creators composes a solution set with respect to constraints. This is then used as input by subsequent loops. This part of a configurator is driven by requirements and not by composition of parts. It finds a set of matching solutions based on a set of constraints according to customer-level requirements. It can find matching single items or combinations of items and ideally also work with soft constraints.

Machine capabilities: Use a database for solution candidates, a constraint solver for filtering. Mapping requirements to solutions can be based on rules or tables.

Known applications: solution sets of train connections or flights between cities, set of route alternatives in navigation systems.

Ranker A ranker ranks the solutions in the set according to a utility function. The value of the ranking is that more appropriate choices (according to the scoring functions) are ranked higher than other choices.

Machine capabilities: Use a solution set creator first, then rank with a scoring function.

Known applications: Top 10 listings in IT managment monitoring systems (severe faults, for example). Top 10 listings in online shops.

Recommender A recommender advises for or against certain solutions in a result set.

Machine capabilities: See “ranker”, then apply a filter to select the top x entries.

Known applications: Recommendations on the sequence of active probing in fault analysis. Product recommendation systems in sales for related products based on the current purchase or previous purchases, e.g. Amazon.

Decision maker A decision maker decides for one option from a set of alternatives.

Machine capabilities: Use a ranker first, then execute top entry.

Known applications: n-modular redundancy (e.g. triple modular redundancy) to mask errors in parallel execution units. Used in airplanes for critical decisions.

Effects forecaster Before making decisions, decision makers typically want to know about the consequences of their decision, such as for a “yes” or “no” question. An effects forecaster will at least tell next actions, for more complex scenarios it will allow “what if” type analyses. Example: Imagine a large rule set for firewall policies. Now, there is a certain change to this rule set and the confirmation question whether to accept the change or not. An effects forecaster could visualize what ranges of IP addresses and TCP/UDP ports are affected by the change in what manner. Depending on context knowledge it could even include services to make the decision less abstract.

Machine capabilities: statistical methods: extrapolation, trend analysis, cause-effect analysis, deduction (e.g. deductive databases), inference engines, reasoning methods.

Known applications: Forecast of congestions in networks, such as data networks or road traffic networks. Intrusion prevention systems. Early-warning systems. Weather forecasts.

Consistency checker Configuration sets easily get inconsistent. This leads to many problems such as: duplicate entries for the same configuration parameter, conflicting values of dependent configuration parameters, missing mandatory settings, old configuration parameters that are no longer taken care of. The severity of such inconsistencies spans a wide range from simply redundant entries to system crashes or boot failures. Therefore, such inconsistencies should be found and resolved. An example are Windows registry checking programs.

Machine capabilities: Policy-based checks, relational databases that enforce features of data consistency on their own within the limits of structure, that the database designer used (e.g. does not check BLOBs), but they are not well suited for unstructured or semistructured data.

Known applications: cfengine, relational databases, auditing systems (security audits, financial audits).

Explainer An explainer loop tells in human-understandable terms what has been done, is done, or will be done next and why. Ideally, it would also allow questions and queries.

Machine capabilities: natural language creation and parsing.

Known applications:

Standard uses of an open feedback loop

Notifier / Watchdog A notifier monitors some managed object, and sends a message when a certain condition applies. A watchdog is slightly more versatile, and triggers an alarm or a predefined action when applicable.

Machine capabilities: simple analysis methods (e.g. comparing numbers, counting events, waiting for events), simple actions (send message, trigger a known handler routine)

Known applications: timer events, threshold alarms, intrusion detection systems, SNMP trap senders/receivers.

Supervisor A supervisor supervises actions and analyzes them for patterns. It can then warn of potential mistakes and recommend other better actions. Doing so, it can also adapt controls to the current situation.

Machine capabilities: recognition of gestures and action sequences, context recognition, situation recognition.

Known applications: Microsoft Clippy in MS Office.

Escalator An escalator transport escalations to the manager.

Machine capabilities: see notifier.

Known applications: via human-bound interfaces: paging system administrators, sending short messages, e-mails, phone calls. The same exists for escalating faults to technical systems via API's, e.g. SNMP traps.

Logger A logger writes status data to a log device, such as a file or data base.

Machine capabilities: simple transformation of events to human-readable log entries.

Known applications: syslog protocol and demon. Aspect-oriented programming to add the logging aspect to existing programs.

Standard uses of a closed feedback loop

Optimizer An optimizer improves some system parameter(s) regarding to a goal function.

Machine capabilities: continuous recomputation of goal function applied to current situation. Adaptive measures then bring the current state closer to an optimal state.

Known applications: disk optimizers against defragmentation, adaptive transmission parameters in communications.

Stabilizer A stabilizer protects some system parameter(s) from external distortions and oscillations and makes changes more evenly spread.

Machine capabilities: stabilization means to dampen changes. This can be done by running optimizers less frequently (temporal dampening) or by introducing change costs into the optimization problem.

Known applications: frequency in network protocols like STP, OSPF, RIP, BGP.

Load balancer A load balancer distributes incoming load evenly to a set of resources.

Machine capabilities: adaptive distribution of incoming load to a number of output entities. Bin packing algorithms. Queueing algorithms.

Known applications: load balancers on clusters/server pools. Queueing in router queues for different QoS treatment.

Queue assigner A queue assigner puts incoming events into one of multiple output queues.

Machine capabilities: Queueing algorithms.

Known applications: Queueing in router queues for different QoS treatment.

Controller A controller is used in control theory. Linear controllers such as PID controllers are among the best studied loops and easy to implement.

Machine capabilities: Many kinds of controllers are modelled in Matlab toolboxes to name only one prominent example.

Known applications: All sorts of control systems from heating applications to flight control systems.

6.7 Catalog of Machine Capabilities Part 3: Loop-step (MAPDEK) level machine capabilities

6.7.1 Overview

The following list gives an overview of typical tasks, methods from various domains, and examples from IT management automation all arranged along the interior components in a MAPDEK loop (see Fig. 6.8, which is repeated here for the reader's comfort). The sections on the loop steps have been split to list to typical tasks, and applicable methods.

Hint: page breaks in this section have been added to ensure that the single loop step categories each are easy to navigate to when reading the document.

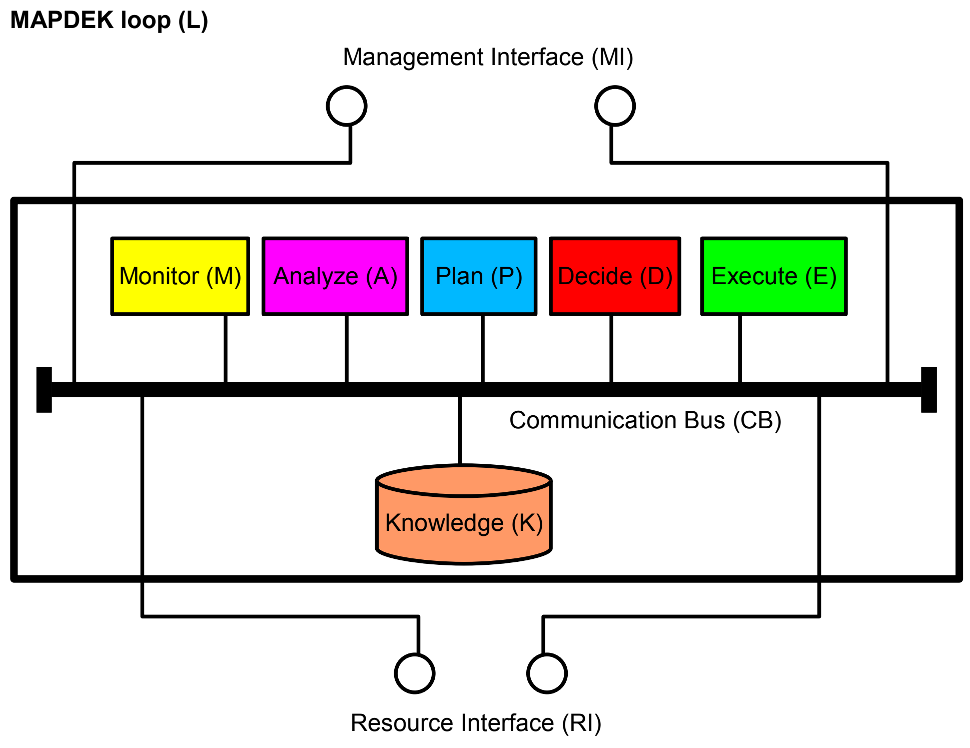


Figure 6.8: Interior of a MAPDEK loop

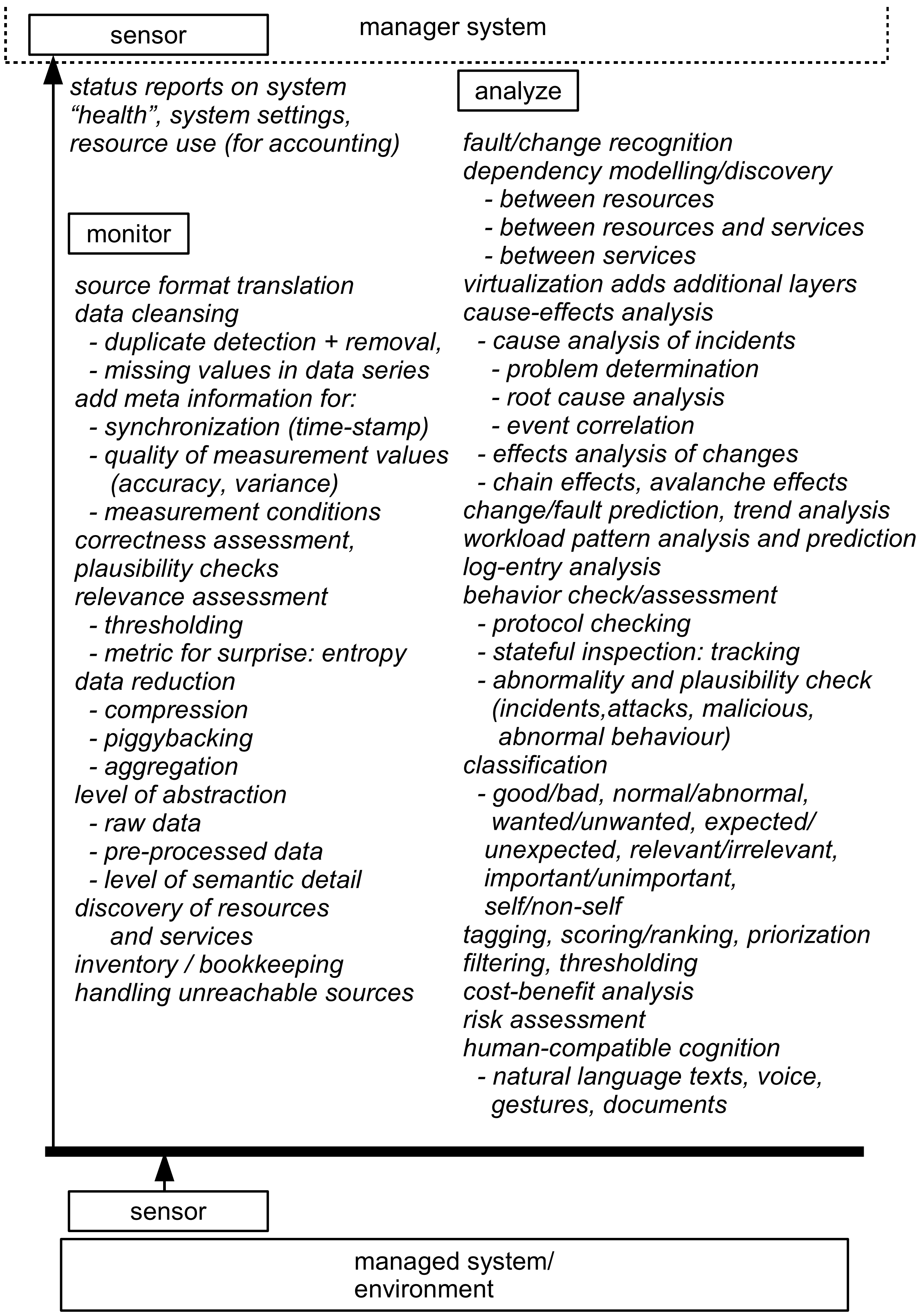

sensor
manager system
status reports on system "health", system settings, resource use (for accounting)
analyze
fault/change recognition
dependency modelling/discovery
- between resources
- between resources and services
- between services
virtualization adds additional layers
cause-effects analysis
- cause analysis of incidents
- problem determination
- root cause analysis
- event correlation
- effects analysis of changes
- chain effects, avalanche effects
change/fault prediction, trend analysis
workload pattern analysis and prediction
log-entry analysis
behavior check/assessment
- protocol checking
- stateful inspection: tracking
- abnormality and plausibility check (incidents,attacks, malicious, abnormal behaviour)
classification
- good/bad, normal/abnormal, wanted/unwanted, expected/ unexpected, relevant/irrelevant, important/unimportant, self/non-self
tagging, scoring/ranking, priorization
filtering, thresholding
cost-benefit analysis
risk assessment
human-compatible cognition
- natural language texts, voice, gestures, documents
monitor
source format translation
data cleansing
- duplicate detection + removal,
- missing values in data series
add meta information for:
- synchronization (time-stamp)
- quality of measurement values (accuracy, variance)
- measurement conditions
correctness assessment, plausibility checks
relevance assessment
- thresholding
- metric for surprise: entropy
data reduction
- compression
- piggybacking
- aggregation
level of abstraction
- raw data
- pre-processed data
- level of semantic detail
discovery of resources and services
inventory / bookkeeping
handling unreachable sources
sensor
managed system/ environment

Figure 6.9: Methods for loop tasks in Automation in IT and Engineering. This figure shows tasks within the loop. When tasks match among systems and when done with caution, we can mix methods as well as associated implementation, tools, experience and design processes from several domains to achieve an overall improvement in the design of a new management automation system.

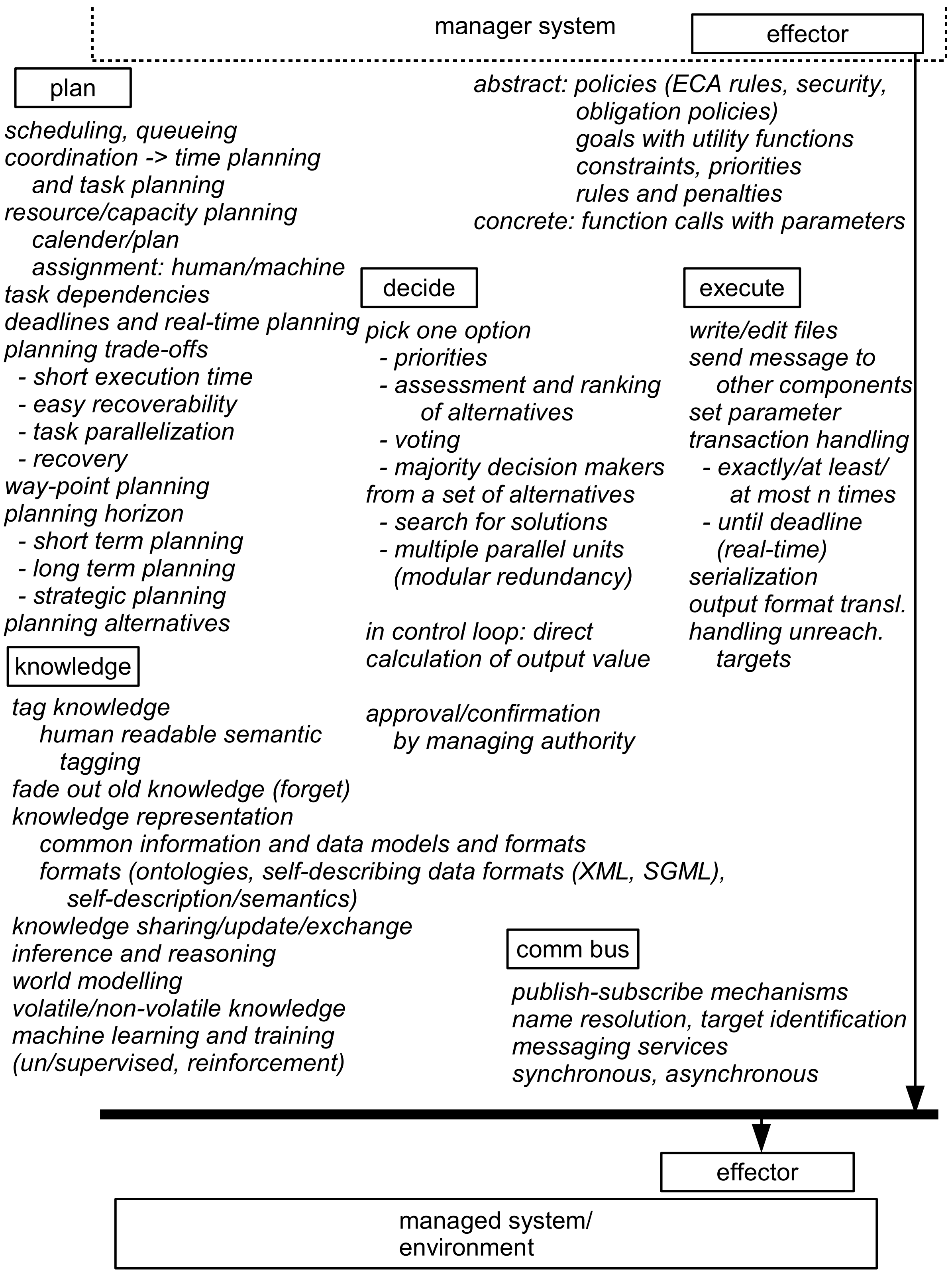
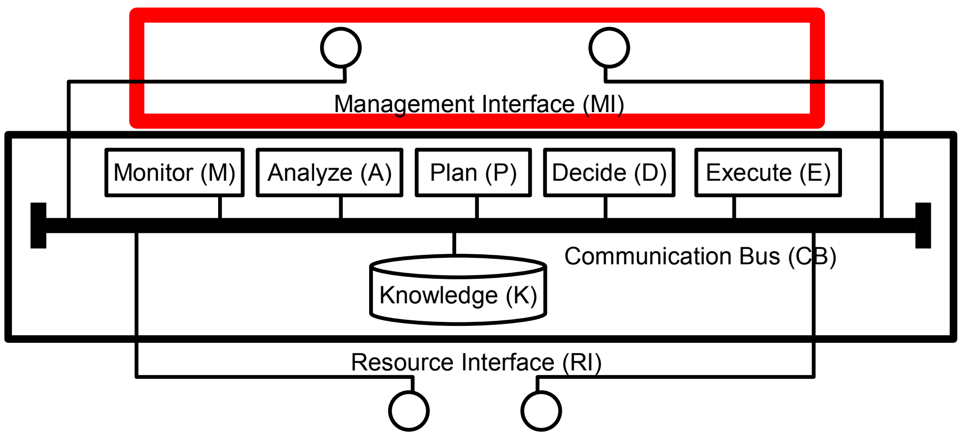


Figure 6.10: This is especially applicable to the analysis and planning steps, which are backed by a large knowledge pool from artificial intelligence.

6.7.2 Management Interface (MI)



Description The management interface of a loop serves for interaction with a superordinate loop/human. Thus the role communicating to the loop can be a hManager or mManager. The purpose of this interface is to control the loop via high-level objectives at a level of abstraction that is comfortable/appropriate for the manager. In principle, we can apply this interface between humans (a boss and an employee), between a human and a management systems, or between machine enabled loops.

Communication at this interface is two-way: management commands are sent from the manager to the loop, status information, reports and escalations are sent back.

Messages from manager to managed entity

- Policies
  - policies aim to express high-level objectives or rules in natural language with only minimal formal syntax
  - are seen as a good interface to upper management
  - policy-based management (see [Kem04, Dan07], research at Imperial College by Sloman)
  - policies can have many forms:
    - event - condition - action (ECA) rules
    - obligation policies (constraints)
    - goal policies
    - priorities
  - technical infrastructure
    - policy refinement
    - policy decision point
    - policy enforcement point (IETF Policy WG)
    - policy languages and engines
      - WS-Policy
      - Ponder
    - policy engine
      - waits for events, checks conditions, runs actions
      - selects appropriate policies
      - example: Ponder
- Rules

  - also here: rules are regarded as more comprehensible to people not used to code level
  - rule selection with implicit goal, evt. minimize overall penalty
  - rules engines
    - Drools rules engine - business rule management system (BRMS) and an Rules Engine
    - based on RETE algorithm

- Objectives/Utility functions
  - utility functions and a generic optimization loop would be universal construct for optimizing loops
  - may be used to quantify value: economic/monetary value, personal value, time value
  - hard to find utility functions that are both simple and true
  - hard to understand/debug at an abstract level
  - no standard format
  - current status: research topic
  - example: Pagerank algorithm in Google

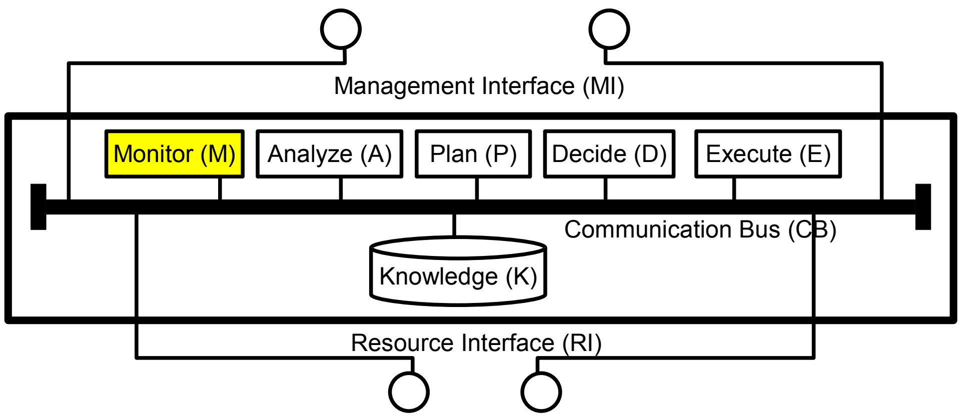
Messages from managed to manager

- status reports
  - system settings
  - usage/accounting information
  - performance indicators
  - health indicators
  - events
    - Common Base Event
  - reports
    - reporting tools
      - Jaspersoft
    - visualization/formatting
      - graph layouting, e. g. graphviz
      - automated image editing, e.g. imagemagick
      - automated charting, e.g. gnuplot
      - typesetting reports, e.g. LaTeX and XSL:FO
      - maps
- escalations
  - exceptions

Communication Middleware between loops

Communication at the management interface with an upper loop is communication in distributed systems. See the hints on machine capabilities for the communication bus inside a MAPDEK loop.

6.7.3 Monitoring (M)



Description The monitoring component mainly gathers information from sensors. In IT management, sensors are typically not analog sensors measuring physical objects, such as temperature, speed, humidity, light intensity, but software interfaces that allow to get readings from.

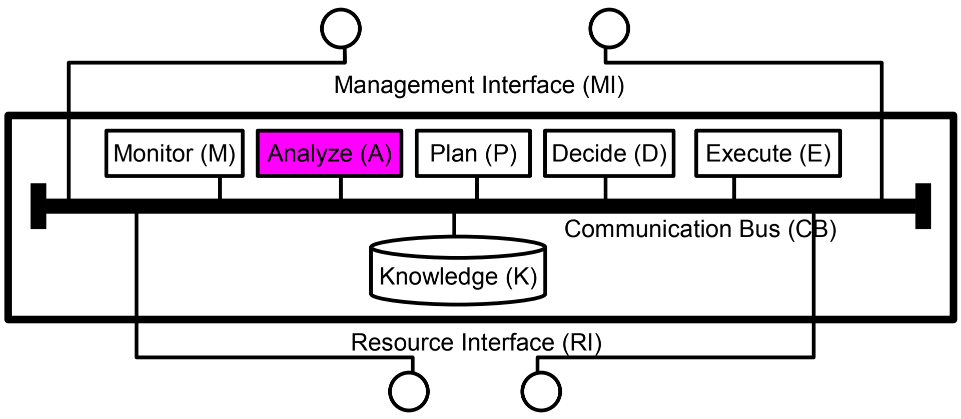
- tasks
  - connection handling
    - source authentication
    - decryption
    - handling unreachable sources
    - data reduction
      - compression
      - piggybacking
  - add/check meta information for:
    - synchronization (time-stamp)
    - quality of measurement values (accuracy, variance)
    - measurement conditions
    - data source
  - source format translation
  - data cleansing
    - duplicate detection and removal
    - handling missing values in data series
  - correctness assessment, plausibility checks
  - relevance assessment
    - thresholds
    - metric for surprise: entropy
  - store data
    - keep amount of data manageable
    - aggregate old data
      - RRDtool
  - simple display
    - aggregation (changing level of abstraction)
      - raw data
      - pre-processed data
      - level of semantic detail

        - views
            - charts
            - event lists
            - traffic-lights semantics
            - maps

- input properties
    - media
        - data
            - boolean (button presses)
            - integer
            - real
        - text
            - natural language
            - artificial/formal language
        - sound
            - voice
        - image
            - 2d
            - 3d
    - data sampling
        - frequency
        - data rate
        - accuracy

- tools
    - monitoring tools (for networks, servers, applications, ...) via API's
    - see Resource Interface

6.7.4 Analysis (A)



Description The analysis component processes information input and tries to extract patterns, identify situations, or find statistically distinctive features. In abstract terms, analysis shall find out, whether certain changes in the loop are necessary to adapt to goal changes or sensor input.

- tasks
  - fault/change recognition
  - dependency modeling/discovery
    - between resources
    - between resources and services
    - between services
    - virtualization adds additional layers
  - cause-effects analysis
    - cause analysis of incidents
      - problem determination
      - root cause analysis
      - event correlation
    - effects analysis of changes
      - side-effects
    - chain effects, avalanche effects
  - change/fault prediction, trend analysis
  - workload pattern analysis and prediction
  - log-entry analysis
  - behavior check/assessment
    - protocol checking
    - stateful inspection: tracking
    - abnormality and plausibility check
      - analysis for security incidents, attacks, malicious, abnormal behavior
  - classification
    - put incoming events into one of several categories
    - good/bad
    - normal/abnormal
    - wanted/unwanted
    - expected/unexpected
    - relevant/irrelevant

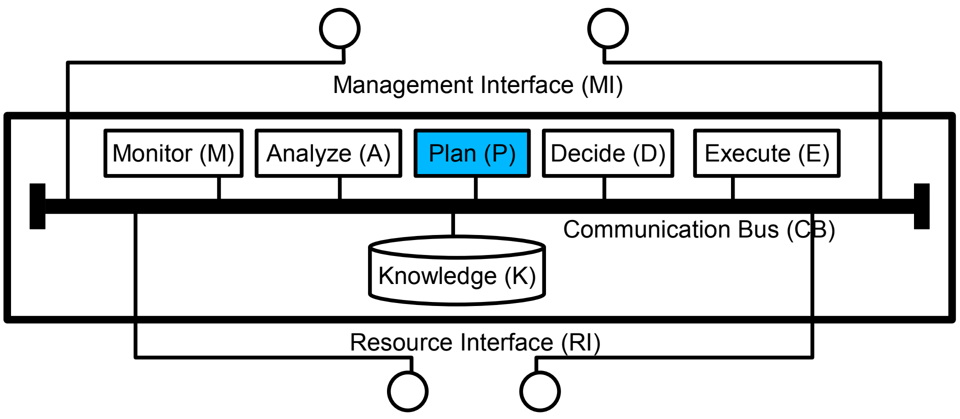
        - important/unimportant
        - self/non-self
    - reasoning
        - case-based reasoning
    - inference
        - XSB inference engine
        - FLORA-2 deductive database
    - tagging
        - add tag
            - Thunderbird, supported by human
    - scoring/ranking
        - add score
            - spam classifiers
            - scorefile in usenet news readers
            - efficient human to human communication
    - prioritization
    - filtering
        - monitor an event queue, drop certain events
        - thresholds
        - killfile
    - cost-benefit analysis
    - risk assessment
    - natural language processing, voice recognition, optical character recognition
    - rater
        - calculate utility of alternative choices with a utility function

- methods
    - linear programming
    - constraint satisfaction solver - configurators
        - dependencies as constraints
    - Artificial Intelligence
        - event correlation / machine reasoning
            - case based reasoning
            - rule based reasoning
            - hybrid
        - decision trees / classifiers
        - Bayesian networks and filters - spam filters
        - neural networks - OCR
        - fuzzy logic - heat control, home appliances
        - expert systems (classification, categorization, symptoms → cure) - medicine
        - pattern matching (e.g. regular expressions) - log file matching
    - FMEA - Failure Modes and Effect Analysis
        - used in car maintainability engineering
        - also named in ITIL V3
        - also in Boeing system design
    - CFIA (Component Failure Impact Analysis)
    - (business) impact analysis
    - clustering
    - sorting, indexing, searching - Google
    - computational intelligence: counting and statistics ("counting instead of thinking")

- statistics
  - correlation and dependence analysis
    - regression
  - trend analysis
    - extrapolation
    - value known, time unknown: When will threshold be broken?
    - value unknown, time known: Where will the value be at a certain time in the future?
  - guessing missing values
    - interpolation / extrapolation
  - heuristics
  - feature/pattern recognition: check statistical relationships
    - count certain patterns
    - find similar patterns
- relevance assessment
- distributed hash tables
- self-organizing maps
- situation analysis/mapping
- graph theory
  - matching
  - covering problem
  - set problems
- Online Analytical Processing (OLAP)
- data mining

* tools
  - statistics
    - packages in R (programming language)
    - `http://en.wikipedia.org/wiki/List_of_statistical_packages`

6.7.5 Planning (P)



Description The planning component typically builds a plan, that aims to reach a desired state from a current state, and that breaks down the way to this goal into manageable actions/steps. Scheduling puts actions into a schedule, and thus attaches points in time to actions.

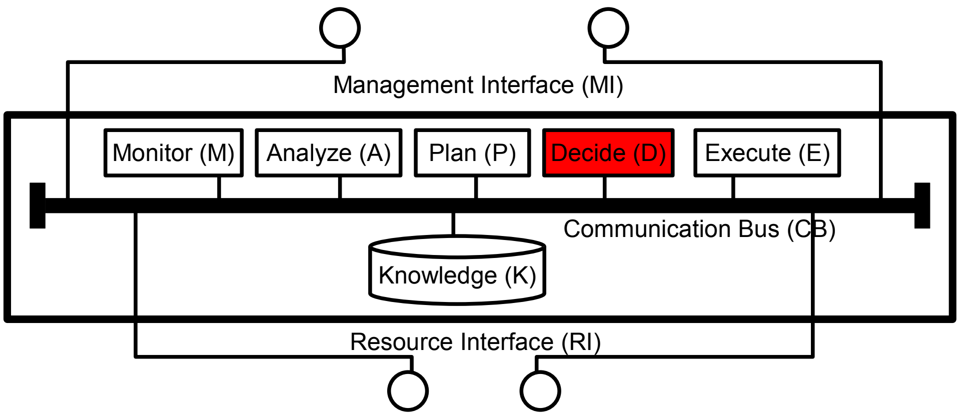
Planning is only useful, if we will not receive another goal by a management command in the near future that overrides the previous goal and therefore the plan. The necessity of planning depends on the frequency of management commands. For systems with frequent management goal changes, reactive systems are used.

Regarding computational complexity, planning algorithms often quickly suffer from state-space explosion. In such situations, a reasonable problem model needs to be found to make the planning algorithm applicable with the results keeping certain value despite simplification.

- scheduling, queuing
  - scheduler
    - take jobs, assign start and end times in a schedule
    - task scheduler
    - grid scheduler
    - batch scheduler
  - periodic execution
    - `cron`
    - Windows task scheduler
  - one-time execution
    - `at`
- forecasts
- coordination: time planning and task planning
  - take into account: task dependencies
- resource/capacity planning
  - calendar/plan
- recovery planning
- deadlines and real-time planning

- planning trade-offs
  - short execution time
  - easy recoverability
  - task parallelization
  - easy recovery
  - quick replanning
- way-point planning
  - example: route planning in navigation systems
- planning horizon
  - short term planning
  - long term planning
  - strategic planning
    - hard to automate, if strategies cannot be modeled
- create alternative plans, or only one?
- general purpose planners
  - STRIPS
    - Stanford Research Institute Problem Solver
  - SHOP
    - simple hierarchical ordered planner
    - jSHOP
    - `http://www.cs.umd.edu/projects/shop/description.html`
- dynamic replanning
  - in case of non-predicted change
- tools
  - Enterprise Resource Planning (ERP) tools
  - project planning (e.g. MS Project)

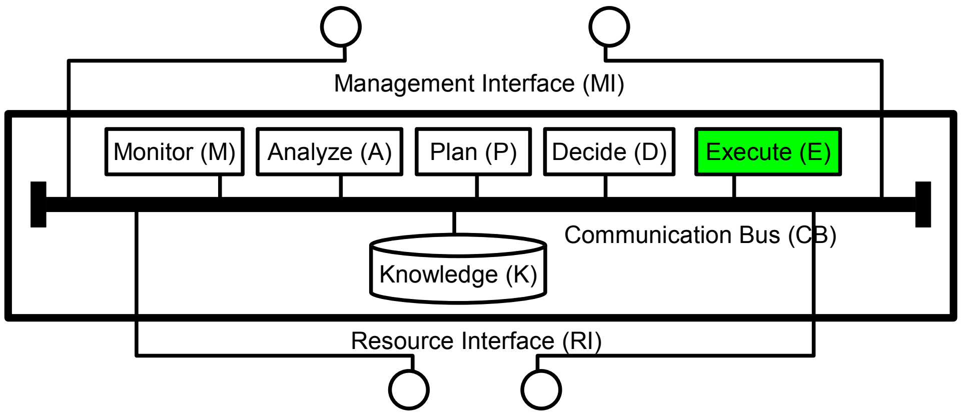
6.7.6 Decide (D)



Description Decision is a phase of selecting one option from a set of options, e.g. a certain plan, reviewing it and its actions taking into account its consequences (along criteria that may be unknown to the roles that have created the set of alternatives), and authorizing execution. It is a typical step left to an upper manager (from the viewpoint of a loop). It can include to deny all options or combining features of multiple options to a new plan.

- to decide means to pick one option from a certain set of alternatives
- set of alternatives
  - typically by searching for solutions in a certain space, keep the best ones
- pick one
  - priorities
  - assessment and ranking of alternatives
- voting schemes
  - secret sharing schemes
- majority decision makers
  - triple modular redundancy
    - run three identical components, ideally they should all output the same result for the same input
    - but there may occur differences due to technical defects or outside influences
    - then hopefully, only one system is affected, which can be recognized by its differing results
- approval/confirmation by authority
- preview of decision effects
  - simulation
  - forecast
- decision support systems

6.7.7 Execution (E)



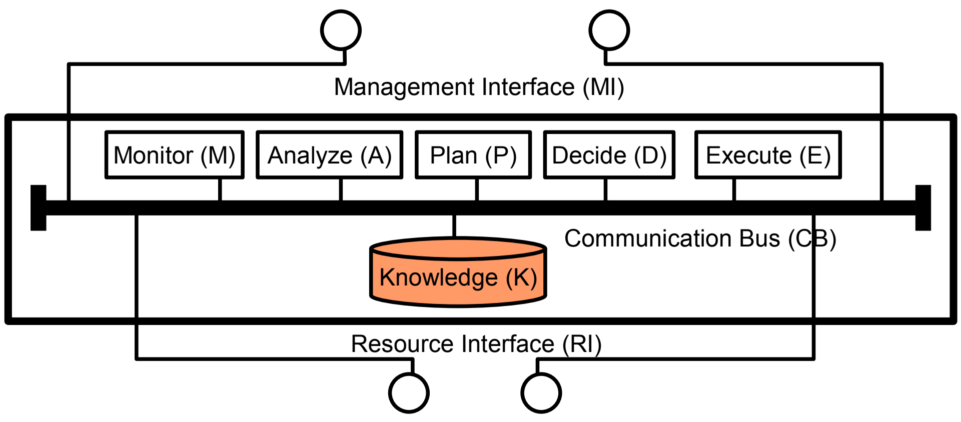
Description Execution performs the action(s) that have been decided before. It does so via actor interfaces. In IT management automation these are no physical actors, but also software interfaces where files can be modified, messages sent, actions triggered.

- actions
  - write/edit files, esp. configuration files
  - send messages/signals to other components
  - parameter setting
  - execution tracking
  - handling of unreachable targets
- methods
  - network management set instructions
    - SNMP
  - smart actuator
  - output format translation
  - serialization
  - message formation
    - encryption
    - authentication
    - compression
  - transaction handling/RPC call semantics
    - exactly n times
    - at least n times
    - at most n times
    - until deadline (real-time)
  - retry
    - retry execution of actions multiple times to work around transient faults in systems
- common actor types
  - actors to machines
    - messages, signals
      - remote/local function calls with parameters
      - starting/stopping processes

    - physical actors
      - servo engines, valves, ...
      - hardly used so far in IT management
  - actions to humans
    - display/visual
      - human sense: sight
      - video output
      - projector
    - sound/acoustic
      - human sense: hearing
      - speaker
    - other human senses: taste, smell, feel/haptic, temperature
      - hardly used so far in IT management

- tools
  - snmp-lib
  - API's of hardware, software
  - also see Resource Interface

6.7.8 Knowledge (K)



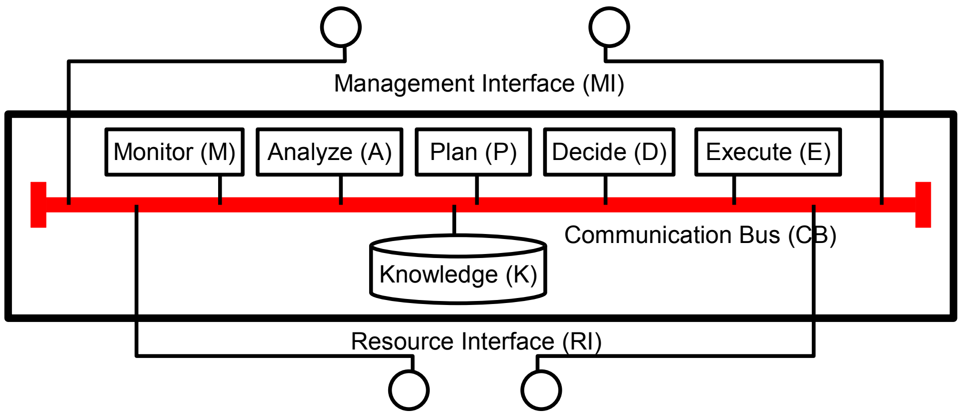
Description Knowledge is not a functional step in a MAPDEK loop but a component that acts as a loop-local store for information like a knowledge base. We can assume, that most of the knowledge known at design time (such as dictionaries, static dependencies, category names, formulae, weights) is preloaded by systems development. At this point, systems operations may choose from this knowledge, but has limited influence on changing it or bringing in completely new knowledge. Knowledge will also store operational data such as logs on past sensor input, manager input, actions.

However, knowledge is not a passive data store alone. Where applicable and useful, actions like machine learning and inference can create new information from existing one, and also from sensor input. This will be restricted to very specific cases, but it is likely to be extended once algorithms and resources catch up. Like in data collection and analysis, about 20 years ago it would have seemed impossible to create an index for a distributed collection of documents such as the WWW and search in this vast amount of data in milliseconds and find relevant results, yet today Google is close enough to achieving this goal so that it is used in practice. Wolfram Alpha, which is coined as “computational knowledge engine”, already does a remarkable job in query interpretation for some questions/queries on statistics.

- tag knowledge
  - human readable semantic tagging
- fade out old knowledge (forget, aggregate)
- knowledge representation
  - common information and data models and formats
  - formats
    - ontologies
    - self-describing data formats (XML, SGML)
    - self-description/semantics
- knowledge sharing/update/exchange
- volatile/non-volatile knowledge
- training
  - mandatory or optional training
  - training sets

- modeling
  - model specification
  - model checking/model verification, e.g. a protocol
  - formal descriptions of system architectures
- machine learning
  - helps to reduce initial knowledge that the system must be equipped with when shipped
  - prerequisite for wide-range adaptation, that was not intially considered by developers
  - increases heterogeneity (every system may have learnt something different): may hinder predictability, maintenance
  - influence on robustness unclear
    - good: learning keeps the system a moving target, not all instances are the same, "genetic variation"
    - bad: attacker may exploit learning to mislead the system
    - bad: learning implementation may be complex and error-prone
  - combination of update management (direct update of knowledge base) and machine learning
    - machine learning
      - local knowledge update increases heterogeneity within individuums
      - local effects forecast
      - short term planning
    - direct update of kb via communication
      - global knowledge update increases homogeneity within individuums
      - global effects forecast
      - long-term, strategic planning
  - behavior in case of power loss
    - storage of knowledge non-volatile memory
      - state-ful over power cycling, resume where you were before
      - remembers things from last time
      - may need extra reset button
    - storage of knowledge volatile memory
      - (state-less) over power cycling, defined start state
      - must be saved before removing battery, otherwise it is newly learned
      - easy to reset
    - resetability through power-cycling
      - wished for by customers?
      - where? where not?
      - physical reset or software reset?
  - methods
    - reinforcement learning
    - cognitive learning
- content
  - historic data (logs)
  - models, e.g. model input by system designer
  - dependencies, e.g. those that cannot be discovered, or that have been discovered in the past
  - general information store for MAPDE steps
- knowledge base

6.7.9 Communication bus (CB)

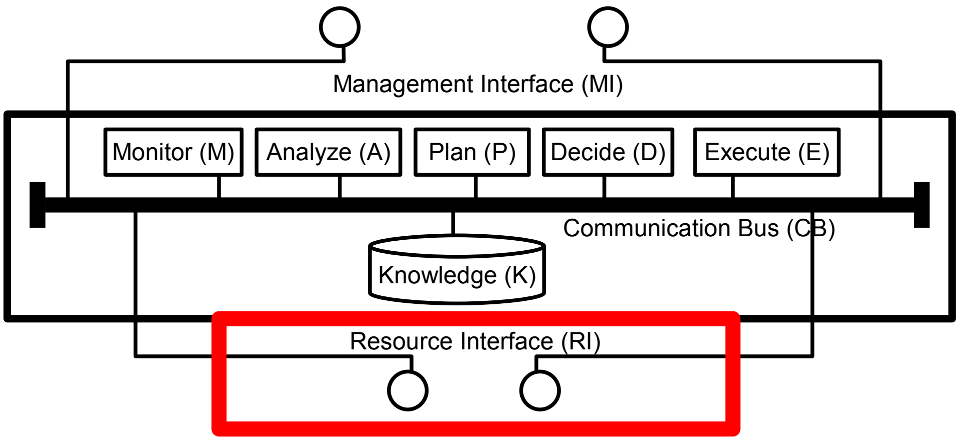


Description The communication bus is the abstract connecting element in a loop. It provides a communication facility for the MAPDEK components and the two outside interfaces. Its structure is bus-like so as to not constrain communications, and to easily replace components with other alternatives. In a real implementation, this logical construct can be implemented in a different way, if this is necessary for performance reasons.

The same techniques can also be used for loop-to-loop communications, so that for example a lower loop can handle MAPEK, and an upper one the decision step.

- messaging services/middleware
  - Web services/SOAP
  - Java Message Service (JMS)
  - CORBA
- workflow management systems
- BPEL engine
- loose coupling, dynamic binding
- name resolution, target identification
- publish-subscribe mechanisms

6.7.10 Resource Interface (RI)



Description The lowest loops that actually interface to resources will need management interfaces to the managed hardware and software elements to read monitoring data from them and send control data to them. The list shows the major management interfaces with most managed elements.

In front of a resource interfaces, an adapter needs to care for protocol and data translation between the loop format and the resource specific format. This adapter should be encapsulated, so that later it can be changed more easily and evt. modified or extended. Only few management interfaces are so mature, that they last for a very long time. So, at this place adaptation to changing resources, changing management interfaces and protocols as well as changing capabilities will be needed over and over again.

The adapter should also prepare for connection loss, non-existing communication partners, and delayed communications and handle these cases gracefully.

Management Interfaces

- Applications
  - proprietary API's
  - Web services (SOAP)
  - WWW interface via HTTP
  - Application Response Measurement (ARM)
- Operating system
  - installed management agent software, e.g. for SNMP or proprietary
  - connect to a local text terminal, e.g. ssh, PowerShell
  - connect to a local graphical terminal, e.g. VNC, RDP, XDMCP
  - configuration
    - Windows
      - registry
      - services manager
      - local software list
      - Windows API Web Based Enterprise Management (WBEM)
    - Linux

      · files in /etc
      · rpm, apt
      · yast, redhat-config, ubuntu system tool

- Virtualization layer
  - Vmware: API's, VMWare server console, VMware cmd, Vmware vCenter
- Servers
  - hardware agents in BIOS (e.g. Sun Lights Out Management, Intel AMT Active Management, Dell Management Console, ...)
  - Wake on LAN to switch server on remotely, NIC in standby
  - keyboard, video, mouse (KVM) access (analog, digital, via IP)
  - built-in remote maintenance cards
- Storage
  - SNIA Storage Management Initiative Specification (SMI-S)
  - file system
- Network equipment (switches, routers, printers, UPS)
  - Simple Network Management Protocol (SNMP), Remote Monitoring (RMON)
  - NetConf
  - NetFlow, IPFIX
  - router CLI: terminal to Cisco IOS, Juniper OS, OS of HP network equipment, ... via LAN (telnet, ssh, WWW) or serial connection
- Universal data-center equipment
  - remote power distribution units (power sockets manageable via serial interface, IP (ssh, HTTPs))
  - door access
    - ∗ proprietary electronic, programmable locks with tokens for workers
  - heating, ventilating and air conditioning (HVAC)
    - ∗ proprietary industrial standards, SNMP
  - remote visual monitoring
    - ∗ simple web cams are common-place for remote visual monitoring by humans
    - ∗ also for surveillance purposes
  - machine identification
    - ∗ human-readable tags: sticky labels
    - ∗ machine-readable tags: RFID, bar codes for inventory management
    - ∗ remote switchable visual beacons (e.g. DELL)

6.8 Conclusion

EIMA motivates a catalog of machine capabilities, so that function allocation to machines does not only result in a wish list. Instead, with such a catalog each machine-allocated function can be tagged with one or more hints to software development on applicable methods and tools for each task. The machine capabilities are described at a variable level of abstraction to serve as implementation patterns, with which even complex systems can be sketched to be constructed from components.

A step-wise matching method has been presented to find appropriate machine capabilities in a catalog. It is rather general and oriented along structure, not semantics. By design, it is not exactly a cook-book approach. The reason is, that a cook-book approach would underestimate the range of options to choose among implementation alternatives.

Overall, the EIMA approach resembles the one of tool-boxes, where complex solutions are composed of individual components along a common structure. The entries in the individual sets of the tool box (and therefore the listed machine capabilities in the catalog) shall serve as a good collection of knowledge. It should not be considered a closed and complete set however. The catalog itself has been structured into three parts, each of which can be considered as one such tool-box at a different EIMA level of abstraction.

An example has shown the application of the method and demonstrated how it can be used. Resulting from function allocation and machine capability assignments we have reached the point where the overall system has been decomposed into machine functions which can then be implemented by domain experts in AI, software development, statistics and all the other domains that have experience in building such smaller systems. The overall management automation system will essentially integrate its cognitive piece parts and provide the glue around the internal logical (MAPDEK) and communication components (CB, RI, MI).

All in all, the EIMA approach does not build a system on its own. Instead it helps to functionally decompose (bigger) IT management automation systems and to name machine capabilities to implement parts of the overall system. It does neither replace experience nor knowledge of the automation designer, but it supports his memory and capacity for remembering. A human can hardly remember many thousands of algorithms, methods, tools, libraries, etc. But when he sees a list of 20 keywords for such machine capabilities, then for a certain task he may have an instant intuition: “Yes, neural networks (one of the 20 keywords) are a promising method to tackle this problem.” So after EIMA’s indexing filters have composed a set of likely applicable methods, the actual decision or priorization which methods, tools to apply does a human automation designer do. He should however not rely on completeness of the EIMA catalog of machine capabilities.

More examples that at the same time serve as more fleshed out application examples for the EIMA approach are shown in the next chapter.

7 Application Example: Automation potential in End-to-end links with Quality of Service for a WAN

Contents

This chapter will demonstrate, how EIMA is applied in one chosen example scenario about end-to-end links with Quality of Service for a WAN. It will provide proof, that the EIMA approach is beneficial and does in fact show up automation potential (utility).

In addition to this main scenario, three other ones dealing with update management in different setups will only shortly be introduced, in which the author of this thesis has successfully applied EIMA. This should make plausible, that EIMA is universally applicable to IT management automation scenarios (generality).

7.1 Selection of Scenarios

Utility vs. Generality EIMA's goal is in quickly identifying automation potential in IT management automation scenarios, after they have been aligned to the EIMA structure. So, it's about effectivity and generality. However, regarding generality, IT management automation scenarios span a wide range along multiple aspects. In particular, they differ in:

- IT resources to be managed automatically
  - type: of device/software
  - scale: small scale, large scale
  - distribution (spatial complexity): local and distributed ones
  - automation maturity before: level of automation
- tasks to be automated
  - subset of the AutoITIL scheme
- organizational complexity
  - number and type of involved organizations and people

- available input when EIMA starts
  - automation from scratch or raising the level of automation

The difficulty of a proof of concept for EIMA along the two aspects 1) utility and 2) generality is that these are typically a contradiction: For a method to have maximum utility for a certain scenario, the method will need to take into account all the details, limitations, constraints and optimizations. But for a method to be universal, it will need to abstract from those very individual aspects of a certain scenario.

This contradiction has a few implications: With EIMA, we cannot possibly think at a level of abstraction, that takes into account all the details specific to the scenario. Otherwise, we would be faced with the whole complexity of the problem, which is never quickly doable, even if we could increase utility by investing more time. Instead we need to make a trade-off: grabbing the general structure of the problem quickly and in a very first quick evaluation do the EIMA steps mentally. Given the limitations of the human mind (and short-term memory) one MAPDEK loop with 6 steps, as well as management and resource interface are about appropriate.

Supported with pen and paper or a white-board we can then create a first sketch of an automation problem and categorize the sub-problems along the EIMA approach. We can assess the level of automation along this loop and compare the current state with available machine capabilities. Which will result in a proposal on automation potential (utility of EIMA) in each certain scenario (generality of EIMA).

With the *proposal* on automation potential, EIMA recognizes that there is rarely a “best” automation solution due to two aspects: 1) a lack of commonly accepted utility function for the given scenario and 2) the dynamic influences on the automation system from the environment. Instead, if we can quickly consider the feasible alternatives for implementations, then this is a benefit already.

Subsequent to the automation proposal, all modeling with (potentially EIMA-based) modeling or development tools will be refinement. These details will need development tool support, tool boxes for the machine capabilities, simulations, and metrics. These subsequent steps are not considered here.

The chosen scenarios Two scenarios were selected in a way so that they are different from each other along the scenario properties listed above. One example will be elaborated in detail for depth, the other one will be briefly shown at the end of this chapter for generality.

The main scenario of this chapter is about enabling a management system for QoS end-to-end (E2E) links in GEANT 2, the European research and education network that is composed of the national research and education networks (NRENs). Here, the establishment, change, monitoring and termination of such QoS E2E links is to be automated. Each end-to-end link will cross multiple NRENs and may be composed of pieces of different underlying layer-2 technologies. It is important, that the individual NRENs keep control on what links are routed through their networks for the matter of cost. For this scenario, a proposal on automation exists, that uses a distributed approach. It is described in [Yam09], and is based on a collaboration scheme for chain services described in [Ham09]. We will use Yampolskiy’s proposal as a start for automation in this scenario.

The second scenario is update management. The author of this thesis has dealt with update management in actually two update management automation projects. Before EIMA starts, in both cases we can assume that some of the devices will have individual update mechnisms already. The challenge is therefore in coordinating updates in each of both scenarios:

1) Update managent in “smart home” environments. In this scenario, we foresee the problem, that the number and heterogeneity of electronic appliances in households will rise. We can foresee electronic

locks, window and blind mechanisms, heating control, power metering, washing machines, fridges, central display and controllers, etc. In such scenarios, there will be the need for updating the firmware of the devices to introduce new functionality, improve compatibility and also to fix bugs. This was tackled in an industry project with Siemens in 2005 to 2006.

2) Update management for VMs and hosts in an IT appliance using server virtualization. Here, the scale of VMs demands for automation. At the same time, the new capabilities make it possible to use rolling updates on a set of servers without externally observable service outage. This was tackled in an industry project with Fujitsu Siemens Computers in 2008-2010. We considered Linux und Windows guest operating systems and integrated their individual existing update mechanisms.

Both scenarios are different, as in the first case, we have a certain (complex!) automation proposal to review, and in the second one there is no such previous automation proposal. The first case involves a wide area network and a heterarchy of network service providers with relatively few resources to be managed. The second one is about many more resources to be managed, but in each household there is only a small set. In addition, the resource type is different, the tasks are different and there is no automation proposal to review.

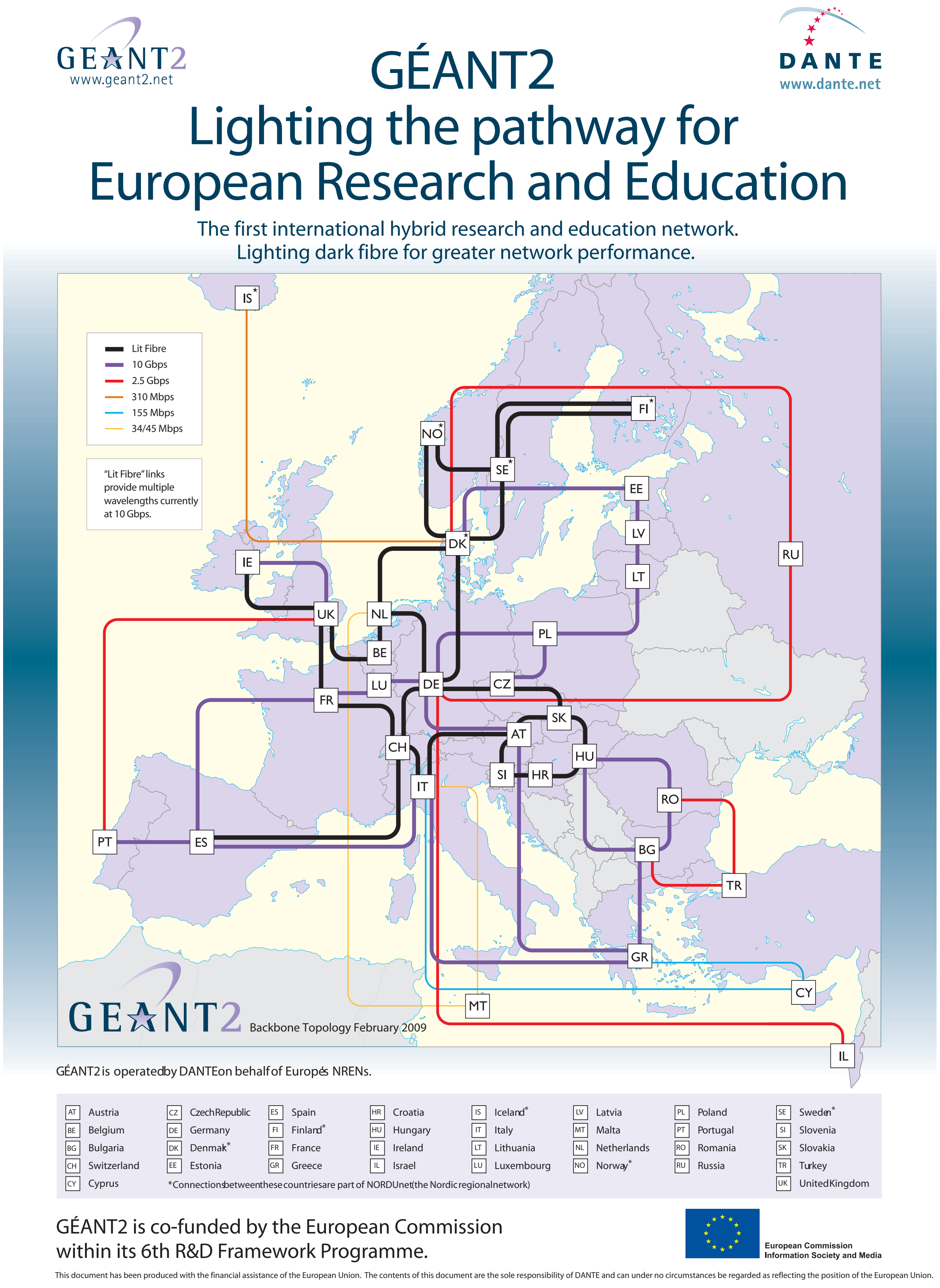
Both are promising scenarios in terms of automation potential: The first one replaces a process (establishing a QoS E2E link in GEANT2) that took weeks before the introduction of automation. Most of this time is spent on communication of people and negotiations. The technical implementation of the link is much less time-consuming. The need for automation also arises from the number of E2E links in GEANT2, and the number of changes in network topology.

In the second scenario, utility of automation with the EIMA approach can be seen in the large number of resources, the multiple combinations of resources in one household, and the dynamicity of new functions as well as bug-fix releases. The need for automation also arises from the issue, that customer acceptance for manually updating smart home electronics will be very limited. But at the same time, vendors have an interest in reduced time-to-market which will increase necessary bug fixes as well as functions added once a certain product is already in the field. This second scenario also considers reuse of knowledge: In fact, we could reuse much of the knowledge gained with update management in smart homes and apply it to the VM server example. Despite IT resource inequality, knowledge transfer worked in this case.

7.2 The automation scenario “Automated E2E QoS links for a WAN across providers”

The following sections will investigate the main scenario in more detail and show, how EIMA is applied to propose feasible automation. The goal in this automation scenario is to automate the setup and termination of end-to-end links with a certain guaranteed quality of service (QoS) in the European research and education and network GEANT2, see Fig. 7.1 for the topology as of Feb 2009. For example, if a dedicated end-to-end link from a high-performance cluster in Barcelona was needed to another one in Munich with a bandwidth of 10 GBit/s, then it should be possible to create this 10 GBit/s link along a certain path, that offers this bandwidth as a (spare) resource.

Before the automation project, the link was set up by human to human communication among NRENs. One side started the request and asked the neighbouring NREN, which again asked the neighbouring NREN until the goal NREN was reached. Administrators configured WDM multiplexers to set up the path, or used other techniques that their local network equipment could handle. This way, the whole end-to-end link was built step by step, with resources being ordered, bandwidths being negotiated, wavelengths configuration data exchanged and so on. The typical duration of setting up a link was in



GÉANT2
Lighting the pathway for
European Research and Education

The first international hybrid research and education network.
Lighting dark fibre for greater network performance.

GÉANT2 is operated by DANTE on behalf of Europe's NRENs.

AT	Austria	CZ	Czech Republic	ES	Spain	HR	Croatia	IS	Iceland*	LV	Latvia	PL	Poland	SE	Sweden*
BE	Belgium	DE	Germany	FI	Finland*	HU	Hungary	IT	Italy	MT	Malta	PT	Portugal	SI	Slovenia
BG	Bulgaria	DK	Denmark*	FR	France	IE	Ireland	LT	Lithuania	NL	Netherlands	RO	Romania	SK	Slovakia
CH	Switzerland	EE	Estonia	GR	Greece	IL	Israel	LU	Luxembourg	NO	Norway*	RU	Russia	TR	Turkey
CY	Cyprus													UK	United Kingdom

*Connections between these countries are part of NORDUnet (the Nordic regional network)

GÉANT2 is co-funded by the European Commission within its 6th R&D Framework Programme.

European Commission
Information Society and Media

This document has been produced with the financial assistance of the European Union. The contents of this document are the sole responsibility of DANTE and can under no circumstances be regarded as reflecting the position of the European Union.

Figure 7.1: GEANT topology as of February 2009, source: `http://www.geant2.net`

the range of weeks to months due to phone calls and e-mails being exchanged among administrators without a certain process.

7.3 Automation analysis and general improved system design proposal

The original requirements for a more automated solution than this manual process included the following requirements:

- no global knowledge: For the NRENs each managing their own piece of GEANT2, they were reluctant to share topology information with other NRENs or a central authority. The automation solution therefore demanded for distributed topology knowledge so that service providers could keep their secrets and even propose worse QoS to others than they actually could provide. With the E2E link being a "chain service", this means a service being provided by a concatenation of service providers, this heterarchic organization was to be kept.
- no/little central control: For the reason of being a network of equal partners (heterarchic organisation) the solution demanded for one that did not introduce a central instance, which decides on routing of the links in the network or which is responsible for fixing the links. Instead, a scheme of distributed control should lead to distributed responsibilities of the NRENs itself.

Yampolskiy in [Yam09] designed a source-routing based protocol that essentially automated the manual approach done before: Aggregating the E2E link in a step by step fashion by communication of neighbouring service providers. In parallel, a monitoring service is composed, that is also composed of link pieces.

When analyzing this solution, we can review the decisions made there from an automation point of view:

- many bad automation properties of the problem
  - distributed knowledge
  - heterogeneous router infrastructure
  - no hierarchical control structure
  - unclear incentive scheme from transitting traffic (unclear utility function)
  - link requests are not planned, but occur without prior notice
- some good
  - small set of resources
    - 30 NRENS
    - about 400 routers
  - one service only: QoS network link service
  - few resource types to manage: routers, WDM multiplexers

Good properties of original design with respect to automation

- The solution found there meets many of the requirements stated there. While a few of them can be doubted (such as scalability), within the set of constraints of the original requirements, it is a feasible solution.
- As the automated protocol is close to the original manual way of setting up links, there is a high likelihood for acceptance with those administrators, that have understood the original way by

heart. At least, those system administrators feel “at home” with this type of automation and their remaining control.

Bad properties of original design with respect to automation

- On the bad side, the solution is very close to the previous manual workflows. This may ease adoption of the protocol, but as we learnt from Sheridan (see Chapter 3 Task Analysis), sticking to the original process is seldom a good idea when automation is the key goal.
- The solution only considers a single E2E link at a time, and does not consider the interactions between the links.
- There is no global optimization of routes. As soon as a route is found that fulfils all QoS constraints the protocol stops and uses that route upon approval. Other links are not rerouted to “make room” nor are they rerouted to distribute traffic more evenly. Instead of global rerouting, there is only provider-internal rerouting, which can be done without notification of a central authority.
- There is also no global resource planning/optimization or capacity planning.
- The scheme has high organizational complexity with many potential delegations and proxying. In general the value of these delegations and proxying is unclear in practice, as it complicates matters enormously by introducing layers of indirection.
- Each of the 30 NREN operator needs to implement all 31 workflows and provide staffing for them, see Fig. 7.4 for an illustration of the resulting complexity. The distributed scheme keeps service providers in control, but it is mandatory control, not optional control. They *have to* do those workflows locally and manage their local parts of the links and their local workflows instead of *being able to intervene*, which would be optional management. EIMA’s overall goal is to reduce necessary management, but leave optional management influence where desired.

  Indeed, five countries (Finland, Sweden, Iceland, Norway, Denmark) have already combined their infrastructure in NORDUnet as a single NREN covering all five countries (see GEANT topology in previous figure). This is good proof, that the ineffectivity is also recongized with GEANT and certain members try to reduce process complexity.
- While there is no global knowledge in this scheme, any partner can ask any other partner with certain fake requests and this way explore other partners topologies. The true benefit of topology privacy of this non-global scheme is therefore questionable.
- The method still takes long because of the many human interventions.
- A decentral scheme would have no central place where the system state can be monitored (e.g. progress along setting up routes, or the state of existing routes). At the end of his work, Yampolskiy recognized this as one major issue in decentralized schemes: monitoring is hard! With “I-SHARE” and E2EMon, central information systems for monitoring the progress of the distributed processes were introduced, composed of central and decentral components and therefore breaking up the decentralized nature of the original design.
- There is no global automated incident management across AS’s. Only within AS’s. Eventually, a monitoring service will detect link failures and even notify the NRENs that are most likely to cause the errors, incident management once again would fall back to human work.

System design proposal to improve automation All in all, Yampolskiy designs a system that meets most of his stated requirements, but he designs one that misses much automation potential in terms of effectivity and efficiency.

From the point of view of the author of this thesis, the distributed design misses the most critical point in releasing automation potential: Questioning the requirements and rephrasing them to allow more automation!

Therefore the proposal that is presented here may not have been a valid solution for the author there in the scenario and with the acceptance that was present there. The reader of this thesis should therefore see the following alternative automation proposal as one that would have to be reconsidered by the stakeholders. It is technically feasible, but the scenario-specific cost-benefit analysis would have to be made there.

As a matter of past experience we know that central, hierarchical schemes with global knowledge are a good basis for management and a good basis for automation. Their special features are good efficiency, traceability along the hierarchy, and effectivity.

If we changed the requirements slightly, which would probably be accepted by the stake holders, then we could:

- Introduce global knowledge instead of the distributed topology discovery that happens each time in Yampolskiys approach with a tree-based search.
- Allocate almost all tasks to a central control instance (a system being run at and by a network operations center, NOC, which is a known concept and also exists in GEANT2).
- To keep local staff of SP's in the decision loop (and therefore in control), have them vote on QoS E2E link changes and thereby have them select a most appropriate alternative from a list of choices.

  Here, we make use of delegation at one single point in the MAPDEK scheme: "Decide". This is why the Decide step has originally been introduced: keeping a point to monitor the system and intervene. This mode can be selected with a control on the level of automation.

All these considerations are made before EIMA actually is applied.

However, now with EIMA in mind, we can sketch a control system for the same automation problem along the EIMA structure with one global MAPDEK loop. The new system would be similar to a control system: The observed state would be the state of all E2E links and their reported QoS, both as reported by a monitoring system. The desired state would be the list of all E2E link requests and their desired QoS, both from a central link request repository. A discrete time event controller would periodically try to bring the observed state close to the desired state. This would be the outermost loop, a loop which can establish new E2E links, change links for all reasons (changes in topology, changes in the requests), monitor links, and also terminate E2E links. This works as all these cases are simply differences between the desired and observed state.

In the subsequent sections, the details will be described along the EIMA approach: task analysis, MAPDEK loops, function allocation and machine capabilities.

7.4 Step 1: EIMA task analysis concepts for analysis and redesign of tasks

EIMA task analysis is structured along three artifacts: a role hierarchy, a task decomposition tree, and a automation information table. So we use these artifacts here to structure this section.

Role hierarchy

Original design In the original design, the roles are assigned to organizations/people, based on which NREN a link request originated from. Therefore, all role names are relative, such as “SP-Domain providing service”, “SP-domain requesting service”, “Routing responsible”. As there are no central roles, quasi-central roles are associated based on the link requests with roles in the NRENs.

Analysis Overall, due to the lack of central entities, the resulting complexity for each NREN’s local view is manageable only as long as links originate in that domain or terminate in this domain. The organizational complexity for *transit* links however is dramatical! Also the complexity from a global point of view is high. By design, there is no real hierarchy of roles in the original design, as one of the goals was to deliberately not introduce a hierarchy, but keep the heterarchy.

Redesign With EIMA, we give up the peer-to-peer-style, heterarchic design. Instead, we will introduce a role hierarchy, see Fig. 7.2. A central QoS E2E link routing manager role will be responsible for all of the tasks, with exception of: NREN network monitor, NREN network configuration manager, NREN routing decider, all of which will exist for each service provider. The central QoS E2E link routing manager role will be responsible for route analysis, global rerouting, incident management, as well as planning tasks. The Requester role will be an end user entity which demands an E2E link.

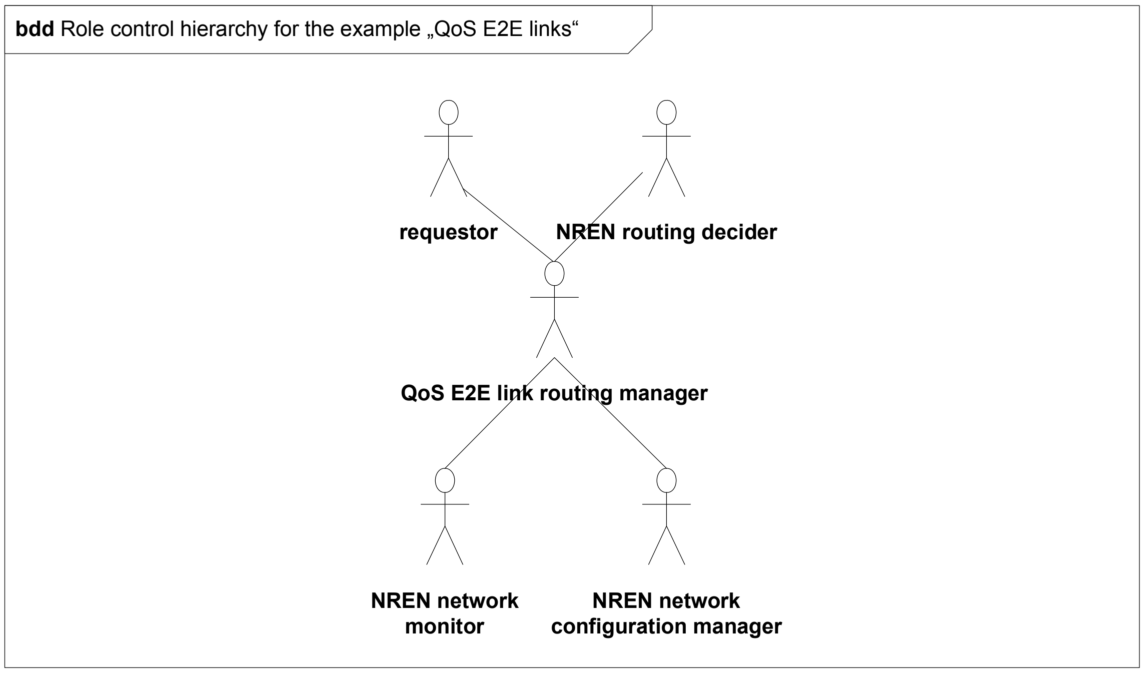

bdd Role control hierarchy for the example „QoS E2E links“
requestor
NREN routing decider
QoS E2E link routing manager
NREN network monitor
NREN network configuration manager

Figure 7.2: Proposed role hierarchy in the QoS E2E link scenario

Task decomposition tree

Original design Task analysis already happened in the original work. Based on the manual collaboration scheme and with requirements taken into account, a task scheme was proposed there. The original task descriptions (with additional input from the author of [Yam09], as the thesis itself does not include all workflows) was stated as 31 BPMN workflows:

- 4 SLM workflows (top-level) tasks along the life-cycle of the link
  - order QoS E2E link
  - monitor QoS E2E link
  - change QoS E2E link
  - terminate QoS E2E link
- 14 intermediate layer workflows as helpers
- 13 basic workflows (bottom-level)

The descriptions of these workflows, despite being presented in mostly tabular form take up about 100 pages, which itself is an indicator for complexity. This complex scheme describes in the workflows, how an E2E link is put together based on piecewise aggregation of partial services and a depth-first search for solutions matching the QoS constraints. Unfortunately, there is no general overview on how the tasks interact with each other (no explicit task hierarchy). The protocol described by the workflows gets even more complex when considering the included delegations, proxying, separate monitoring service instances, and recursive calls in the workflows when tasks are delegated to other partners in GEANT2.

For better comprehension, first the tasks were put into a task hierarchy, see Fig. 7.3.

Analysis The overall structure in this figure can be aligned with EIMA in two respects: 1) Inputs on the management interface (requests to set up new links, change links, terminate links) are associated with the top-level tasks. This is in alignment with EIMA.

And 2) The basic tasks are all *closer* to the resources. This also matches with EIMA. However, here, they are not refined down to actual management interactions to actual network components. Instead, it can be seen, that the basic managed items are subscription lists, resource reservations/orders, and not the actual network elements. So, below this ITSM support system, a lower level resource management system—this means a network management system (NMS) in this case—will be needed. Closed-loop control and actual automation (automatic configuration of network components) therefore depends on the capabilities of the NMS.

The original design is based on the strategic behaviour of the NRENS. To enable the NRENs to behave strategically, matters are not objectified (there is no notion of a cost model for example), instead they are negotiated each time. Negotiations are enabled implicitly by e.g. letting service providers hide internal details or by service providers promoting to the outside world a lower quality than they can really offer.

Overall, the scheme suffers from a larger number of helper workflows and basic functions though. This scheme is overly complicated with too many options for delegations. Its limited automation with much human involvement leaves much automation potential.

The original design is a complex layer for coordination, network planning and negotiation on top of actual network management. Despite its organizational complexity, it lacks functionality like fair/optimal resource use considerations or optimization for multiple links as well as a utility function. Decisions are not objectified.

Redesign The redesign was intended to simplify the design, objectify decisions, allow optimizations and centralize link routing. To keep the NRENs in, we intend to make proposals to them about link routing, which they can either confirm (prospected to be the usual case) or deny.

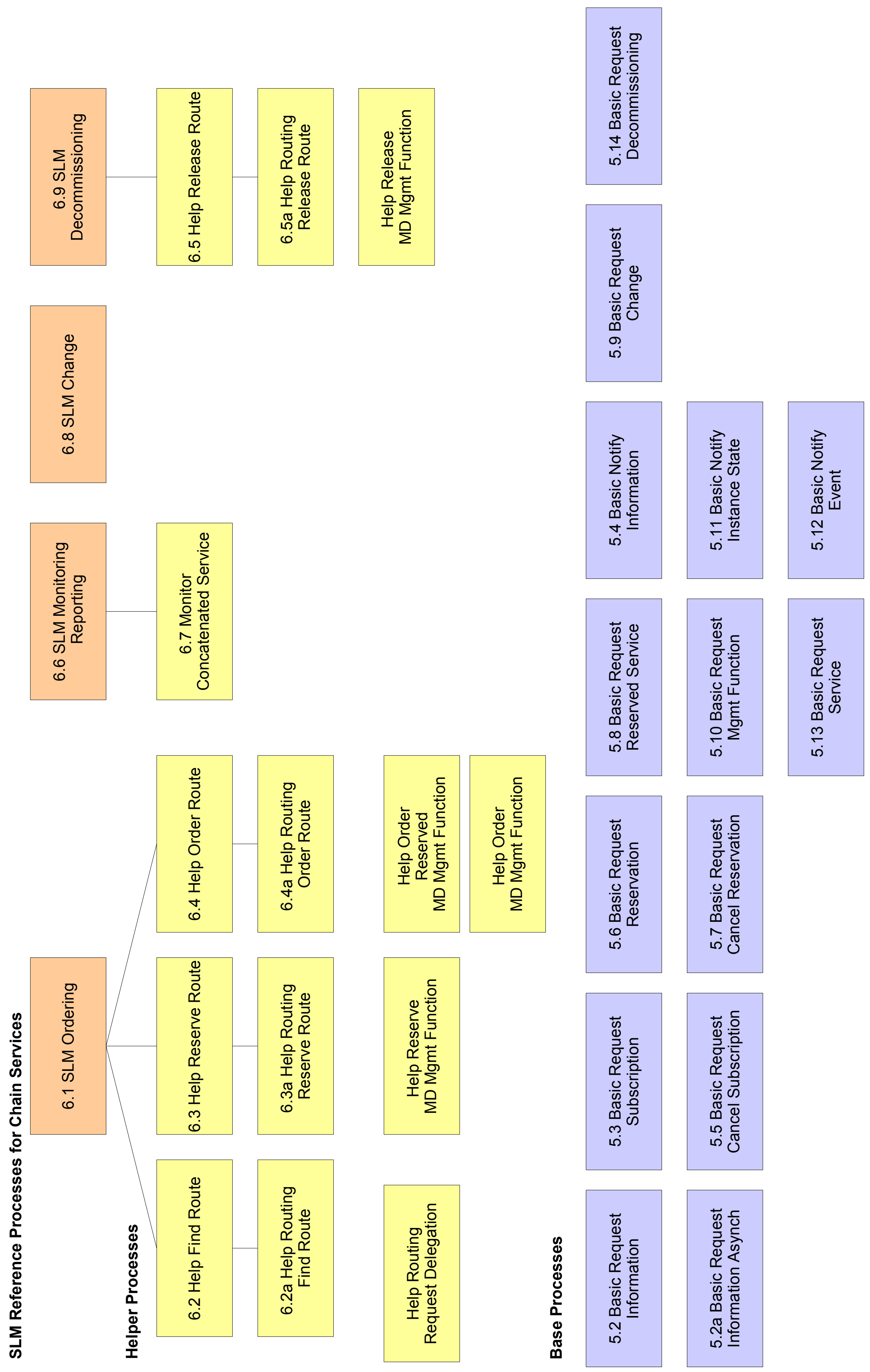

SLM Reference Processes for Chain Services
6.1 SLM Ordering
6.6 SLM Monitoring Reporting
6.8 SLM Change
6.9 SLM Decommissioning
Helper Processes
6.2 Help Find Route
6.3 Help Reserve Route
6.4 Help Order Route
6.7 Monitor Concatenated Service
6.5 Help Release Route
6.2a Help Routing Find Route
6.3a Help Routing Reserve Route
6.4a Help Routing Order Route
6.5a Help Routing Release Route
Help Routing Request Delegation
Help Reserve MD Mgmt Function
Help Order Reserved MD Mgmt Function
Help Order MD Mgmt Function
Help Release MD Mgmt Function
Base Processes
5.2 Basic Request Information
5.3 Basic Request Subscription
5.6 Basic Request Reservation
5.8 Basic Request Reserved Service
5.4 Basic Notify Information
5.9 Basic Request Change
5.14 Basic Request Decommissioning
5.2a Basic Request Information Asynch
5.5 Basic Request Cancel Subscription
5.7 Basic Request Cancel Reservation
5.10 Basic Request Mgmt Function
5.11 Basic Notify Instance State
5.13 Basic Request Service
5.12 Basic Notify Event

Figure 7.3: Workflows by name, specified in [Yam09], but now arranged hierarchically related to a task decomposition tree. The basic processes have not been associated with the respective helper processes to avoid clutter. Each of the rectangles translates into a BPMN diagram.

In the new design, we keep only the 4 top-level tasks and will break them up in a different way, that replaces negotiations on routing and proxying with a global link routing logic. We imagine, that all tasks will later be refined down to the network element management level of abstraction. A detailed additional task analysis is not necessary here, see the MAPDEK loop in the next section.

Automation information table

Original design Inside the workflows, the tasks were already enumerated, the alternatives (symbolized by diamond marks) in the workflows were not, but they are simple decision tasks. The tables in [Yam09] that contain task descriptions can be seen as input for the automation information table. Even though it lacks the information items structured in a way as defined in Chapter 3, automation information table.

Analysis The original workflows will not be reused.

Redesign Such a detailed investigation is not necessary in identifying automation potential in this case.

7.5 Step 2: EIMA MAPDEK loops for analyis and loop design

Original design The original design was stated in terms of BPMN workflows, not in terms of loops. The BPMN workflows were structured along the service life-cycle, where at first a service is set up and provisioned, then used and monitored, then eventually changed, and finally terminated.

The workflow were instantiated one time for each NREN partner, resulting in tremendous complexity as shown in Fig. 7.4.

Analysis We first tried whether the original workflows fulfil the MAPDEK loop structure, and indeed they do. So it would be possible to create MAPDEK for each workflow and we could transform all the workflows to loops, but after the result of task analysis above, the task structure will be different anyway. Because still, the overall architecture would not match EIMA's implicit idea of an observer-controller loop pattern with a controller doing reconfigurations to match the observed state of the network with the desired state.

Redesign For this reason, we apply the MAPDEK loop pattern and a hierarchy of MAPDEK loops is created, with resemblance to the the role hierarchy. This results in a control system for this overall task and is shown in Fig. 7.5.

The central loop is a loop that analyses two inputs: desired state (the list of requested links and their QoS requirements, as input by requestors via the management interface of the loop) and observed state (current network topology, currently set up links, QoS properties of these links) received as input from the resource interface of the loop. The central element of this loop is the analysis step. This step runs an algorithm, that places all links into the network, so that as many of them of as possible are implemented, and routed in a cost-effective way around network failures. In more detail this means:

- a new link needs to be routed through the network

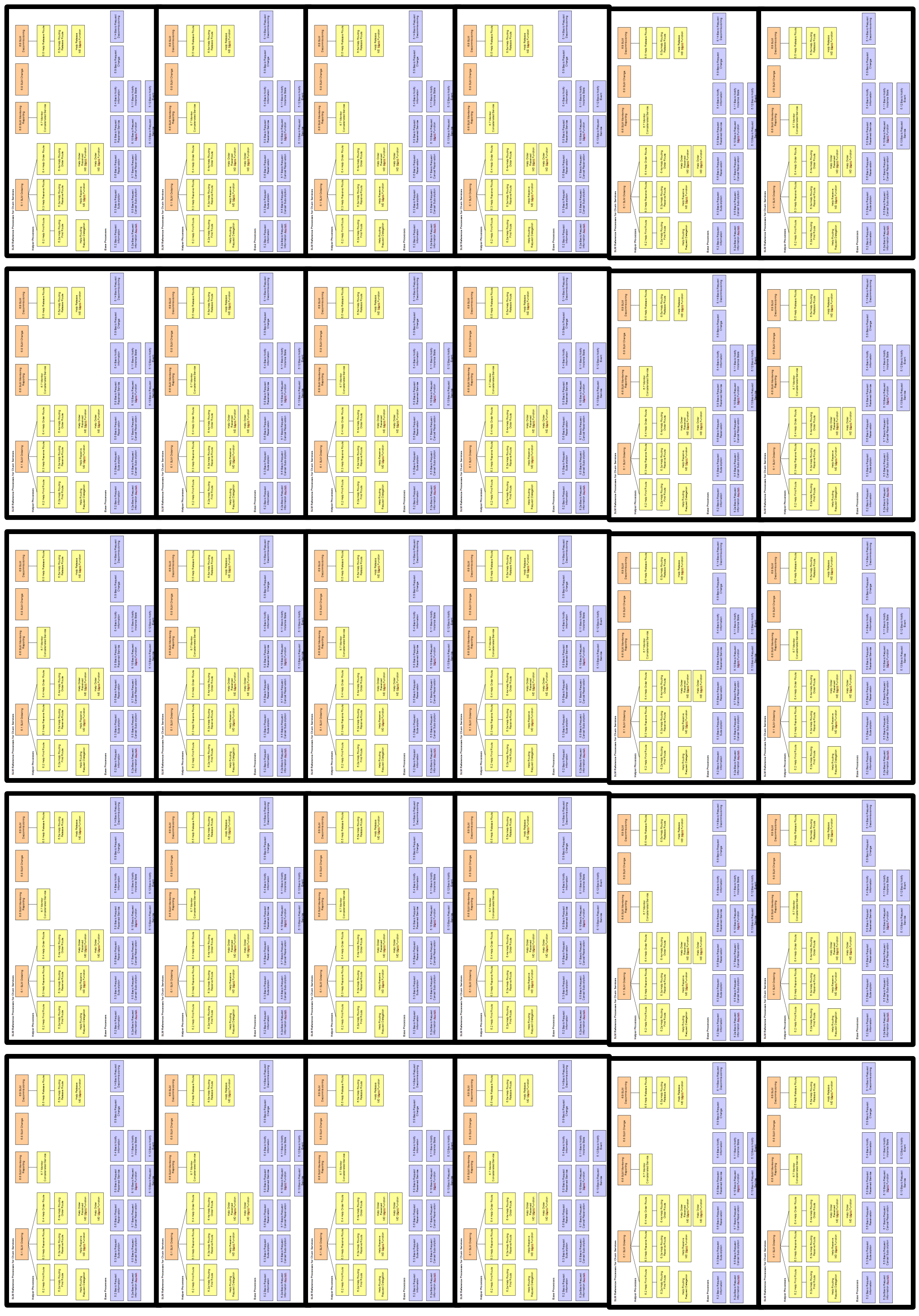


Figure 7.4: Illustration of the resulting process complexity, when Yampolskiys process scheme is implemented at 30 NRENs. Links between neighbouring NRENs were left out on purpose.

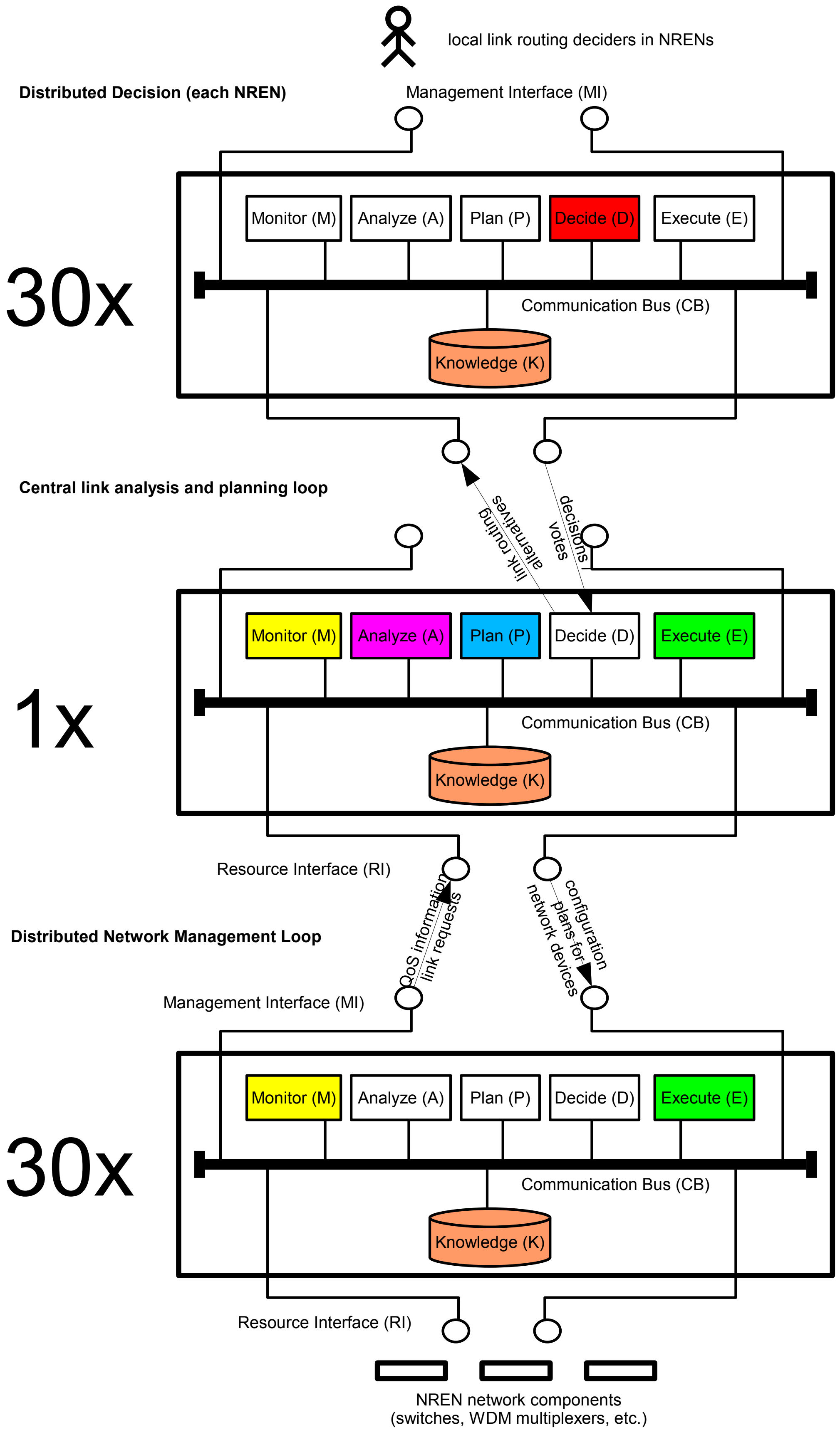

local link routing deciders in NRENs
Distributed Decision (each NREN)
Management Interface (MI)
Monitor (M)
Analyze (A)
Plan (P)
Decide (D)
Execute (E)
30x
Communication Bus (CB)
Knowledge (K)
Central link analysis and planning loop
link routing alternatives
decisions votes
Monitor (M)
Analyze (A)
Plan (P)
Decide (D)
Execute (E)
1x
Communication Bus (CB)
Knowledge (K)
Resource Interface (RI)
QoS information link requests
configuration plans for network devices
Distributed Network Management Loop
Management Interface (MI)
Monitor (M)
Analyze (A)
Plan (P)
Decide (D)
Execute (E)
30x
Communication Bus (CB)
Knowledge (K)
Resource Interface (RI)
NREN network components (switches, WDM multiplexers, etc.)

Figure 7.5: New MAPDEK scheme for this scenario

- changing links may need to be rerouted
- released links can be removed

The algorithm would take care of avoiding oscillations in the routes. It would also take care of creating too many changes in each round, as each change will be likely to cause a glitch or disruption in the link E2E service.

Below the central loop and residing with each NREN, there will be local agents that monitor the local infrastructure and provide this information to the central loop. These loops will also contain an Execution component to change the configuration of the network elements. These loops are the interface to the individual NREN's network components.

7.6 Step 3 and 4: EIMA function allocation and machine capabilities

Original design The original design included machine capabilities only in terms of message exchange as a protocol. The only actual machine processing of actual information is a depth-first search in a distributed fashion with aggregation of QoS parameters and deciding whether the overall QoS objective has been met. Instead of a reliance on more automation, it includes many points for humans to intercept and interpret the information.

Analysis The original concept is stated in terms of workflows, which contain roles. Unfortunately, the roles have not been mapped to human roles and machine roles. Instead, there is an implicit assumption on these roles about function allocation. We assume, that they are meant to be mostly human performed roles.

This way, overall protocol design would be a human-executed protocol (or a process). Not a machine driven protocol. Still, the original work does not make this decision explicit.

With knowledge in more machine capabilities in such a scenario, we believe that running the original depth-first search with first fit for each workflow is not good enough to utilize the automation potential inherent to this scenario.

Redesign: Machine capabilities in MAPDEK scheme With EIMA, we introduce a single global loop that does global optimization of the links. For this overall task, we look for existing capabilities in a catalog of machine capabilities related to this matter.

Index 0: AutoITIL

none.

Index 1: Existing whole loops

Full automation with existing methods unlikely, even though there are ideas, as we know that there are machine capabilities for QoS routing, if knowledge is global. In fact, a cross-domain QoS-aware routing algorithm would be perfect and standardized. However, BGP with QoS is not link-specific, and traffic engineering with MPLS is not applicable to such optical links.

So, whole loops/whole methods are unlikely to automate this scenario properly. The only matching loop is a solution set creator loop or ranker loop that composes the link routing alternatives and escalates them to the NRENs to vote on their favourites.

Still, we will have to look into the MAPDEK steps themselves, as well as the two interfaces for management and to resources, and the communication bus.

Index 2: Management Interface

At this interface, there are essentially only two information items: 1) link requests and 2) decisions on link proposals. We can use any interface we like for this matter, a graphical user interface on a website would be a feasible choice. But also a command-line interface would be appropriate, so that automation on the NREN side would be easier. The GUI can then be built on top of it.

Index 3: Monitoring

Monitoring here means to gather the data about topology and existing QoS E2E links.

In our proposal, this would be centralized with the information being provided by agents at distributed monitoring sources. The overall link state would be aggregated from distributed sensors in the same way as for other successful GEANT projects, e.g. PerfSonar.

In contrast to the original design, we would use a fix entity as a QoS monitoring service provider. This way, we do not have to decide on one, which reduces complexity. And we do not have to aggregate the information from piece parts, that formed individually for a link to create a private entity behind a common proxy. It would have to be investigated, how existing PerfSonar can be adapted to this new type of QoS E2E links.

Index 4: Analysis

Routing the links through GEANT2 would be centralized. The link routing algorithm would be based on a global topology model. With the whole topology known, we could achieve a fair placing of links in net-wide QoS routing. An algorithm would periodically compute best routes (new link routes) for the overall set of E2E links and propose a new configuration with an artificial cost for change to work around oscillations.

To find the algorithm, we can use flow algorithms in graphs/multigraphs. Cost-minimal placement of flows is a known problem, where we are confident good algorithms exist already, e.g. in [AMO93].

For such global optimization across all concurrent link requests we need a utility function, that rates how willing the providers will be to accept this overall link routing proposal. Here, we can rely on economics to express the willingness in terms of gain/profit and loss and therefore have a good metrics. In addition, a set of constraints could rule out unwanted solutions that are a matter of policies which cannot be expressed in the utility function.

The main benefits of the new scheme are that it makes decisions more objective, saves global resources by optimization, limits the cost for each provider and add a cost benefit model, which is especially important for transit traffic.

Index 5: Planning

Planning would be centralized as well. So far, the analysis routine in the previous step as described is reactive only as the topology model is sensed each time in an ad hoc manner and link requests come ad hoc.

The idea for planning is therefore to implement a dynamic model (a calendar of expected changes) of the link requests and topology, run the algorithm on a time instance. This way, we can move from reactive to a proactive scheme. The global calendar enables planning future demands and future resources and therefore will allow running the link routing algorithm for future points in time. To gain acceptance, calendaring would be for documentation purposes first, and then would be introduced in a step by step manner from the reactive scheme to a proactive scheme.

This way, we can also plan the transition from the current setup to the new setup and derive an action list (a plan) for each NREN's network configuration manager, enabling simple resource-level and global forecasts.

Index 6: Decision

The NRENs demanded for staying in control. For this very reason, decision making about link routing would be distributed: The central analyis and planning routine only makes proposals to the local NREN routing deciders, but collects their votes. So once a new global configuration has been calculated it would have to be voted on by the deciders of the corresponding NRENS with a voting mechanism yet to be defined. This scheme makes it necessary, that the original algorithm already does a fair but also performant distribution of the links to gain many confirmations (votes in favor of a common solution) and few refusals (votes in favor of different solutions).

To reduce refusals in the decision we would need agreed upon metrics for fairness. This incentive scheme for NRENs can be based on a donation scheme or bidding (related work by University of Zurich, Burkhard Stiller's group on bandwith pricing and a P2P style market/stock exchange for ISP's for link bandwith). The details of the decision scheme and voting mechanism are to be determined based on policies.

This decision phase would be an excellent one for the application of automation levels, so that NRENs can for example auto-confirm certain proposals (vote in favor of the top entry). In every case, the voting scheme would include a decision deadline mechanism to ensure timely replies to keep overall process time low. Otherwise a single NREN could still block the whole mechanism.

Index 7: Execution

Finally, we need to configure actual resources, based on the action plans for the NRENs. At this place, we can use whatever management interface the corresponding network components or local network management system will offer. It is likely, that SNMP access will be available or access via a device centric proprietary interface. Everything which is programmable in network components such as optical switches, WDMs, etc. should be programmed. However, there may remain a few physical tasks such as cabling work, which will finally be attributed to people with a clear list of instructions.

Index 8: Knowledge

For all knowlege we would use a central database, that contains the data on: topology, link requests, link costs, current link routing, as well as a history of the aforementioned entities.

Index 9: Resource Interface

We assume at least read access to all network devices (in the same way as in Perfsonar). This would enable network configuration and link monitoring. Write access is optional and could be included in higher automation levels to choose from.

7.7 Resulting partial projects

With EIMA, we have transformed the original problem into an automation problem. This automation problem can be solved by solving the following partial problems, that have been derived from using EIMA:

Upper loop

Task	create a requestor interface, where users can order, monitor, change, and terminate links
Responsible	software engineer, human factors engineering specialist

Central loop

Task:	create a utility function for the links, measure cost and benefit
Responsible:	economists

Task:	central routing logic
Responsible:	mathematicians for flows in networks
Input:	links, topology, constraints, utility function from economics
Output:	optimal, fair placement of links

Lower loop

Task:	create a common API for network resources to be configured
Task:	create a common API for links to be monitored
Responsible:	network management specialist, optional: PerfSonar developers (adapt PerfSonar to ebable E2E QoS link monitoring)

7.8 Conclusion for this scenario

By applying EIMA in this way, we gain the desired property that a true automation effect with improvements in effectivity and efficiency will arise. Provided that the partial projects can be solved and deliver their individual solutions, we have the measures to compose the overall automation system. Fig. 7.4 (before) in comparison to Fig. 7.5 (with EIMA and a reorganization) well show the achievements possible in reducing scale and heterogeneity of processes. Tab. 7.1 wraps up the comparison.

7.9 Discussion of the alternate proposed solution

Discussion of changing requirements In the presented solution, we changed the requirements and created central instances, which enabled global roles and a global task hierarchy. This clearly is against the original requirements.

We did so for two reasons:

1) Also Yampolskij has introduced a global component named I-SHARE to exchange link status information, monitoring data, etc. He noticed, that it is easier to implement and keep consistent with a central, global system. We used this inconsistency in his approach to question the requirements in the original design.

2) In addition, global monitoring systems like Perfsonar, which have been introduced in GEANT and are used, leave a different impression with the author of this thesis than the requirement for true network topology privacy with the original NRENs.

Table 7.1: Comparison of the two proposed solutions

Proposal in [Yam09]	Proposal in this work
ad hoc along a manual process	along the MAPDEK scheme, and therefore structured along cognitive sub-tasks
any partner can stop overall progress	a central authority takes care of the progress
documented	optimizing in the maturity level scheme
mainly included the communication work: infos, notifications, reservations, subscriptions: all this is just moving data back and forth in the end	left out all the communication work, which is just a normal information system, instead, quickly found the cognitive tasks and concentrated on them!
simple math: aggregating QoS properties with a min() fashion for bandwith, or additive (for delays)	added new functionality: optimized resource allocations for multiple links along a utility function, constraints, transition planning. better reuse of existing knowledge and technology
unspecified allocation to humans/machines	awareness for human/machine allocation

Clearly, this modification of the original problem, while also reducing complexity, also provides a better basis to apply EIMA.

The critique on Yampolskij's proposed solution is not only caused by the requirements given to him however, which he could not question or change.

It is rather about his focusing on automation of a "human protocol", which he turned into a "machine protocol" with influences from source routing and depth-first search. In general, this scheme still fulfills Sheridan's critique that too often, automated solutions are created with too close resemblance to the original manual way of doing the tasks. Sheridan instead proposes to abstract from the manual tasks and instead consider a "criterion of satisfactory completion".

Yampolskij creates an information system and a UML-specified protocol, but leaves most of the cognitive tasks to the system administrators. The following list of the cognitive tasks that arise around QoS E2E link placement and are not considered by Yampolskij shall make this more plausible:

- Analysis: optimization of link placement, considering more than 1 link, finding a utility/scoring function for an objective score for a link placement, trading-off between alternative solutions.
- Planning of topoloy and link changes. These changes imply changes in the network components. Ideally these would be coordinated and planned in advance, reducing overall negative effects on the data flows in the network and keep down-times low. This is not trivial and will require global coordination anyway.
- Decision: distributed decision making and finding agreements. The depth-first search does not allow this kind of coordination, as the NRENs have to agree already when the link is being built. They cannot wait for the final routing alternatives and then collectively decide on a best solution.
- Loops: global link repair (in terms of rerouting not within a single NREN, but across NRENS)

Experience tells requirements engineers, that requirements do not always reflect the true will of the stakeholder. When knowing about the solution alternatives, the stakeholder may well change his mind and drop some of the original requirements in the interest of more efficiency and/or effectivity.

A solution idea within the original requirements Yet, to make plausible that EIMA would also be applicable to the original problem and have a benefit, the following list of considerations shall convince the reader, that also for the original problem a distributed solution could be found with EIMA, even if it is not as efficient as a central one:

Clearly, in this distributed case, the central MAPDEK loop and central roles are inapplicable. The new top-level MAPDEK loops would then be per NREN as with Yampolskij's solution. They all would stand on the same level next to each other. They could look the same with each NREN apart from the resource interface. EIMA step 1 task analysis would now investigate tasks within one NREN, as locally there are hierarchical tasks and hierachical roles. This matches the organization in GEANT2.

The following changes would apply in terms of candidates for machine capabilities:

- NREN-local loops (monitoring/execution and decision) would stay as they are. So there is no change involved here.
- The central loop with central analysis and central planning and central knowledge would all three be split up to pieces residing with the individual NRENS.
  - Turning central analysis into distributed analysis: This step contains an optimized link placement of all links. This central algorithm would be replaced by a distributed algorithm executed on NREN local link routing compute components. It is a task for domain experts to find or create such a distributed link routing algorithm. Yampolskij's approach with blending source routing with a depth-first search is a first simple one for each single link without much optimization taking the first match. But I am very confident that a better algorithm could be found that indeed optimizes more than one link at the same time and that copes without the proxying and relaying in Yampolskijs described solution. This would need domain experts like from theory of computer science and mathematics and their publications.
  - Turning central planning into distributed planning: The central planner would need to be replaced with a distributed planner. This should be available in distributed planning, a domain e.g. needed by multi-agent systems.
  - Turning central knowledge into distributed knowledge: Two machine capabilities would enable this transition:
    1) Peer-to-peer networks are a way of decentrally storing large amounts of information (key value pairs) redundantly. This is a way to store something in a distributed fashion but yet to keep the impression to handle one big storage instead of many small ones. It would clearly have drawbacks in terms if privacy, data integrity and manageability, but it would decentralize the knowledge.
    2) Distributed databases would be yet another way to implement this. CMDBs should exist with the NRENs already, a federated CMDB with trust relationships would provide a quasi-central view but keep the data with the NRENs and have them control access to information.

In reaction to critique, that the proposals made here are far too inspecific, we have the opinion that IT management automation engineers for future large systems will need to see the big picture in every of their IT management automation problems. They need to decompose the original problem properly and then delegate the partial tasks to domain experts. In a topic such as IT management automation, ranging from algorithm and bit-level in the most diverse components to the level of economic strategy and business processes, a single person can unlikely cover this whole range of appropriate methods, tools and kowledge.

Application of the catalog of machine capabilities In the example shown, there was a mix of machine capabilities already identified in the catalog of machine capabilities and other ones, so far not listed and categorized there.

The reasons are two-fold:

- The catalog as presented in Ch. 6 does provide an extensive set of machine capabilities but yet for special problems it remains quite incomplete. It should resemble something such as "world knowledge for IT management automation engineers", but this already makes clear, that it needs to be maintained, extended and updated. Within the candidate sets the atuomation engineer still needs a good intuition.

  For new problems we eventually need to identify new solution externally from the catalog and then add them to the catalog so that future runs can profit from this knowledge. In fact, this is not much different than with the role model of EIMA tools: applications such as Matlab or AutoCAD are structured in a similar fashion. They are provided with a certain common base set of general capabilities but for special problems they profit from domain-specific toolboxes, plugins, extensions. The same applies to the catalog of machine capabilities.
- The filtering views applied when searching the catalog are shown in Fig. 6.2. This view creation is easy in a tool, but hard in a document such as this thesis, as this view creation would have to be performed over and over again. Just to document the finding of machine capabilties for 10 tasks would need 10 tempory candidate sets, each of them filling a page. We consider it unnecessary to make this explicit, when the general scheme has been shown before. And as also said before: EIMA will not replace the automation engineer, but support him. We will have to rely on human intuition and judgement to then select the methods that seem most appropriate and applicable. There may even be methods performing certain tasks in an equally good manner.

7.10 Other scenario: Automated update management

During the development of the EIMA method in the course of writing this thesis in the last four and a half years, the author of this thesis with support from colleagues and students has investigated several IT management automation projects. The scenarios were used to test-case and apply previous automation attempts to develop and improve EIMA as a method. The following list of three projects in the next three paragraphs shall make EIMA's universality more plausible but also give an insight into its development.

Policy-based Update Management in Smart Homes, Siemens (Sep 2005 to Mar 2007)
In an industry project with Siemens, Corporate Technology, Software & Engineering group, we investigated policy-based update management for smart homes, see [KWDT06] for the project report.

The intended use case was to automate the update management in homes, that have many smart interacting electronic devices (home appliances, communication and entertainment technology, alarm systems, building services such as heating and cooling). For reasons of short time to market and frequent additions of new devices, it was foreseen, that updates will be likely and necessary. The aim was to solve the involved compatibility of updates and an optimal new configuration with knowledge of compatibilities and features of the individual devices.

To not bother the home owner with too many low-level configuration decisions, update decisions should be based on policies, that can be preconfigured by the device manufacturers, but also influenced by the customer.

As a result, we created a concept for this use case and a software demonstrator, see Fig. 7.6 for a system overview. It was written in Java using the JGraph library and a simple tree search algorithm by the author of this thesis and Diana Weiss (see Fig. 7.7 for an intermediate result), and a policy engine implementation (by Vitalian Danciu).

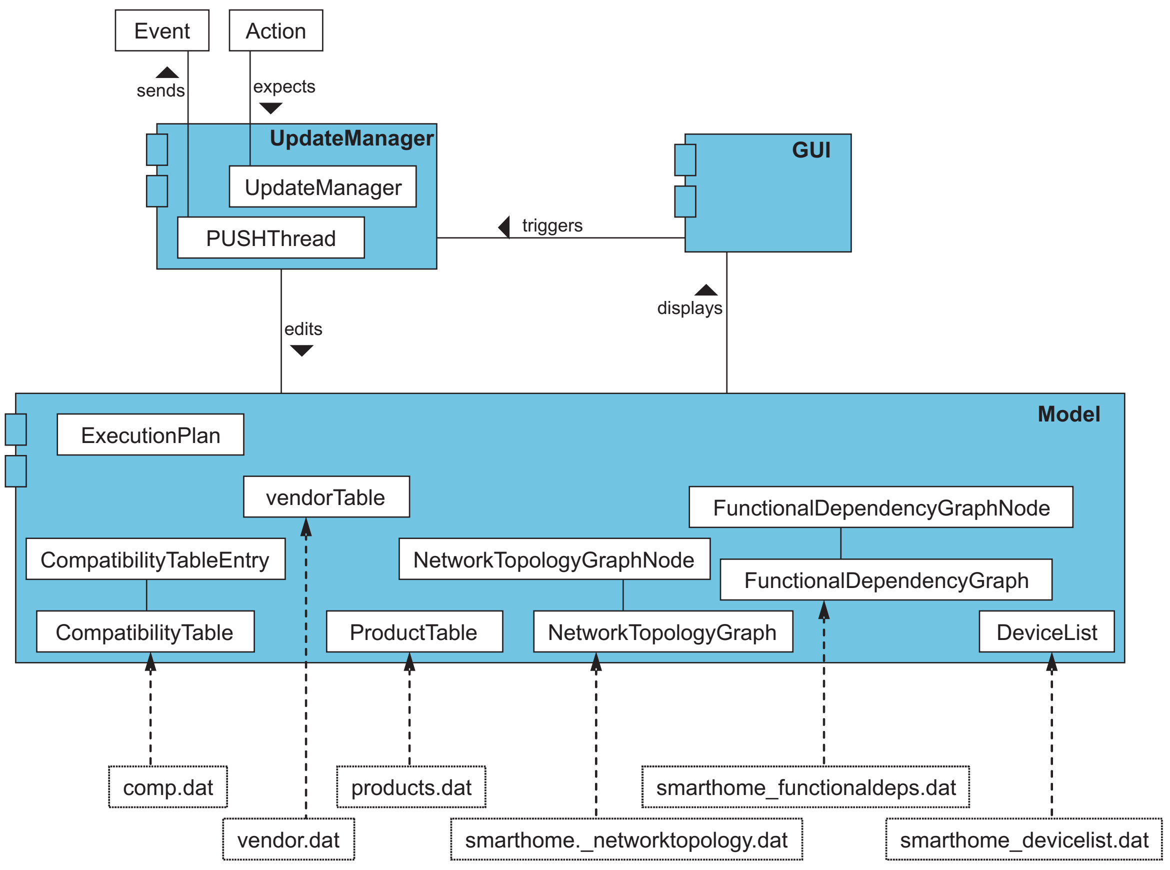
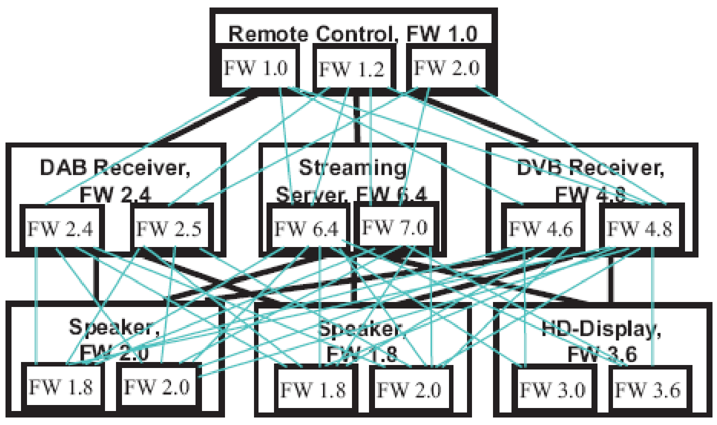


Figure 7.6: Application example 1: An update management simulator for a smart home environment. Overall system architecture.

Lessons learnt in this project include the following findings: This project was driven with IBM's Autonomic Computing and MAPEK loop in mind. Along this scheme, one idea was to create an evaluation scheme for systems with self-management capabilities, which was also turned into practice. However, the evaluations showed to be much effort. And while there are actual patterns in them, all the bookkeeping which pattern was applied where and why was not considered worth it. For this reason, the general evaluation was not done in terms of an actual detailed catalog but to enhance the knowledge of the author.

This project also showed the necessity of automation levels, and the need for a DECIDE step to use them (see Fig. 7.8). Searching for related work, we found the OODA loop and systems engineering are applicable. This led to the inclusion of knowledge from the systems engineering domain, task analysis and the MAPDEK loop. We also found that for operators of such an update service for smart home devices, an alignment of tasks to ITIL in the organizational world seemed beneficial. There we dicovered the overlap of ITIL, FCAPS, and the Self-CHOP properties.



updateTable

Remote Control, FW1.2 / DAB, FW 2.5 / Streaming Server 7.0 /DVB, FW 4.8 / Speaker, FW 1.8 / Speaker, FW 1.8 / Display, FW 3.6
Remote Control, FW1.2 / DAB, FW 2.5 / Streaming Server 7.0 /DVB, FW 4.8 / Speaker, FW 2.0 / Speaker, FW 1.8 / Display, FW 3.6
Remote Control, FW1.2 / DAB, FW 2.5 / Streaming Server 7.0 /DVB, FW 4.8 / Speaker, FW 1.8 / Speaker, FW 2.0 / Display, FW 3.6
Remote Control, FW1.2 / DAB, FW 2.5 / Streaming Server 7.0 /DVB, FW 4.8 / Speaker, FW 2.0 / Speaker, FW 2.0 / Display, FW 3.6

Figure 7.7: Application example 1: Dependency graph of functional dependencies (thick black lines) and firmware compatibility (thin blue lines). A tree search algorithm uses this information to find configurations with compatible firmwares that satisfy as many functional dependencies as possible.

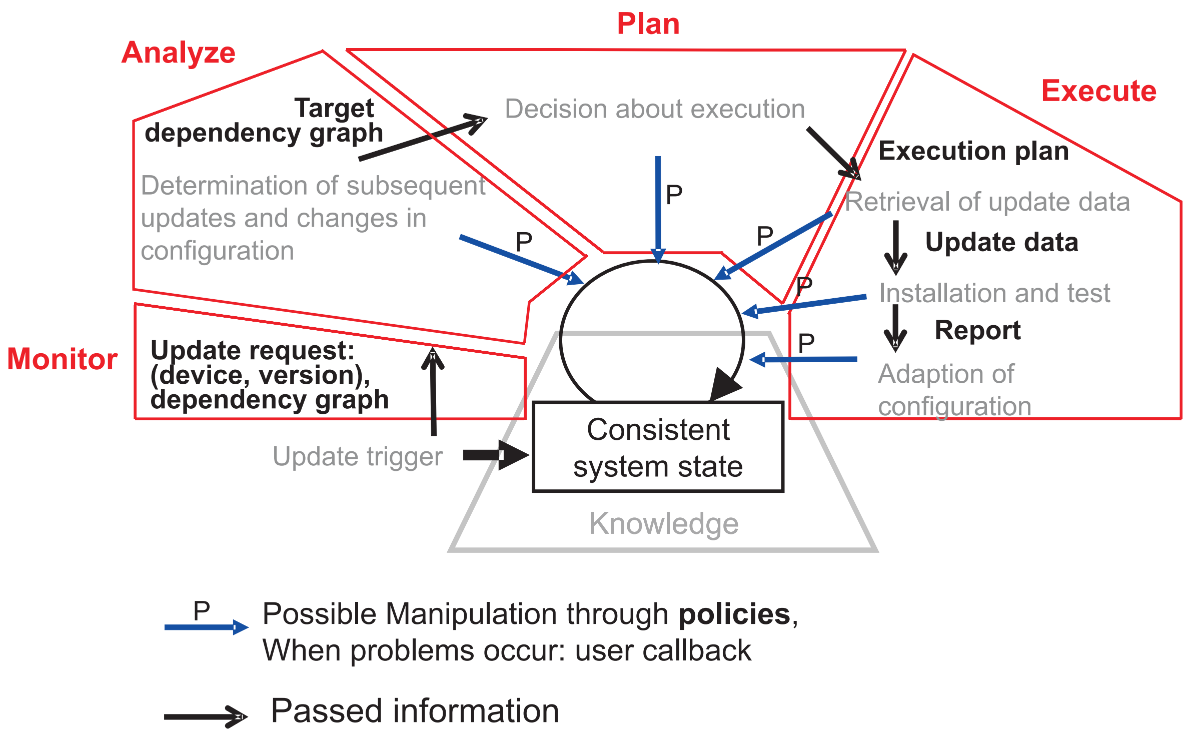


Figure 7.8: Application example 1: MAPEK loop for this example. Notice, how there is a decision in the planning phase. This led from MAPEK to MAPDEK loops.

Virtual machine update management for a cluster, Fujitsu-Siemens Computers (Jan 2008 to Jan 2009) In the scope of an industry project with Fujitsu-Siemens Computers, Young-Chul Jung (a TUM student supervised by the author of this thesis) took care of virtual machine update management for a cluster in his bachelor thesis [Jun08].

Thanks to the EIMA scheme, the tasks "add/remove server", "add/remove VM", "distribute VMs", "rolling update of servers" were quickly identified and initially turned into workflows in BPMN using Bizagi Process Modeller. Based on a keyword search with the keyword "bin-packing" from the catalog of machine capabilities, an optimization algorithm for balancing VMs on servers with few VM transitions was quickly found in an academic paper.

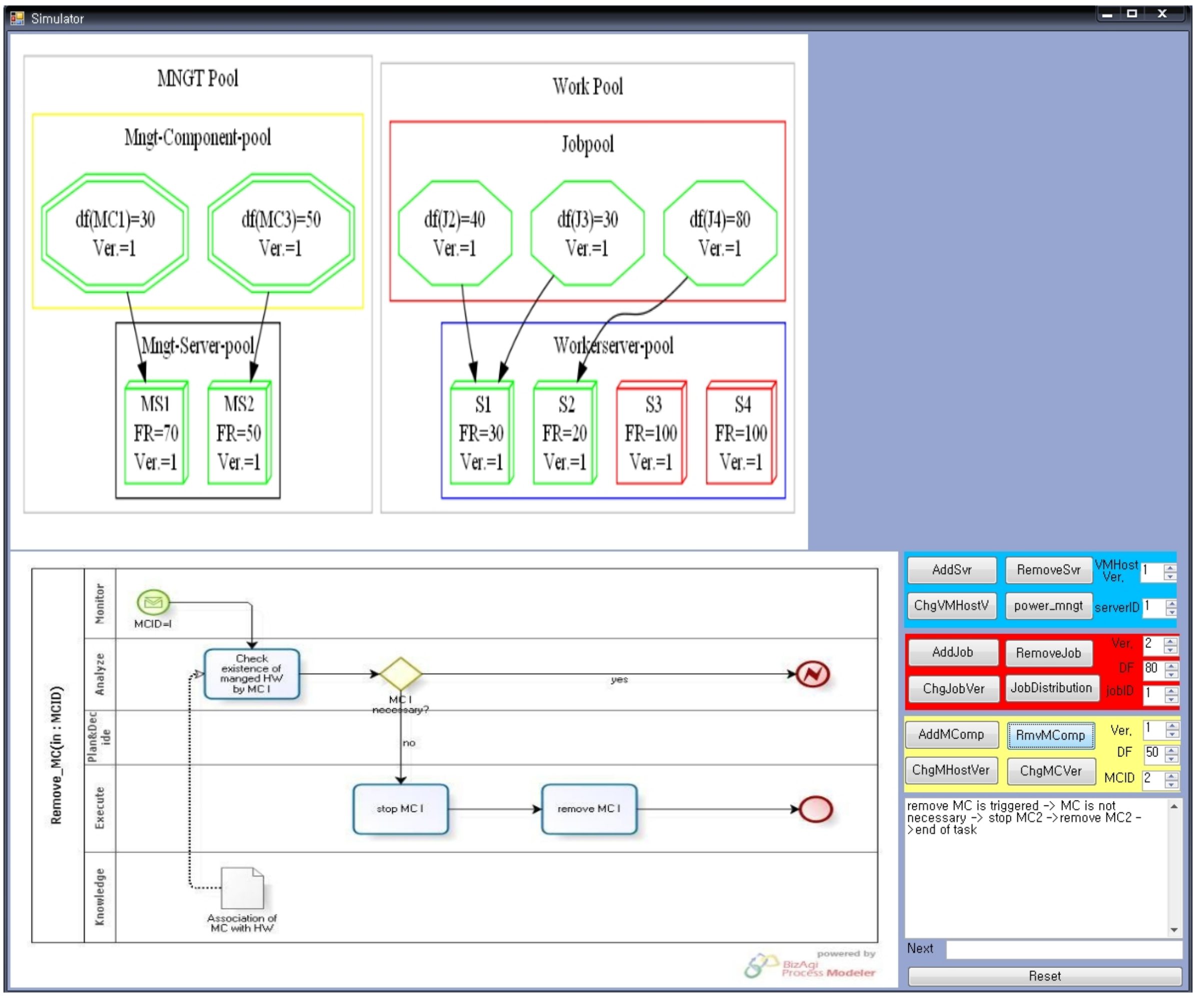
The student implemented the algorithm for moving VMs with a miminal number of shifts, and created a simple simulator with animated graphics in Visual Basic (this choice was based on previous programming experience by the student). As a special feature, this simulator also featured the parallel display of both: workflow diagram and the changes in a system model, while the workflow is being performed (see Fig. 7.9). The simulator allows the user to interactively create VMs with certain loads and to run updates on them. Then, the user can step through the changes done, when update commands are received. The distribution of the VMs running on a server with the hypervisor being updated were distributed along the bin-packing scheme, then the update was performed, and the VMs were moved back. The bin-packing placement could also be triggered independently.

Like the first one, also this second project resulted in a few lessons learnt: This project reused the overall MAPDEK scheme from the first project and therefore quickly got started. Monitoring/Execution interfaces to actual hypervisors/virtualization management software and dynamic function allocation however were left out in the interest of conceptuality and brevity.

This application example illustrated, that indeed a simple EIMA-based demonstrator running a simulation is valuable, provided that it shows both the task decomposition and the effects on the infrastructure at the same time. Before, even despite an intensive search for tools, we noticed that existing modelling or simulation tools do not allow parallel infrastructure simulation and a representation of tasks. Here, we learnt, that an EIMA model is nice (like a CAD drawing), but the simulation (or Computer Aided Engineering as in the domain of mechanical engineering) adds even more value to these models. With that respect, EIMA contains basic information about an IT management automation problem, but it remains static. A simulation based on this infomation further increases the trust in the analysis algorithm to see how the algorithm performs instead of just having pseudo code.

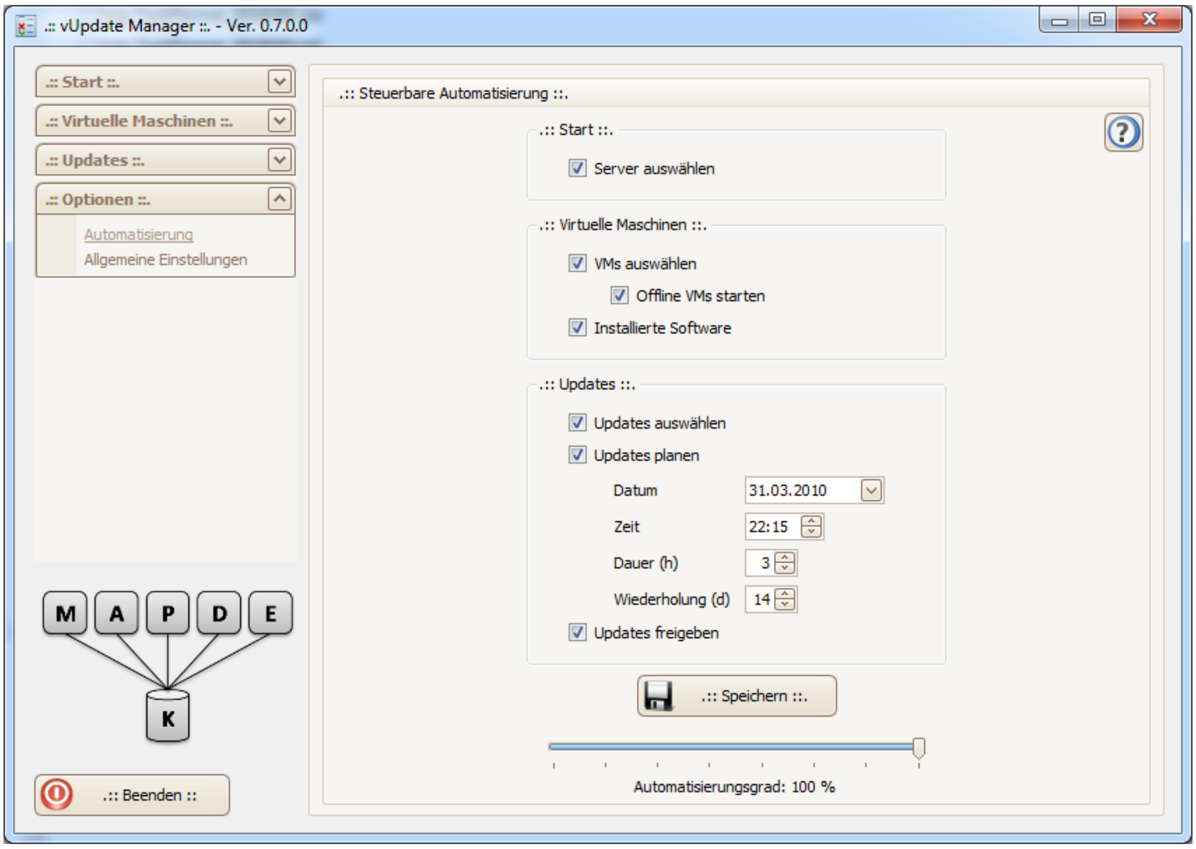
VM update management for a single ESX host with Linux/Windows guest OS's, LRZ (Sep 2009 to Mar 2010) This diploma thesis [Pre10] by Florian Preugschat, student at LMU, supervised by the author of this thesis as well, dealt with a more concrete automation problem arising in practice: consistent update management for the many Linux and Windows VMs on an ESX server. Again, the MAPDEK scheme for general update management could be reused, but it now needed monitoring and execution components interfacing to the VM host, and the individual VM running Linux or Windows. This was solved by getting access to the Vmware API for the ESX server, the Windows update agents (WUA) on Windows VMs, and `apt` on Linux VMs.

For consistency among VM groups, VM group features were added on top of individual VM update loops that reused the original OS's WUA/apt update mechanism. In addition, automation levels were implemented that allow full automation or manual decisions and override in the individual MAPDEK steps and planned tasks (see Fig. 7.10). An overlaid "one – some – many" update plan transports system operator experience into this scenario. Concepts for extended functionality that have been described but implemented include: rolling updates and snapshots for quick recovery from update faults.



remove MC is triggered -> MC is not necessary -> stop MC2 ->remove MC2 ->end of task

Figure 7.9: Application example 2: An update management simulator for a server cluster. The top half shows a model of the infrastructure with management and worker nodes. The lower right hand side shows actions that can be triggered, the lower left hand side shows the associated workflow that is organized in a MAPDEK fashion.



.:: Steuerbare Automatisierung ::.

.:: Start ::.

- [✓] Server auswählen

.:: Virtuelle Maschinen ::.

- [✓] VMs auswählen
  - [✓] Offline VMs starten
- [✓] Installierte Software

.:: Updates ::.

- [✓] Updates auswählen
- [✓] Updates planen

Datum	31.03.2010
Zeit	22:15
Dauer (h)	3
Wiederholung (d)	14

- [✓] Updates freigeben

Automatisierungsgrad: 100 %

Figure 7.10: Application example 3: Automation levels in an update management automation application for the consolidated update of Windows and Linux VMs

Conclusion All three scenarios deal with update management, but in different settings. Each next one was easier than before, with a knowledge reuse along the MAPDEK scheme.

In these settings, even with EIMA being developed in parallel, it proved its utility: we found the EIMA approach to be applicable in terms of generality as well as promising in terms of utility. The scenarios could be blended into each other, necessary replacements of functions could be easily identified via the MAPDEK scheme. The third scenario achieved a remarkable automation ability while keeping most automation flexibility.

A general finding we could make was however: we would need better tool support for EIMA. The existing systems taken into account (Matlab/Simulink, simulation programs, modelling software, CASE tools) are just not appropriate in their functionality that they would cover the vision of EIMA. This is elaborated in the section on potential follow-up work in the next chapter.

8 Conclusion, future work and vision

Contents

This chapter finally wraps up the previous findings. In addition to a check againts the requirements stated in Chapter 2, it lists the improvements of the EIMA approach over the state of the art before. It will then be shown how EIMA can be applied in multiple ways in systems analysis, as well as systems design in all kinds of IT management automation systems. The chapter closes with an overview on potential follow-up work. Indeed, even though giving an answer to the main research question of the thesis, EIMA at the same time opened up a range of new additional research topics.

8.1 Engineered IT management automation as a method

This thesis deals with the problem, that IT management automation projects are all tackled in a different manner with a different general approach and different resulting system architecture.

It is a relevant problem for at least two reasons: 1) more and more IT resources with built-in or associated IT management automation systems are built today. It's inefficient to try to solve each on their own. And 2) doing so, reuse of knowledge between IT management automation systems, as well as reuse of knowledge from other domains is severely limited. While this worked with simple stand-alone remote monitoring and remote control facilities, automation of cognitive tasks will more and more profit from existing knowledge in domains such as AI, statistics, control theory, automated planning. A common structure also would ease integration and coupling of such systems, delegating cognitive partial tasks, and switching between common levels of automation.

So far, this problem is only partly solved. Prior work on approaching systems design includes systems engineering with its steps task analysis and function allocation. But so far systems engineering has not yet been widely applied to IT management automation systems, in contrast to technical application domains such as automotive or aerospace engineering.

The state of the art in IT management automation itself on the technical side includes IBM's Autonomic Computing Initiative, proposing a structure of feedback loops for autonomic systems. On the organizational side, a certain standardization of IT service management processes has been reached with the introduction of process management standardization work like the IT infrastructure library (ITIL), which is a collection of best practices in IT service management, and ISO 20.000, a standard derived from ITIL V2. So far, the mentioned developments all stand each on their own.

The idea of this thesis is now to combine this prior knowledge to create a universal approach to the design of IT management automation systems in four steps (see Fig. 8.1 for an illustration).:

1. Task analysis in the IT management automation scenario, with an implicit goal to associate the identified tasks with a subset of the ITIL reference processes. (see Chapter 3)
2. Reformulation of the tasks into feedback loops of a common structure for cognitive loops. (see Chapter 4)
3. Function allocation of the loop steps to either performed by humans or machine components. (see Chapter 5)
4. Identification of a set of relevant machine capabilities in a catalog structured along the previous three steps. (see Chapter 6)

Each of steps 1–3 is the result of combining existing knowledge in systems engineering and existing knowledge in IT management automation. During this combination the author of this thesis also creates new structures, such as an automation-oriented subset of ITIL V2 called AutoITIL, the MAPDEK loop as basic loop and simpler ways of function allocation to humans/machines. The catalog of machine capabilities, presented during step 4 then adds a non-exhaustive catalog of machine capabilities in IT management automation. It is domain-specific by design and clearly exceeds similar more general attempts such as Fitt's "Men are better at, machines are better at" list, 1951. Other concepts are taken over unaltered after a review during this combination, such as proposals by Sheridan (2000) on the structure and description of tasks and levels of automation. All in all, this way we gain a domain-specific method for engineered IT management automation (EIMA).

8.2 Requirements check

The output of EIMA was required in Chapter 2 to be an appropriate communication artifact 1) between automation engineers and *system operators* and 2) between automation engineers and *system implementers*. Table 8.1 now checks how these six previously mentioned requirements are met by EIMA.

All in all, EIMA meets the requirements stated before. With a uniform way of describing and decomposing IT management automation problems even of high complexity at a high level of abstraction, the information structured along EIMA indeed can serve as a means for communication between multiple stake holders. This top-down approach makes even complex IT management automation systems comprehensible and concise in their description. Especially for system administrators this is a clear benefit during their involvement into systems design, but also at system operations time. Only after EIMA's high-level decomposition, system designers and domain specialists will take care of the details by further refining the tasks and implementing system components.

8.3 Improvements by this approach

To express that this work combines concepts from automation engineering and IT management, the overall method is named "Engineered IT Management Automation" (EIMA). To convince others to apply the EIMA method, it provides a number of advantages over current automation approaches.

The overall benefit of this approach is a reduction of complexity by exploiting similarities of the entities at all 4 layers. These similarities become visible in 1) the common task structure, 2) the common MAPDEK loop structure, 3) the common function allocation scheme and 4) the implementation method pattern catalogue, which itself is structured in alignment to machine capabilities associated with full loops as well as for individual loop steps.

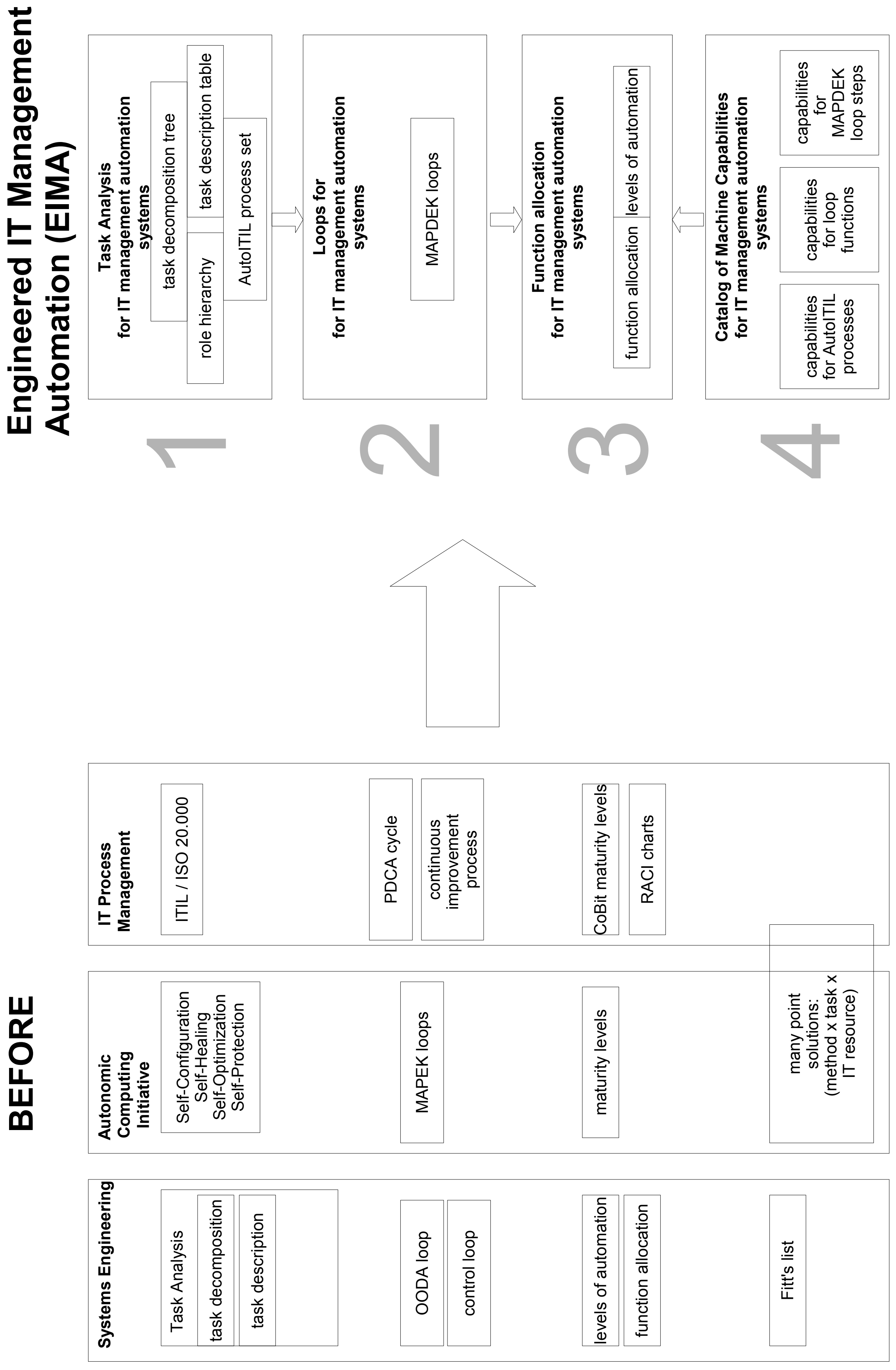

BEFORE
Systems Engineering
Task Analysis
task decomposition
task description
OODA loop
control loop
levels of automation
function allocation
Fitt's list
Autonomic Computing Initiative
Self-Configuration
Self-Healing
Self-Optimization
Self-Protection
MAPEK loops
maturity levels
many point solutions: (method x task x IT resource)
IT Process Management
ITIL / ISO 20.000
PDCA cycle
continuous improvement process
CoBit maturity levels
RACI charts
Engineered IT Management Automation (EIMA)
1
Task Analysis for IT management automation systems
task decomposition tree
task description table
role hierarchy
AutoITIL process set
2
Loops for for IT management automation systems
MAPDEK loops
3
Function allocation for IT management automation systems
function allocation
levels of automation
4
Catalog of Machine Capabilities for IT management automation systems
capabilities for AutoITIL processes
capabilities for loop functions
capabilities for MAPDEK loop steps

Figure 8.1: EIMA combines previous knowledge from Systems Engineering and IT management. It takes over existing concepts, but also creates new ones. This way, we gain a domain-specific engineering method.

Table 8.1: Requirements check for EIMA with the requirements from Chapter 2

Requirement	**Fulfilment in EIMA**
Automation must be dependable for operators. Ideally, it is comprehensible, reliable, verifiable, and verified by system operators. Each of these steps will improve acceptance. At the level of abstraction they can understand, operators need to be involved even at design time to assess early whether the designed system will serve their needs. Later, at operations time, if necessary, they may need an insight into implementation.	EIMA meets this requirement by stating the AutoITIL tasks for a certain IT management solution. This way, system operators have a first impression where this system is applicable. It then supports comprehension by its simple structure of MAPDEK loops. These loops are close to any cognitive task and so also to the tasks of operators. It then gives operators an overview on function allocation, so it is clear which tasks remain with the operators and which tasks are now assigned to machines. Finally, for the machine tasks with the machine capabilities, they get a rough impression, what method/technology takes care of a particular machine tasks. Overall, the EIMA information about the IT management automation system is comprehensive to system administrators as well as concise.
Automation itself must be manageable (monitor and control). Therefore operators need automation levels.	EIMA supports levels of automation in function allocation. In either its static or dynamic variety.
From the design of an automation system, the implementers shall have enough hints to really implement the functionality of the system.	EIMA refers to known implementation methods to enable automation, even though it does not give detailed cookbook-like advice on which machine capabilities to use for what task. In this point it must rely on the experience, background knowledge and creativity of a user of the EIMA method, all of which is supported by EIMA, but not guided along a single path of cause effect relationships to one particular systems design.
When a design is handed over to implementers, the links to the tasks in IT management still need to be visible.	By the method, tasks are linked with loops and later to machine capabilities. This way, we keep the references between the artifacts. If implementers ask themselves for which task they have to implement e.g. a planning system, then they can go back and refer to this task.
In the design, we need to differentiate between a) IT management automation "intelligence" and b) interfacing to management resources.	This split is supported in EIMA by the architecture of the MAPDEK loop. Management "intelligence" is in (M)APD(E)K, the link to the resources is in Monitoring, Execution and the Resource Interface.
This thesis shall encourage automation engineers to discover new implementation methods only rarely used so far.	The implementation method pattern catalog indeed spans a number of research domains, e.g. AI, automation engineering, statistics. It can be extended with much more knowledge.

These four refinement steps allow the reuse of existing and upcoming knowledge in many IT management automation systems by the use of abstraction. Table 8.2 lists the improvements made possible by this approach to balance a more engineering-style way to tackle IT management automation with keeping in mind the links and relationships to the point of view of actual IT management staff.

Table 8.2: Improvements by the approach of this thesis ordered along the layer structure (part 1), continued on next page

Without EIMA	**With EIMA**
TASK ANALYSIS	
Tasks were not structured in a common manner. Workflow diagrams were hardly structured along control hierarchies or categories of tasks.	Tasks get a hierarchical structure inspired by work in systems engineering, decision support systems, and robotics.
LOOPS	
Sub-tasks were not classified.	A cognitive loop structure is used to group tasks into one of 6 categories, Monitor, Analzye, Plan, Decide, Eecute, Knowledge. In addition, a touchpoint describes the sensor/actor interface between the loops.
The MAPEK loop did not explicitly contain a decision step, which is an important one in systems engineering as well as decision support systems.	A step "Decide" was added to the MAPEK loop. This forms MAPDEK loops, combines work from ACI (MAPEK) and systems engineering, and also fits the nature of decision support systems.
FUNCTION ALLOCATION	
System administrators later need to take care of all sub-tasks which an automation management system does not automate. These sub-tasks rarely were listed explicitly. This led to confusion for human administrators, as human necessary and optional involvement was hardly documented.	A function allocation layer makes the split of tasks to either human administrators or a machine explicit.

Table 8.3: Improvements by the approach of this thesis ordered along the layer structure (part 2)

Without EIMA	**With EIMA**
Systems either were quite manual (many IT support tools which rather provide group communication support than actual closed-loop management), or too automated but not very rich in features (e.g. cron jobs). Automation levels remained mostly static.	The introduction of dynamic automation levels allows a flexible delegation of management tasks and a progressive way of learning about the (in-)capabilities of automation step by step. This can improve acceptance significantly.
MACHINE CAPABILITIES	
The knowledge on machine capabilities in IT management automation was only accessible in point solutions. These point solutions rarely are good candidates for taking over general methods to IT management automation tasks not investigated so far intensively.	A (non-exclusive) catalog of machine capabilities and implementation methods suitable for IT management automation captures the knowledge on algorithms and implementation methods. Its structure in loop patterns and step patterns along the MAPDEK loop eases reuse of knowledge.
Monitoring at the sensor interface to IT resources was well investigated, and some approaches to automatic analysis have been applied to IT management with mixed results. The other steps in the MAPDEK loop were usually left to human administrators however.	This thesis presents machine capabilities along the full MAPDEK loop. In addition to presenting the well-known sub-tasks in monitoring it presents capabilities so far not commonly associated with IT management automation, with regard to automatic support in analysis, planning, decision making, execution, and knowledge.
Many other domains with similar problems as IT management automation have developed their own methods and come up with automation routines for a number of sub-tasks. Current publications in IT management automation rarely take this a cross-domain view, and therefore lack inspiration of trying these methods on their particular problems.	To gain new ideas, in addition to many of the point solutions in IT management automation also work in domain-specific engineering for management automation systems, automation engineering, systems engineering, artificial intelligence, and decision support systems has been scanned. Successfully applied methods from these domains are proposed to be adopted in EIMA as well. By adding automated analysis, planning, decision making, and execution, while combining closed-loop control with optional “breakpoints” along multiple levels of automation, two extremes are traded off: using more of the potential of the existing automation methods, but at the same time being aware of the associated risks and responsibilites and therefore keeping human administrators in control of their managed IT environments.

8.4 Applications of the EIMA method

This thesis contributes to the domain of IT management automation the four step EIMA method. The work differs from related work in not focusing on single automation scenario instances (regarded as point solutions), but instead proposing a general method applicable to all such problems and bringing a benefit by its association with existing concepts in systems engineering and IT management. As the method is partly rooted in systems engineering, we can build on the experience that has been gathered there in the design of control systems like autopilots, driver assistance systems. With this approach, the range of applications is broader than that of a typical thesis that tackled a much more focused problem statement.

The EIMA method is intended to be used in the analysis and design of IT management automation systems. It's purpose is to first align the structure of the automation problem to EIMA and then to quickly identify automation potential enabled by existing mathematical methods and machine capabilities. EIMA can be applied in a number of different situations that constantly come up in IT management automation:

- Systems analysis and system evaluation. EIMA can be used to create evaluation schemes or to compare systems along the steps in the method.
- Systems design. EIMA can be used to describe IT management automation systems in terms of the four steps. The chosen level of abstraction makes it well usable to stimulate and structure the communication of systems designers, system implementers, and potential system administrators.
- Sytem requirements. Systems are typically specified along functional and non-functional requirements, which are then mapped to a system model, systems design and system implementation. However, in the scope of management automation systems, requirements are often vague as a result of the unknown capabilities of automated machine support, which can lead to underspecification or overspecification of automation. At this point, the EIMA method can be used to structure the system, present its link into the current process landscape in an organization, and align requirements to the EIMA steps.
- System modeling and simulation. As with system requirements, also system models and system simulators and demonstrators can pick up the EIMA structure, especially rapid prototypes whose overall purpose is to serve as a communication artifact and to jointly refine the requirements. Seeing the tasks broken down and the hierarchy of feedback control loops will facilitate discussions on matters of control early in the design phase. Along EIMA this can be fast, as all functional blocks can be replaced by black boxes at first, with a scope on the other three steps. The generality of EIMA even makes it possible to cover much prototyping on management automation systems with the same generic modeling and simulation tool, instead of having a new simulator or demonstrator for each new problem.
- System documentation. The EIMA method provides a good structure for a special document as part of overall system documentation. Along its structure, designers can explain matters like the relevance to the tasks already done in the environment where a new management automation system is rolled out. It can also explain the hierarchy of control loops and the overruling of different configuration options along the hierarchy of loops. Automation levels can be explained and justified and their effects shown. Finally, administratorstrators can be given just enough information on the actual machine capabilities, algorithms, and implementation methods.

In a proof of concept (see Chapter 7), we show how an existing complex proposal for partial automation in a wide area network scenario is analyzed, and a new proposal is derived that improves automation along known concepts of control systems, and at the same time simplify the organizational structure.

It does so by decomposing the overall problem into a set of smaller problems that can be solved by domain experts in economics, graph theory, network configuration management. A comparison shows that while the previous proposal has created a UML-based draft model for a protocol that eases information exchange, the new proposal actually includes the cognitive tasks that had been left to system administrators in the first proposal. A variable level of automation leaves enough influence by network operators on the automation routine, and therefore provides the basis for better acceptance.

8.5 Potential follow-up work

Potential follow-up work includes the creation of tools for EIMA itself: modeling tools, a knowledge base of machine capabilities, the electronic representation of partial solutions such as loop patterns, and simple visualization and simulation. In the same way as the design of electronic circuits today is supported with circuit simulation programs such as SPICE, or design of 3D objects such as machines or building with CAD applications such as AutoCAD. There, on the basis of these models virtual crash tests are performed on digital car body prototypes, and structural analyses are performed on the statics of buildings.

So, in a likewise manner IT management automation would need virtual, model-based construction of IT management solutions. Once such tools would be available, EIMA models could be exchanged in terms of formal, machine-interpretable files instead of mainly verbal system descriptions interpreted by humans. And these models would open the door for computer-aided engineering like analyses and simulations.

The previous examples in this thesis were created manually with minimal tool support, except a spreadsheet application for tables, drawing applications and a graph layout tool for trees and hierarchies. This proved to be tedious, troublesome, akin to inconsistencies.

Based on this experience, follow-up work on this thesis may derive modeling tools with toolboxes for tasks, loops, automation levels, general simulation tools, as well as CASE tools such as implementation recommendation systems or automation feasibility assessment tools. In general, along the EIMA structure, tools for analysis, validation, simulation, automation planning and documentation can be built. This thesis provides the part that links tasks to implementation methods, while the actual tools will need to be complemented with (existing) modeling components for system architecture and system behavior and are outside the scope of this thesis.

The idea is to now, that the EIMA method itself has been described by this thesis, to create a toolbox around EIMA, probably from existing tools. It would implement the 4 EIMA steps with the features listed in Tab. 8.4.

Table 8.4: Desirable features of a potential tool support for EIMA

Desired feature	**Remarks**
TASK ANALYSIS	
built-in modeling	for task tree, role hierarchy, automation description table
predefined typical tasks	for the AutoITIL set based on an analysis of ITIL and ISO 20.000
built-in process modelling	build processes for the EIMA tasks jointly with system architecture
built-in workflow support	BPMN import/export
LOOPS	
loop modeling	easy rearrangement, easy creation of loop hierachies
predefined loop templates	at least MAPDEK, optionally: control loops
FUNCTION ALLOCATION	
function allocation modeling	automated creation of allocation matrices from tasks and roles, user fills in the matrix with machine support from the catalog of machine capabilities
pre-defined automation levels	easy definition and documentation of automation levels
CATALOG OF MACHINE CAPABILITIES (MC)	
knowledge base	electronically modelled, for collaborative editing, e.g. as a Wiki of models and theory
structure	to link the concepts, we could use ontologies, mind maps, topic maps, concept maps
machine capability adviser	Auto-filter on categories for each task to only highlight relevant methods. Even better: a ranking of methods depending on the task.
MC's in AutoITIL processes	pre-defined models for the techniques, libraries
MC's in loop functions	pre-defined models, e.g. a scheduler in many of its behaviours such as round robin, fifo, etc.
MC's in MAPDEK loops	behavioral models, code examples, model examples, software libraries as a toolbox, thousands of references into appropriate textbooks, reference books, theory and application examples
MC's in resource interfaces	software libraries with access to a set of actual IT resources (network components, servers, storage, applications, management software, ...)
MC's in management interfaces	pre-defined set with interfaces for control loops, policy-based management, utility functions etc.
MC's in communication buses	easy use of existing middleware, different types of middleware abstracted behind a common interface
MODEL INTEGRATION	
linking	in the four previously mentioned steps between modelled artifacts for consistency
graphical modelling and visualization	is important as designers are visually oriented, see other domains. Also as a communication means.
graphical simulations	would enable simple what-if analyses. System models and artifacts could be simulated on the spot with sample input data.
high-level verification	simple completeness checks, component statistics

List of Figures

List of Tables

Bibliography

[AB05] James S. Albus and Anthony J. Barbera. Rcs: A cognitive architecture for intelligent multi-agent systems. *Annual Reviews in Control*, 29(1):87 – 99, 2005.

[Air04a] Airbus. *Flight Operations Briefing Notes, Standard Operating Procedures, Golden Rules*, January 2004.

[Air04b] Airbus. *Flight Operations Briefing Notes, Standard Operating Procedures, Optimum Use of Automation*, January 2004.

[Air06] Airbus. *Flight Operations Briefing Notes, Standard Operating Procedures, Operating Philosophy*, September 2006.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, united states ed edition, 2 1993.

[Bal00] Helmut Balzert. *Lehrbuch der Software-Technik - Software-Entwicklung - mit 2 CD-ROM*. Spektrum-Akademischer Vlg, 2., überarb. und erw. a. edition, 12 2000.

[BC06] Mark Burgess and Alva Couch. A.: Autonomic computing approximated by fixed-point promises. In *In: Proceedings of First IEEE International Workshop on Modeling Autonomic Communication Environments*, 2006.

[BKM$^+$04] Rob Barrett, Eser Kandogan, Paul P. Maglio, Eben M. Haber, Leila Takayama, and Madhu Prabaker. Field studies of computer system administrators: analysis of system management tools and practices. In James D. Herbsleb and Gary M. Olson, editors, *CSCW*, pages 388–395. ACM, 2004.

[BMKB05] Rob Barrett, Paul P. Maglio, Eser Kandogan, and John H. Bailey. Usable autonomic computing systems: The system administrators' perspective. *Advanced Engineering Informatics*, 19(3):213–221, 2005.

[Bou09] L. Boursas. *Trust-Based Access Control in Federated Environments*. Dissertation, Ludwig–Maximilians–Universität München, March 2009.

[Bre07] M. Brenner. *Werkzeugunterstützung für ITIL–orientiertes Dienstmanagement — Ein modellbasierter Ansatz*. Dissertation, Ludwig–Maximilians–Universität München, July 2007.

[BSTB$^+$05] David Bustard, Roy Sterritt, A. Taleb-Bendiab, Andrew Laws, Martin Randles, and Frank Keenan. Towards a systemic approach to autonomic systems engineering. In *Proc. of the 12th Intl. Conf. and Workshops on the Engineering of Computer-Based Systems (ECBS'05)*, pages 465–472, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[CCK$^+$07] Z. Cai, Y. Chen, V. Kumar, D. Milojicic, and K. Schwan. Automated availability management driven by business policies. *Integr. Netw. Manage. IEEE*, 2007.

[CF00] C. Castelfranchi and R. Falcone. Does control reduce or increase trust? a complex relationship. In *Proceedings of Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies*, pages 49–60, June 2000.

[CKS09] M. Cardosa, M. Korupolu, and A. Singh. Shares and utilities based power consolidation in virtualized server environments. In *Proceedings of IFIP/IEEE Integrated Network Management (IM), 2009.*, 2009.

[Cla06] C. Clauss. Entwicklung und Anwendung einer Methodik zur Verfeinerung von ITIL Prozessbeschreibungen am Beispiel des ITIL Change Managements. Master's thesis, Ludwig–Maximilians–Universität München, November 2006.

[CSW$^+$08] David Carrera, Malgorzata Steinder, Ian Whalley, Jordi Torres, and Eduard Ayguade. Managing slas of heterogeneous workloads using dynamic application placement. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 217–218, New York, NY, USA, 2008. ACM.

[Dan07] V. Danciu. *Application of policy-based techniques to process-oriented IT service management.* Dissertation, Ludwig–Maximilians–Universität München, July 2007.

[DFT02] John C. Doyle, Bruce A. Francis, and Allen R. Tannenbaum. *Feedback Control Theory (Dover Books on Engineering).* Dover Publications, 2002.

[Dow04] Jim Dowling. *The Decentralised Coordination of Self-Adaptive Components for Autonomic Distributed Systems.* PhD thesis, Dept of Computer Science, Trinity College Dublin, 2004.

[dV09] Paul deGrandis and Giuseppe Valetto. Elicitation and utilization of application-level utility functions. In *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*, pages 107–116, New York, NY, USA, 2009. ACM.

[(ed08] Hui-Min Huang (ed.). Autonomy levels for unmanned systems (alfus) framework, volume i: Terminology, version 2.0. Technical Report NIST Special Publication 1011-II-1.0, NIST, October 2008.

[FFBS04] Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. *Head First Design Patterns.* O'Reilly Media, 1 edition, 10 2004.

[FHSL96] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[FMS09] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML (Revised Printing): The Systems Modeling Language (The MK/OMG Press).* Morgan Kaufmann, revised edition, 9 2009.

[Fow96] Martin Fowler. *Analysis Patterns: Reusable Object Models.* Addison-Wesley Professional, 10 1996.

[FPAC94] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonself discrimination in a computer. In *In Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE Computer Society Press, 1994.

[FS07] Hans-Peter Fröschle and Susanne Strahringer, editors. *IT-Industrialisierung.* Dpunkt Verlag, 1., aufl. edition, 8 2007.

[GC03] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

[GCC07] Sven Graupner, Nigel Cook, and Derek Coleman. Automation controller for operational it management. In *Integrated Network Management*, pages 363–372. IEEE, 2007.

[gF08] N. gentschen Felde. *Ein föderiertes Intrusion Detection System für Grids*. Dissertation, Ludwig–Maximilians–Universität München, December 2008.

[GHJV99] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, illustrated edition edition, 18 1999.

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory & Practice (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, 1 edition, 5 2004.

[Ham09] M. K. Hamm. *Eine Methode zur Spezifikation der IT-Service-Managementprozesse Verketteter Dienste*. Dissertation, Ludwig–Maximilians–Universität München, June 2009.

[HAN99] Heinz-Gerd Hegering, Sebastian Abeck, and Bernhard Neumair. *Integrated Management of Networked Systems*. Morgan Kaufmann, San Diego, 1999.

[Han07] A. Hanemann. *Automated IT Service Fault Diagnosis Based on Event Correlation Techniques*. Dissertation, Ludwig–Maximilians–Universität München, July 2007.

[HDPT04] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[HMAG07] Hui-Min Huang, Elena Messina, James Albus, and Ad Hoc ALFUS Working Group. Autonomy levels for unmanned systems (alfus) framework, volume ii: Framework models, version 1.0. Technical Report NIST Special Publication 1011-II-1.0, NIST, December 2007.

[Hom07] W. Hommel. *Architektur- und Werkzeugkonzepte für föderiertes Identitäts–Management*. Dissertation, Ludwig–Maximilians–Universität München, July 2007.

[Hor01] Paul Horn. Autonomic computing: Ibm's perspective on the state of information technology, 2001.

[HV08] Mike Hinchey and Emil Vassev. An evaluation study of the effectiveness of modeling NASA swarm-based exploration missions with ASSL. In Chunming Rong, Martin Gilje Jaatun, Frode Eika Sandnes, Laurence Tianruo Yang, and Jianhua Ma, editors, *Autonomic and Trusted Computing, 5th International Conference, ATC 2008, Oslo, Norway, June 23-25, 2008, Proceedings*, volume 5060 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2008.

[HWM07] Klaus Herrmann, Matthias Werner, and Gero Mühl. A methodology for classifying self-organizing software systems. *International Transactions on Systems Science and Applications*, 2007. (accepted for publication).

[IBM06] IBM. An architectural blueprint for autonomic computing (4th ed.), 2006.

[IC01] Software Group IBM Corporation. Autonomic computing concepts. Technical Report G325-6904-00, IBM Corporation, 2001.

[ILX05] Key Skills ILX. *ITIL Service Support and Service Delivery Process Model*. Key Skills ILX, 12 2005.

[Ilx08] Ilxgroup. *ITIL V3 Process Model [11 x 15] laminated*. Key Skills Ltd, 1st edition, 1 2008.

[Ins95] Carnegie Mellon Univ. Software Engineering Inst. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley Professional, 1995.

[Ins07] IT Governance Institute. *Cobit 4.1*. Isaca, 5 2007.

[Ish04] Yoshiteru Ishida. *Immunity-Based Systems*. Springer, 1 edition, 5 2004.

[Jun08] Young-Chul Jung. Simulation and visualization of procedures in distributed it infrastructures. Bachelor thesis, Technische Universität München, October 2008.

[JVN06] M. Jansen-Vullers and M. Netjes. Business process simulation - a tool survey. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Aarhus, Denmark, October 2006.

[KA92] B. Kirwan and L. K. Ainsworth, editors. *A Guide To Task Analysis: The Task Analysis Working Group*. CRC, 1 edition, 9 1992.

[KBMH08] Eser Kandogan, John H. Bailey, Paul P. Maglio, and Eben M. Haber. Policy-based it automation: the role of human judgment. In AEleen Frisch, Eser Kandogan, Wayne G. Lutters, James D. Thornton, and Mustapha Mouloua, editors, *CHIMIT*, page 9. ACM, 2008.

[KCES07] V. Kumar, B.F. Cooper, G. Eisenhauer, and K. Schwan. Enabling policy-driven self-management for enterprise-scale systems. *HotAC II: Hot Topics in Autonomic Computing on Hot Topics in Autonomic Computing*, pages 4–14, 2007.

[Kem04] B. Kempter. *Konfliktbehandlung im policy–basierten Management mittels a priori Modellierung*. Dissertation, Ludwig–Maximilians–Universität München, August 2004.

[KHBM09] Eser Kandogan, Eben M. Haber, John H. Bailey, and Paul P. Maglio. Studying reactive, risky, complex, long-spanning, and collaborative work: The case of it service delivery. In Julie A. Jacko, editor, *HCI (4)*, volume 5613 of *Lecture Notes in Computer Science*, pages 504–513. Springer, 2009.

[KHW$^+$04] Alexander Keller, Joseph L. Hellerstein, Joel L. Wolf, Kun-Lung Wu, and Vijaya Krishnan. The champs system: change management with planning and scheduling. In *NOMS (1)*, pages 395–408. IEEE, 2004.

[Kor01] David Kortenkamp. Adjustably autonomous control systems. `http://www.traclabs.com/~korten/Sweden/adjustable_autonomy.pdf`, 2001.

[Kub93] Carol Kubicki. The system administration maturity model - samm. In *LISA '93: Proceedings of the 7th USENIX conference on System administration*, pages 213–225, Berkeley, CA, USA, 1993. USENIX Association.

[KWDT06] Ralf König, Diana Weiss, Vitalian Danciu, and Georg Treu. Policy-based update management in smart home environments (push). Technical report, Kooperationsbericht Siemens — MNM-Team, Siemens/TUM-Kooperation, December 2006.

[Lim02] Thomas Limoncelli. *The Practice of System and Network Administration*. Addison-Wesley, Boston, 2002.

[LLMY05] Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and Yijun Yu. Towards requirements-driven autonomic systems design. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, 2005.

[Lov07] Jonathan Love. *Process Automation Handbook: A Guide to Theory and Practice*. Springer, London, 1 edition, 10 2007.

[LP05] Zhen Li and Manish Parashar. A decentralized agent framework for dynamic composition and coordination for autonomic applications. In *DEXA Workshops*, pages 165–169. IEEE Computer Society, 2005.

[Lun06] Jan Lunze. *Ereignisdiskrete Systeme*. Oldenbourg Wissensch.Vlg, 9 2006.

[Lun07] Jan Lunze. *Automatisierungstechnik: Methoden für die Überwachung und Steuerung kontinuierlicher und ereignisdiskreter Systeme*. Oldenbourg, 2., überarbeitete auflage. edition, 11 2007.

[Lyk00] Howie Lyke. *IT Automation - The Quest for Lights Out*. Prentice Hall, Englewood Cliffs, 2000.

[Mah83] Günter Heinz Mahr. Automatischer operator. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 3:188, 1983.

[Mea08] Donella H. Meadows. *Thinking in Systems: A Primer*. Chelsea Green Publishing, 12 2008.

[MGH$^+$98] Drew Mcdermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *The Planning Domain Definition Language*, 1998.

[MH03] T.B. Malone and C.C. Heasly. Function Allocation: Policy, Practice, Procedures, & Process. *Naval Engineers Journal*, 115(2):49–62, 2003.

[MN06a] Jan Mendling and Markus Nüttgens. Epc markup language (epml): an xml-based interchange format for event-driven process chains (epc). *Inf. Syst. E-Business Management*, 4(3):245–263, 2006.

[MN06b] Jan Mendling and Markus Nüttgens. Xml interchange formats for business process management. *Inf. Syst. E-Business Management*, 4(3):217–220, 2006.

[MSL08] A. McCloskey, B. Simmons, and H. Lutfiyya. Policy-based dynamic provisioning in data centers based on SLAs, business rules and business objectives. In *Network Operations and Management Symposium*, pages 1–4, 2008.

[Nor04] Ovidiu Noran. Uml vs. idef: An ontology-oriented comparative study in view of business modelling. In *ICEIS (3)*, pages 674–682, 2004.

[Ogc07] Ogc. *Itil Lifecycle Publication Suite, Version 3: Continual Service Improvement, Service Operation, Service Strategy, Service Transition, Service Design*. Stationery Office, version 3 edition, 7 2007.

[PKGV06] Janak J. Parekh, Gail E. Kaiser, Philip Gross, and Giuseppe Valetto. Retrofitting autonomic capabilities onto legacy systems. *Cluster Computing*, 9(2):141–159, 2006.

[Pre10] Florian Preugschat. Automatisierung des software-update-prozesses für eine komplexe virtuelle infrastruktur. Master's thesis, Ludwig–Maximilians–Universität München, March 2010.

[PSW00] R. Parasuraman, T.B. Sheridan, and C.D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 30(3):286–297, 2000.

[Pub08] Van Haren Publishing. *MOF (Microsoft Operations Framework): A Pocket Guide: V 4.0 (2008)*. Van Haren Publishing, 3rd revised edition edition, 8 2008.

[Rei08] H. Reiser. *Ein Framework für föderiertes Sicherheitsmanagement*. Habilitation, Ludwig–Maximilians–Universität München, May 2008.

[SBD$^+$05] Joseph S. Sventek, Nagwa Badr, Naranker Dulay, Steven Heeps, Emil Lupu, and Morris Sloman. Self-managed cells and their federation. In Jaelson Castro and Ernest Teniente, editors, *CAiSE Workshops (2)*, pages 97–107. FEUP Edições, Porto, 2005.

[Sch07a] A. Scherer. Entwicklung einer Methodik zur Informations- und Datenmodellierung in IT Service Management Prozessen am Beispiel der ITIL–Prozesse Service Level Management und Configuration Management. Master's thesis, Technische Universität München, November 2007.

[Sch07b] M. Schiffers. *Management dynamischer Virtueller Organisationen in Grids*. Dissertation, Ludwig–Maximilians–Universität München, July 2007.

[Sch08a] T. Schaaf. *IT–gestütztes Service–Level–Management — Anforderungen und Spezifikation einer Managementarchitektur*. Dissertation, Ludwig–Maximilians–Universität München, December 2008.

[Sch08b] D. Schmitz. *Automated Service–Oriented Impact Analysis and Recovery Alternative Selection*. Dissertation, Ludwig–Maximilians–Universität München, July 2008.

[SDBD06] John Strassner, Panagiotis Demestichas, Dragan Boscovic, and George Dimitrakopoulos. Focale—a novel autonomic computing architecture. Presentation at Wireless World Research Forum WG6, April 2006.

[SFC08] David E. Smith, Jeremy Frank, and William Cushing. The anml language. In *Proceedings of The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, 2008.

[SFLS09] Alberto Schaeffer Filho, Emil Lupu, and Morris Sloman. Realising Management and Composition of Self-Managed Cells in Pervasive Healthcare. In *3rd International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, March 2009.

[She02] Thomas B. Sheridan. *Humans and Automation: System Design and Research Issues (Wiley Series in Systems Engineering and Management)*. Wiley-Interscience, New York, 2002.

[Spa07] George Spalding. *Continual Service Improvement ITIL, Version 3*. Stationery Office, version3 edition, 5 2007.

[Tak06a] Hitoshi Takeda. *LCIA - Low Cost Intelligent Automation*. Redline GmbH, 6 2006.

[Tak06b] Hitoshi Takeda. *The Synchronized Production System: Going Beyond Just-In-Time Through Kaizen*. Kogan Page, 9 2006.

[USA09] Setpoint USA. Top 10 things you should know about custom automation, 20. December 2009.

[Vas09] Emil Vassev. *ASSL - Autonomic System Specification Language: A Framework for Specification and Code Generation of Autonomic Systems*. LAP Lambert Academic Publishing, 11 2009.

[VM09] Emil Vassev and Serguei A. Mokhov. An assl-generated architecture for autonomic systems. In Bipin C. Desai, Carson Kai-Sang Leung, and Olga Ormandjieva, editors, *C3S2E*, ACM International Conference Proceeding Series, pages 121–126. ACM, 2009.

[VP07] Emil Vassev and Joey Paquet. Assl - autonomic system specification language. In *SEW*, pages 300–309. IEEE Computer Society, 2007.

[WC05] John R. Wilson and Nigel Corlett, editors. *Evaluation of Human Work, 3rd Edition*. CRC, 3 edition, 4 2005.

[Wil04] David S. Wile. Patterns of self-management. In *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 110–114, New York, NY, USA, 2004. ACM Press.

[Wil07] David Williams. It operations run book automation: The evolving vendor landscape. Technical Report G00146848, Gartner, Inc., May 2007.

[Woo04] Simon Wood. Flight crew reliance on automation. Technical Report 10, Civil Aviation Authority (CAA), December 2004.

[Yam09] M. Yampolskiy. *Maßnahmen zur Sicherung von E2E-QoS bei Verketteten Diensten*. Dissertation, Ludwig–Maximilians–Universität München, December 2009.

[Zet99] Paul Zetocha. A feasible approach for implementing greater levels of satellite autonomy. In *Proceedings of Ground System Architecture Workshop*, 1999.

List of Abbreviations

AAA – authentication, authorization, accounting
AADI – acquire, analyze, display, implement
ABC – atomic, biological, chemical
ABLE – Agent Building and Learning Environment
ABS – anti-blocking system
AC – Autonomic Computing
ACI – Autonomic Computing Initiative
ACID – atomicity, consistency, isolation, durability
ACL – access control list
ACT – Autonomic Computing Toolkit
AE – Autonomic element
AFS – Andrew File System
AI – artificial intelligence
AIPS – Conference on Artificial Intelligence Planning Systems
ALFUS – Autonomy Level for Unmanned Systems
AMT – Intel Active Management Technology
ANML – Action Notation Modeling Language
AP – Analysis and Planning
APD – Analysis, Planning, and Decision
API – Application Programming Interface
ARIS – Architecture of Integrated Information Systems
ARM – Application Response Measurement
ARS – Action Request System
AS – Autonomous System
ASCET – Advanced Simulation and Control Engineering Tool
ATOP – Automatic Operator

BGP – Border Gateway Protocol
BIOS – Basic Input Output System
BLOB – Binary Large Object
BP – Business Process
BPEL – Business Process Execution Language
BPM – Business Process Management
BPMN – Business Process Modeling Notation
BRMS – business rule management system

CAA – Civil Aviation Authority
CAD – Computer-aided design
CAE – Computer-aided engineering
CAM – Computer-aided manufacturing
CAPEX – Capital Expenditures
CAPP – computer-aided process planning
CAQ – computer-aided quality assurance

CAR - computer assisted retrieval
CASE - computer-aided software engineering
CB - Communication Bus
CBE - Common Base Event
CD - compact disc
CDP - Cisco Discovery Protocol
CEO - Chief Executive Officer
CFIA - Component Failure Impact Analysis
CFO - Chief Financial Officer
CHAMPS - Change Management with Planning and Scheduling
CHOP - (Self-)Configuration, Healing, Optimization, Protection
CI - Configuration Item
CIFS - Common Internet File System
CIM - Common Information Model
CIO - Chief Information Officer
CLI - command-line interface
CMDB - configuration management database
CMIP - common management information protocol
CMM - capability maturity model
CMMI-ACQ - CMMI for Acquisitions
CNS - compute, network and storage
COBIT - Control Objectives for Information and related Technology
CORBA - Common Object Request Broker Architecture
COTS - Commercial off-the-shelf
CPU - central processing unit
CRC - cyclic redundancy check
CRM - customer relationship management
CSV - comma separated values
CTA - Cognitive task analysis

DAC - Digital Analog Converter
DoDAF - Department of Defense Architecture Framework
DARPA - Defense Advanced Research Projects Agency
DASADA - Dynamic Assembly for Systems Adaptability,
Dependability, and Assurance
DB - database
DBLP - Digital Bibliography \& Library Project
DCML - Data-center markup language
DEN-ng - Directory Enabled Networks - next generation
DFTM -
DHCP - Dynamic Host Configuration Protocol
DITA - Darwin Information Typing Architecture
DMAIC - Define - Measure - Analyze - Improve - Control
DMSS - Decision-making Support Systems
DNS - Domain Name System
DOD - Department of Defense
DSL - Digital Subscriber Line
DSOM - (Workshop on) Distributed Systems: Operations and Management
DVB - Digital Video Broadcasting
DVD - Digital Versatile Disc

DYMOLA – Dynamic Modeling Laboratory

ECA – event, condition, action
ECAM – Electronic Centralized Aircraft Monitoring
ECC – Error-correcting code
EICAS – Engine Indication and Crew Alerting System
EIMA – Engineered IT Management Automation
ELROB – European Land-Robot Trial
EPC – Event-driven process chains
EPML – EPC markup language
ERP – Enterprize resource planning
eSCM-SP – eSourcing Capability Model for Service Providers
ESP – Electronic Stability Program
eTOM – Enhanced Telecom Operations Map
EU – European Union

FAA – Federal Aviation Administration
FAQ – frequently asked questions
FCAPS – Fault Configuration Accounting Performance Security (Management)
FCS – Flight Control System
FEC – Forward Error Correction
FIM – Federated Identity Management
FIPS – Federal Information Processing Standard
FMC – Fundamental Modeling Concepts
FMEA – Failure modes and effect analysis
FMS – Flight Management System
FSC – Fujitsu-Siemens Computers
FSM – Finite State Machine
FTA – Fault tree analysis

GOMS – goals, operators, methods and selection rules
GUI – Graphical User Interface
GWSDL – Grid-extended WSDL

HA – High Availability
HFE – Human Factors Engineering
HP – Hewlett-Packard
HSRP – Hot Standby Router Protocol
HTA – Hierarchical Task Analysis
HTML – Hypertext Markup Language
HTN – Hierarchical Task Network
HTTP – Hypertext Transfer Protocol
HUD – Head-up Display
HVAC – Heating, Ventilating, and Air Conditioning

IBM – International Business Machines
IC – Integrated Circuit
ICAPS – International Conference on Automated Planning and Scheduling
ICAS – International Conference on Autonomic and Autonomous Systems
ICT – information and communication technology

IDE	Integrated Development Environment
IDEF	Integrated Definition
IDS	Intrusion Detection System
IDSS	Intelligent Decision Support System
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IET	Institution of Engineering and Technology
IETF	Internet Engineering Task Force
ILS	Instrument Landing System
IMAP	Internet Message Access Protocol
IOS	Internetwork Operating System
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information Export
IPS	Intrusion Prevention System
IRS	Intrusion Response System
ISO	International Organization for Standardization
IT	Information Technology
ITIL	Information Technology Infrastructure Library
ITSM	IT service management
ITUP	IBM Tivoli Unified Process
JAA	Joint Aviation Authority
JADE	Java Agent DEvelopment Framework
JDBC	Java Database Connectivity
JMS	Java Message Service
JSR	Java Specification Requests
KDD	Knowledge Discovery in Databases
KIVS	Kommunikation in verteilten Systemen (Conference)
KPI	Key Performance Indicator
KVM	Keyboard, Video, Mouse
LAN	Local Area Network
LCD	Liquid Crystal Display
LDAP	Lightweight Directory Access Protocol
LIDAR	Light Detection And Ranging
LISA	Large Installation System Administration (Conference)
LLDP	Link-layer Discovery Protocol
LRZ	Leibniz Supercomputing Centre
LUN	Logical Unit Number
MABA MABA	Men are better at, machines are better at
MAP	Monitor, Analyze, Plan
MAPDE	Monitor, Analyze, Plan, Decide, Execute
MAPDEK	Monitor, Analyze, Plan, Decide, Execute based on Knowledge
MAPE	Monitor, Analyze, Plan, Execute
MAPEK	Monitor, Analyze, Plan, Execute based on Knowledge
MARTE	Modeling and Analysis of Real Time and Embedded systems
MAS	Management automation system
MATLAB	Matrix Laboratory

MC – Management Component
ME – Managed Element
MGMT – Management
MI – Management Interface
MIB – Management Information Base
MO – Managed Object
MOF – Microsoft Operations Framework
MPLS – Multiprotocol Label Switching
MTBF – Mean Time Between Failures
MTTR – Mean Time To Repair
MX – Mail Exchange

NASA – National Aeronautics and Space Administration
NAT – Network Address Translation
NGOSS – Next-generation Operations Support System
NIC – Network Interface Card
NIST – National Institute of Standardization and Technology
NMS – Network Management System
NOC – Network Operations Center
NREN – National Research and Education Network

OASIS – Organization for the Advancement of
Structured Information Standards
OC – Organic Computing
OCR – Optical Character Recognition
ODBC – Open Database Connectivity
OGC – Office of Government Commerce
OLAP – Online Analytical Processing
OMG – Object Management Group
OODA – Observe, Orient, Decide, Act
OOP – Object-oriented Programming
OPEX – Operational Expenditures
OS – Operating System
OSGi – Open Services Gateway initiative
OSI – Open Systems Interconnect
OSPF – Open Shortest Path First
OTRS – Open Ticket Request System
OV – Openview

PC – Personal Computer
PCB – Printed Circuit Board
PCI – Peripheral Component Interconnect
PDCA – Plan, Decide, Check, Act
PDDL – Planning Domain Definition Language
PDF – Portable Document Format
PID – proportional-integral-derivative (controller)
PKI – Public Key Infrastructure
PLOP – Pattern Languages of Programs (Conference)
POMR – problem oriented medical records
PPC – production planning and control

PRM-IT - Process Reference Model for IT

QoS - Quality of Service

RACI - Responsible, Accountable, Consulted, Informed
RACSI - Responsible, Accountable, Consulted, Support, Informed
RADAR - RAdio Detection And Ranging
RAID - Redundant Array of Inexpensive Disks
RAM - Random-access memory
RBL - Realtime Blackhole List
RCS - Real-time Control System
RDF - Resource Description Framework
RDP - Remote Desktop Protocol
RFC - Request For Comments
RFID - Radio Frequency ID
RI - Resource Interface
RIP - Routing Information Protocol
RMC - (IBM) Rational Method Composer
RMON - Remote Monitoring
ROC - Recovery-Oriented Computing
ROI - Return on Invest
ROV - Remotely Operated Vehicle
RPC - Remote Procedure Call
RPM - RPM Package Manager
RRD - Round-robin Database
RSVP - Resource Reservation Protocol
RT - Real-Time
RTES - Real Time and Embedded Systems
RTM - ready to market
RUP - Rational Unified Process

SAMM - Systems Administration Maturity Model
SAN - Storage Area Network
SASO - stability, accuracy, settling time, overshoot
SATA - Serial ATA
SDD - Solution Deployment Descriptor
SDSDI - Simple Distributed Security Infrastructure
SE - Software Engineering / Systems Engineering
SGML - Standard Generalized Markup Language
SID - NGOSS Shared Information/Data Model
SIP - Session Initiation Protocol
SLA - Service-Level Agreement
SLM - Service-Level Management
SMC - Self-Managed Cell
SME - Small or Medium Enterprise
SMF - Systems Management Function
SMFA - Systems Management Functional Area
SMI-S - SNIA Storage Management Initiative Specification
SML - Service Modeling Language
SMO - Systems Management Overview

SMS – Short Message Service
SMTP – Simple Mail Transfer Protocol
SNIA – Storage Networking Industry Association
SNMP – Simple Network Management Protocol
SOAP – Simple Object Access Protocol
SOX – Sarbanes-Oxley Act
SP – Service Provider
SPEC – Standard Performance Evaluation Corporation
SPICE – Simulation Program with Integrated Circuit Emphasis
SPKI – Simple public key infrastructure
SSL – Secure Sockets Layer
STP – Spanning Tree Protocol
STRIPS – Stanford Research Institute Problem Solver
SVG – Scalable Vector Graphics

TA – Task Analysis
TAM – Technical Architecture Modeling
TAR – Tape Archive
TCO – Total Cost of Ownership
TCP – Transmission Control Protocol
TCSEC – Trusted Computer System Evaluation Criteria
TFT – Thin-Film Transistor

TPC – Transaction Processing Performance Council
TS – Task Structure
TSM – Tivoli Storage Manager
TUC – Chemnitz University of Technology
TUM – Technical University of Munich

UC – Use Case
UDDI – Universal Description, Discovery and Integration
UDP – User Datagram Protocol
UI – User Interface
UML – Unified Modeling Language
UPDM – UML Profile for DoDAF/MODAF
UPS – Uninterruptible Power Supply
USB – Universal Serial Bus

VHDL – VHSIC hardware description language
VHSIC – very-high-speed integrated circuit
VLAN – Virtual LAN
VM – Virtual Machine
VMM – Virtual Machine Monitor
VNC – Virtual Network Computing
VO – Virtual Organization
VP – Variation Point
VRRP – Virtual Router Redundancy Protocol
VW – Volkswagen
WAN – Wide Area Network

WBEM	Web-based Enterprise Management
WDM	Wave-length Division Multiplexer
WG	Working Group
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless LAN
WS	Web Service
WSDL	Web Services Description Language
WSDM	Web Services Distributed Management
WWW	World Wide Web
XACML	eXtensible Access Control Markup Language
XHTML	Extensible Hypertext Markup Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XPDL	XML Process Definition Language
XSL	Extensible Stylesheet Language