A Process Model for the Integrated Reasoning about Quantitative IT Infrastructure Attributes

Dissertation

an der

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

vorgelegt von

Christian Straube

Tag der Einreichung: 06.11.2014

A Process Model for the Integrated Reasoning about Quantitative IT Infrastructure Attributes

Dissertation

an der

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

vorgelegt von

Christian Straube

Tag der Einreichung: 06.11.2014
Tag der Disputation: 15.12.2014

1. Berichterstatter:	**Prof. Dr. Dieter Kranzlmüller**
	Ludwig-Maximilians-Universität München
2. Berichterstatter:	**Prof. Dr. Andrew Grimshaw**
	University of Virginia

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Name, Vorname

Ort, Datum *Unterschrift Doktorand/in*

Formular 3.2

Abstract

IT infrastructures can be quantitatively described by attributes, like performance or energy efficiency. Ever-changing user demands and economic attempts require varying short-term and long-term decisions regarding the alignment of an IT infrastructure and particularly its attributes to this dynamic surrounding. Potentially conflicting attribute goals and the central role of IT infrastructures presuppose decision making based upon reasoning, the process of forming inferences from facts or premises. The focus on specific IT infrastructure parts or a fixed (small) attribute set disqualify existing reasoning approaches for this intent, as they neither cover the (complex) interplay of all IT infrastructure components simultaneously, nor do they address inter- and intra-attribute correlations sufficiently.

This thesis presents a process model for the integrated reasoning about quantitative IT infrastructure attributes. The process model's main idea is to formalize the compilation of an *individual reasoning function*, a mathematical mapping of parametric influencing factors and modifications on an attribute vector. Compilation bases upon model integration to benefit from the multitude of existing specialized, elaborated, and well-established attribute models. The achieved reasoning function consumes an individual tuple of IT infrastructure components, attributes, and external influencing factors to expose a broad applicability. The process model formalizes a reasoning intent in three phases. First, reasoning goals and parameters are collected in a reasoning suite, and formalized in a reasoning function skeleton. Second, the skeleton is iteratively refined, guided by the reasoning suite. Third, the achieved reasoning function is employed for What-if analyses, optimization, or descriptive statistics to conduct the concrete reasoning. The process model provides five template classes that collectively formalize all phases in order to foster reproducibility and to reduce error-proneness.

Process model validation is threefold. A controlled experiment reasons about a Raspberry Pi cluster's performance and energy efficiency to illustrate feasibility. Besides, a requirements analysis on a world-class supercomputer and on the European-wide execution of hydro meteorology simulations as well as a related work examination disclose the process model's level of innovation. Potential future work employs prepared automation capabilities, integrates human factors, and uses reasoning results for the automatic generation of modification recommendations.

Zusammenfassung

IT-Infrastrukturen können mit Attributen, wie Leistung und Energieeffizienz, quantitativ beschrieben werden. Nutzungsbedarfsänderungen und ökonomische Bestrebungen erfordern Kurz- und Langfristentscheidungen zur Anpassung einer IT-Infrastruktur und insbesondere ihre Attribute an dieses dynamische Umfeld. Potentielle Attribut-Zielkonflikte sowie die zentrale Rolle von IT-Infrastrukturen erfordern eine Entscheidungsfindung mittels Reasoning, einem Prozess, der Rückschlüsse (rein) aus Fakten und Prämissen zieht. Die Fokussierung auf spezifische Teile einer IT-Infrastruktur sowie die Beschränkung auf (sehr) wenige Attribute disqualifizieren bestehende Reasoning-Ansätze für dieses Vorhaben, da sie weder das komplexe Zusammenspiel von IT-Infrastruktur-Komponenten, noch Abhängigkeiten zwischen und innerhalb einzelner Attribute ausreichend berücksichtigen können.

Diese Arbeit präsentiert ein Prozessmodell für das integrierte Reasoning über quantitative IT-Infrastruktur-Attribute. Die grundlegende Idee des Prozessmodells ist die Herleitung einer *individuellen Reasoning-Funktion*, einer mathematischen Abbildung von Einfluss- und Modifikationsparametern auf einen Attributvektor. Die Herleitung basiert auf der Integration bestehender (Attribut-)Modelle, um von deren Spezialisierung, Reife und Verbreitung profitieren zu können. Die erzielte Reasoning-Funktion verarbeitet ein individuelles Tupel aus IT-Infrastruktur-Komponenten, Attributen und externen Einflussfaktoren, um eine breite Anwendbarkeit zu gewährleisten. Das Prozessmodell formalisiert ein Reasoning-Vorhaben in drei Phasen. Zunächst werden die Reasoning-Ziele und -Parameter in einer Reasoning-Suite gesammelt und in einem Reasoning-Funktions-Gerüst formalisiert. Anschließend wird das Gerüst entsprechend den Vorgaben der Reasoning-Suite iterativ verfeinert. Abschließend wird die hergeleitete Reasoning-Funktion verwendet, um mittels "What-if"–Analysen, Optimierungsverfahren oder deskriptiver Statistik das Reasoning durchzuführen. Das Prozessmodell enthält fünf Template-Klassen, die den Prozess formalisieren, um Reproduzierbarkeit zu gewährleisten und Fehleranfälligkeit zu reduzieren.

Das Prozessmodell wird auf drei Arten validiert. Ein kontrolliertes Experiment zeigt die Durchführbarkeit des Prozessmodells anhand des Reasonings zur Leistung und Energieeffizienz eines Raspberry Pi Clusters. Eine Anforderungsanalyse an einem Superrechner und an der europaweiten Ausführung von Hydro-Meteorologie-Modellen erläutert gemeinsam mit der Betrachtung verwandter Arbeiten den Innovationsgrad des Prozessmodells. Potentielle Erweiterungen nutzen die vorbereiteten Automatisierungsansätze, integrieren menschliche Faktoren, und generieren Modifikationsempfehlungen basierend auf Reasoning-Ergebnissen.

Contents

Conventions

The following overview describes symbols, abbreviations, and formatting used in this thesis.

Abbreviation Expansion (AE)	Indicates the introduction of an abbreviation or acronym. The expansion is printed in italic letters, followed by the abbreviation or acronym in brackets. They are introduced in every chapter in case the thesis is not read in sequential order. An overview of abbreviations is provided at the end of the thesis on page 367.
↗KB p. 253	References the page of a *Knowledge Base* (KB) entry in Appendix A that further details the used term, diagram, or method.
→Section 1	References a section that builds on the discussed information, fact, or situation. Is mainly used for discussing the research environment and terminology in Chapter 2.
❶	Flags particular details in the scenario descriptions in Chapter 3 to ease referencing those details while extracting and abstracting functional and non-functional requirements.
① ①	Supports referencing figure elements and details in the text throughout the thesis.
NFR-1, T-A1:A6	Identifies research results, in particular (non) functional requirements in Chapter 3, and process model template elements in Chapter 5 and 6.
Shaded box	Acts as container for a template element (cf. Section 5.2), whose numbering and label is provided at its bottom.

continued on next page

continued from previous page

→ *Fig. 2.6, p. 31*	Arranges the section or paragraph in the referenced overview figure, which most chapters and sections provide at the beginning to structure their contents, respectively. The arrangement icon is provided in the left or right page margin, the reference is provided below the icon.

EG-4.1 An example for a generically described situation, fact, or information. The example is referenced in the text by its label that is printed at the side. The label consists of the containing chapter and an ascending numbering. In case an example consists of several parts, each is numbered in the example by italic numbers in brackets *(1)*, and referenced in the text by the example's label succeeded by the number, like *EG-4.1:1*.

Chapter 1

Introduction

This thesis presents a process model for the integrated reasoning about quantitative IT infrastructure attributes. The chapter at hand introduces the underlying research in terms of objectives and methodology, and summarizes key details. In particular, Section 1.1 motivates the research and provides the research context, Section 1.2 derives the pursued research question and related fields. Afterwards, Section 1.3 overviews the presented contribution, and Section 1.4 details the applied methodical course of research.

1.1 Motivation

Information Technology (IT) infrastructures play a central role in science and industry. In science, they facilitate simulation as the third pillar of research [332] by providing exhaustive calculation and storage capabilities. In industry, they act as foundation for IT services, IT-based business initiatives [423, 95, 410] and IT functionality, a complete company might depend on [95, 257]. There are several ways to describe an IT infrastructure, like hardware specifications of contained components, exposed capabilities, or *Total Cost of Ownership* (TOC). A description's suitability and applicability, in turn, depends on the specific situation and the description objectives.

A form of description that is important to IT infrastructure providers and users is the *attribute*, a quantitative IT infrastructure description that considers the IT infrastructure and its components as black boxes during workload execution (*EG-1.1:1*). IT infrastructure attributes are used in a variety of ways and situations, e.g., for describing (legally binding) provisioning specifications in *Service Level Agreements* (SLA), for measuring a company department's success, or for constituting procurement of *High Performance Computing* (HPC) system components.

Aligning the IT infrastructure and especially its attributes to (externally) given target values, stated by “continuing changes in business operations and operating environments” [76, p. 5], is and has been among the top concerns for business executives and researchers [197, 263, 90, 348, 223]. This alignment, also called *IT governance* [423], forces persons in charge to make a variety of short- and long-term decisions regarding the IT infrastructure [95, 131, 335] (*EG-1.1:2*). All decisions share the urgent need to build on (complete) facts, and not (purely) on educated guesses or partial information, because

- the enabling nature and the important role of IT infrastructures require a stable, safe, and forward-looking provision [395, 410, 248, 274, 230],
- potentially entailed modifications often induce not only (positive) intended effects, but also (mostly negative) unavoidable side effects [6] onto attributes (*EG-1.1:3*), resulting in manifold (hidden) dependencies,
- involved stakeholders tend to state disputing attribute target values due to differing interests and objectives (*EG-1.1:4*), producing several correlations.

Thus, decisions must base upon comprehensive facts about all involved aspects, correlations, and potential attribute conflicts, and decision makers need to produce a good attribute value trade-off [436, 90] (*EG-1.1:5*).

EG-1.1

(1) Prevalent HPC attributes are performance and energy efficiency [170], exemplary defined as “time to completion of application execution” and “power consumption in Watt per time step”, respectively. *(2)* Operations must align the former to performance related provisioning obligations stated by SLAs, the latter to expected power consumption levels of hundreds of megawatts in the future [211, 190] and steadily increasing electricity prices. *(3)* Simply speaking, operations could adapt clock rates to address changed performance needs, and level down the HPC infrastructure overnight to reduce power consumption. Both induce two effects: increasing clock rates is beneficial for performance, but disadvantageous for energy consumption (detailed in Section 2.4.2). Another example dealing with contrary effects increases redundancy to improve reliability [134], which also increases energy consumption, and degrades performance due to redundancy overhead [137]. *(4)* Besides the technical correlations, target value conflicts might appear, e.g., power consumption cannot pass a certain level, which contradicts the performance increase according to customer demands. *(5)* Based on these information, a modification’s expected impact on attribute values can be evaluated to decide, whether the expected negative side effects might outweigh the targeted benefits, even before the (costly) investigation of its accomplishment.

This can be achieved by *reasoning*, “the process of forming conclusions, judgments, or inferences from facts or premises”. Compiling useful results and providing a sufficient level of detail in information require facts and premises to fulfill two requirements:

- **IT infrastructure coverage** The manifold components of an IT infrastructure comprehensively interact [244, 401] to provide the IT infrastructure’s capabilities, like *software execution*, *compute* or *store* (*EG-1.2:1*). In this context, it becomes quite difficult to separate the specific contribution of a particular component to the exposed capabilities [139, 268]. Besides, effects induced by local modifications on a single component can quickly and easily cascade and affect the IT infrastructure partly or completely in an unpredictable way (*EG-1.2:2*). This requires reasoning to cover an IT infrastructure in its entirety instead of its single components.
- **Attribute coverage** Several attributes describe an IT infrastructure from different perspectives, producing two correlation types, which require reasoning to cover attributes and particularly their correlations:

  Inter-attribute correlation The focus on potentially differing attribute sets for users and providers produces manifold correlations that were empirically evidenced several times (*EG-1.2:3*).

  Intra-attribute correlation Most attributes are influenced and assembled by a combination of several elements (*EG-1.2:4*).

(1) The SuperMUC at the *Leibniz Supercomputing Center* (LRZ) comprises several components, each built of manifold sub elements (cf. Section 3.2.1), e.g., the 18 thin node islands communicate with three storage areas, interconnected by a specialized InfiniBand setup. *(2)* A clock rate adaption (cf. *EG-1.1:3*) might result in more inter-node traffic, as the faster CPUs process more data, what exceeds the local caches. *(3)* For SuperMUC, users might focus on performance, the LRZ might focus on cost-related power consumption. Barroso et al. observed in Google’s data centers that “every gain in performance has been accompanied by a proportional inflation in overall platform power consumption” [34, p. 50]. *(4)* SuperMUC’s hardware-focused performance is determined by computing cores, communication, and I/O performance [91, 191, 281]. Its application-focused performance is assembled by data distribution, data unit memory access, and (software) communication patterns [31].

EG-1.2

1.2 Research question and related fields

The huge amount of reasoning and analysis approaches for nearly all IT infrastructure component types and attributes (cf. Section 7.4) greatly vary

- in general methodology, i.e., being an empirical or analytic model,
- in attributes and scope (*EG-1.3:1*), and
- in IT infrastructure types (*EG-1.3:2*).

EG-1.3

(1) Abandah et al. [13] consider the performance (*attribute*) of message-passing based node communication (*focus*), Contreras et al. [104] estimate the power consumption (*attribute*) of Intel PXA255 processors (*focus*). *(2)* Pfeiffer et al. [319] and Cao et al. [79] both concentrate on performance, but the former predicts application performance on parallel computers, the latter analyzes agent-based service discovery performance in Grids.

All approaches share the limitation to a (small) subset of component types or attributes. For instance, approaches and models related to performance are usually limited to a single system or application, and allow only the system size and job size to vary, due to the difficulty in producing a truly comprehensive model [30]. Consequently, they neither cover an IT infrastructure in its entirety, nor do they support inter- and intra-attribute spanning reasoning, as required in Section 1.1. Therefore, the thesis and presented results aim at answering the following research question:

How to facilitate reasoning about quantitative IT infrastructure attributes to support decision-making while respecting IT infrastructure complexity and scale as well as inter- and intra-attribute correlations?

The research question is closely related to three fields, each covering particular aspects and issues. The subsequent list discusses the fields ordered according to their employment by the reasoning process:

- **IT infrastructure and attribute modeling** The central role of IT infrastructures (cf. above) cause a (nearly) steady use in production mode. In this mode, IT infrastructures are very sensitive to intrusive analyses, e.g., instrumenting IT infrastructure components or executing a modification and measuring its impact. This situation prohibits reasoning on a suitable level on existing, productive systems [78, 244, 337]. Also physically copying the considered IT infrastructure completely or partly under laboratory conditions to conduct reasoning is rarely possible, because this approach is mostly too expensive and time-consuming [293, 92]. In contrast, this requires *modeling* the considered IT infrastructure, attributes, and related elements. The modeling intent faces two questions:

- *Which elements need to be modeled to facilitate the motivated reasoning?* – The diversity of IT infrastructure characteristics and attributes requires a detailed investigation, which elements are of relevance to achieve the research goal and suitable results.
- *How to manage model complexity?* – Even if a model is perfectly valid, i.e., being sufficiently accurate for the purpose at hand [334], it may have limited usefulness, because its size or bulkiness will outweigh its benefits [334, 307]. Avoiding this imbalance and supporting result creation in a reasonable time frame requires modeling to find a good trade-off between simplicity, selected elements, and granularity. Besides, a simple model is fruitful for reduced run time, fast development, and easy result interpretation [96].

• **Model selection and substitution** Having an IT infrastructure model, attribute model(s) must be assigned to IT infrastructure components according to the reasoning objectives. Assignment faces two questions:

- *How to select an appropriate attribute model?* – Although the alluded diversity of existing attribute models facilitates the use of a specialized, elaborated model, it also poses the questions how to select an *appropriate* one, and how to prioritize model aspects, like accuracy, focus, or result interoperability. Achieving reproducible, and objective results calls for a definition of the selection process.
- *How to provide results in case no appropriate model exists?* – Despite model plurality, there might be situations no model is suitable for a particular reasoning intent. The combination of (partial) results, computed by assigned models, calls for alternatives that provide (reasonable) replacement values in case no suitable model exists.

• **Workload and load integration** Load describes the workload induced use of an IT infrastructure component or the entire IT infrastructure in percent at a certain point in time in relation to its maximum capability or capacity (cf. Section 2.3.3). Most potential model integration candidates consume some kind of (work)load as input parameter, due to its (empirically) evidenced (strong) influence on nearly all IT infrastructure attributes [173, 347] (*EG-1.4*). (Work)load consideration faces the following question:

- *How to select an appropriate workload for the current reasoning activity?* – The variety of existing workloads is split in several (sub) groups, like *applications* and *benchmarks* [93], each pursuing a different goal and implying certain consequences. This requires a clear sighted selection to address implicit effects.

EG-1.4

Koziolek [239] discusses workload impact on system response time (*performance*), Talbot et al. [383] describe the load dependent power consumption of communication components (*energy efficiency*), and Jones et al. [215] analyze workload and failure rate interdependencies (*reliability*).

1.3 Research results and contribution

The section summarizes the presented research results that address and answer the research question discussed above. Figure 1.1 schematically depicts the building blocks: Section 1.3.1 overviews the *process model for the integrated reasoning about quantitative IT infrastructure attributes*, and highlights its key ideas and design concepts. Section 1.3.2 introduces the identified requirements and the extracted morphological field that guide and scope research, respectively. Section 1.3.3 itemizes resulting publications, and Section 1.3.4 relates the research results to existing work. All mentioned aspects are extensively detailed in further sections as indicated.

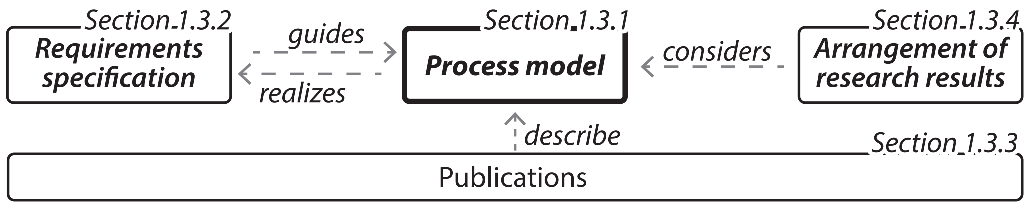


Figure 1.1: Schematic overview of research results.

1.3.1 Process model

→ Fig. 1.1 (p. 6)

Section 1.2 motivates a model-based approach for reasoning about quantitative IT infrastructure attributes. Three reasons disqualify *one-size-fits-all* models that commonly try to cover a problem space entirely (cf. Section 4.1):

Extend of problem space, caused by the great variety of IT infrastructure component types and attributes (*EG-1.5:1*).

Individual parameters and objectives, required by the dynamic IT infrastructure surroundings, and the plurality of reasoning intents (*EG-1.1:4*).

Accuracy vs. portability, implied by the objective of generating a sustainable and widely applicable solution (*EG-1.5:2*).

> *(1)* In the focused field of HPC micro processor hardware, various architectures offer very distinct features (cf. Section 2.2), workloads are optimized for [48] (cf. Section 2.3). Besides, an attribute can be implemented in manifold ways (cf. Section 2.4), e.g., performance as time to completion or *Floating point operations per second* (FLOP/s). *(2)* Analyzing the performance of an HPC cluster based on generic communication patterns is *portable* to a variety of HPC clusters, but less *accurate* than a cluster specific model, as communication patterns do not (fully) cover aspects of the employed hardware, like Infiniband or Myrinet.

EG-1.5

The research's driving idea to address this situation is the development of a process model that generically formalizes *how* to compile an *individual* and casuistic reasoning model, suitable for a specific reasoning intent (cf. Section 4.1). The reasoning model is a mathematical mapping, called a *reasoning function*, of a parameter set on a vector of quantitative IT infrastructure attribute values. Equation 1.1 depicts the reasoning function f in its most generic form. The reasoning function's domain is semantically decomposed in *modification* and *configuration* parameters to enable appropriate handling, respectively (EG-4.2):

- **Modification parameters** Aspects being subject to change of a reasoning intent or planned modification, especially hardware. They are examined in more detail, e.g., in terms of analyzed value ranges.
- **Configuration parameters** IT infrastructure externals and other factors, like application scaling and electricity price deviations, that influence the reasoning outcome, but are *not* subject to change. Instead, they are (mostly) taken for granted, and a small value range constrains reasoning.

$$f(\underbrace{\underbrace{mod_1, ..., mod_n}_{\text{Modification parameters}}, \underbrace{conf_1, ..., conf_m}_{\text{Configuration parameters}}}_{\text{Domain}}) = \underbrace{\underbrace{\begin{pmatrix} attr_1 \\ ... \\ attr_z \end{pmatrix}}_{\text{Attribute values}}}_{\text{Co domain}} \tag{1.1}$$

The decomposition is advantageous for reasoning efficiency and result interpretation, but does not disadvantage reasoning methodologies, like mathematical optimization (↗KB p. 262), since both parameter sets are treated in the same mathematical way.

Two main design concepts underlie the process model in general and the compilation of an individual reasoning function in particular (cf. Section 4.2):

- **Integration of existing artifacts** The alluded central role of IT infrastructure attributes produces a huge amount of specialized (and mature) models, instrumented components, and gained measurements, covering a variety of partial aspects of a reasoning intent. The process model bases upon integration to benefit from this situation and to enable the use of elaborated, established, and especially validated artifacts. In other words, the process model builds upon (important) prior work done and results achieved by other researchers, but also makes a logical progression by combining these results in a new way [212].

- **Iterative function refinement** Although a variety of factors influence an attribute, reasoning might be interested in only a (small) subset. Thus, reasoning function compilation begins with a coarse-grained function skeleton and iteratively refines it until the function (co) domain comply to the reasoning objectives (EG-4.3).

Figure 1.2 overviews the process model's building blocks and highlights its two parts, namely *artifacts and procedures* and the *reasoning methodology*:

- **Artifacts and procedures** Define and formalize notions, models, and methods that are employed by the reasoning methodology. It consists of seven parts, as depicted at the bottom of Figure 1.2 (cf. Chapter 5).

- **Reasoning methodology** Formalizes the reasoning activity as a "comprehensive integral series of techniques and methods creating a general system theory" [199] (cf. [315]), considering a method as a "system of rules and guidelines for a consistent procedure" [315, p. 41]. It consists of three phases, as depicted at the top of Figure 1.2 (cf. Chapter 6).

A declared research objective is the achievement of a generic and widely applicable process model. Nevertheless, based on this generic process model, specialized forks can be derived for a purposely narrowed scope that favor accuracy before portability in the general trade-off motivated in Section 4.1. For instance, adapt the reasoning process model for exclusive reasoning about upcoming mobile IT infrastructures [395] or embedded systems. Although a fork would be less generic than the presented process model, it could benefit when reasoning about the targeted elements, e.g., potential globally valid assumptions could be filled out in advance.

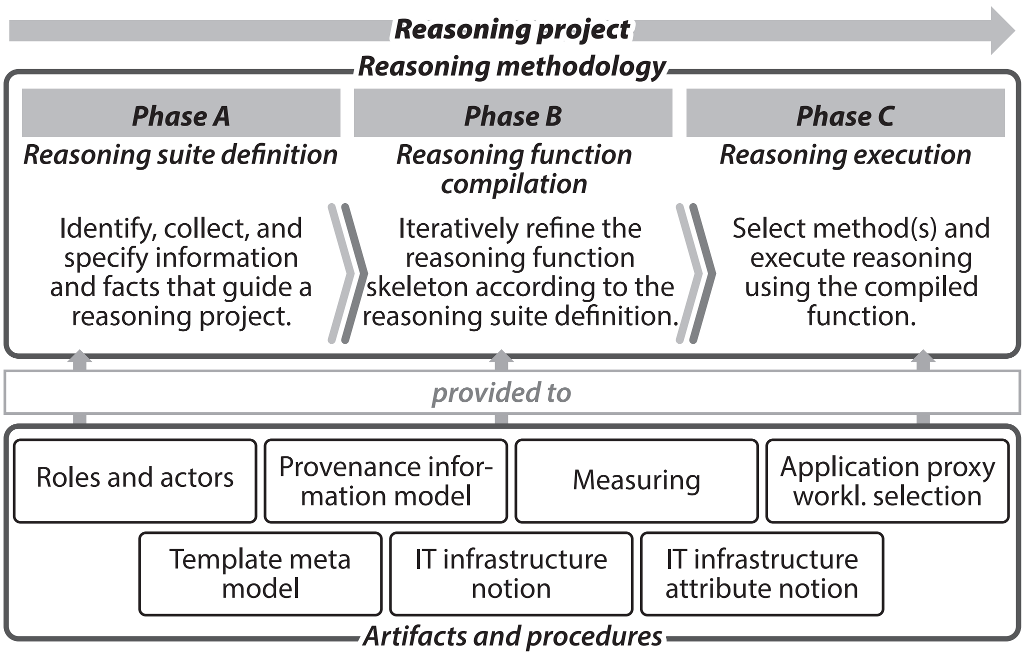

Reasoning project
Reasoning methodology
Phase A
Reasoning suite definition
Identify, collect, and specify information and facts that guide a reasoning project.
Phase B
Reasoning function compilation
Iteratively refine the reasoning function skeleton according to the reasoning suite definition.
Phase C
Reasoning execution
Select method(s) and execute reasoning using the compiled function.
provided to
Roles and actors
Provenance information model
Measuring
Application proxy workl. selection
Template meta model
IT infrastructure notion
IT infrastructure attribute notion
Artifacts and procedures

Figure 1.2: Overview of the presented process model for the integrated reasoning about quantitative IT infrastructure attributes.

1.3.2 Requirements specification

→ *Fig. 1.1 (p. 6)*

A successful and systematic development as well as result validation and assessment prerequisite a requirement set [342, 233, 397]. Thus, research extracted requirements for the integrated reasoning about quantitative IT infrastructure attributes, which also precede the presented process model:

- **Morphological field** An analytic method to capture a (complex) domain and to consider all possible characteristics [356]. Research formalized a morphological field, consisting of nine dimensions and discrete values, which collectively describe research application domain characteristics that are important for developing the process model (cf. Chapter 2). The morphological field scopes requirements engineering efforts, and assures research relevance by framing research activities to the identified needs [187].

- **Requirements specification** A "specification for a particular [...] product [...] that performs certain functions in a specific environment" [374, p. 3]. The thesis presents a *Requirements Specification* (RS) that describes in 18 functional and eight non-functional requirements, what the final reasoning process model is expected to do. The contained requirements originate from an adjusted *Use Case Analysis* [205] that compiles, analyzes, and documents requirements based upon user needs [200] derived from two real world scenarios:

World-class supercomputer Analyzes reasoning about the award-winning three PetaFlop/s HPC system *SuperMUC* (cf. Section 3.2.1).

European-wide execution of hydro meteorology simulations Analyzes reasoning about the *Distributed Research Infrastructure for Hydro-Meteorology* (DRIHM) that enables the European-wide chained execution of *Hydro Meteorology* (HM) simulations (cf. Section 3.2.2).

1.3.3 Publications

→ *Fig. 1.1 (p. 6)*

The subsequent lists itemize in chronological descending order all publications published in the context of this thesis. The first list provides publications where I was the main author, meaning I contributed the majority of the scientific contribution, and co-authors supported with their experience. The second list provides publications I contributed to as co-author. Each list entry summarizes the publication's results, highlights the respective contribution to this thesis (indicated in parentheses), and has a reference to the detailing bibliography entry.

Main author

All publications in the following list collectively form the motivation (Section 1.1), and formulate the research question (Section 1.2) of this thesis.

- C. Straube, W. Hommel, and D. Kranzlmüller. **"Design Criteria and Design Concepts for an Integrated Management Platform of IT Infrastructure Metrics"** [1] – The invited journal article deals with the management of IT infrastructure metrics, targeting a high-level management cockpit. It uses findings about quantitative IT infrastructure attributes and in particular reliability presented in [6] (cf. below). The article covers groundwork about IT infrastructures (Section 2.2) and measurements (Section 5.5), and it considers the SuperMUC at the LRZ as a scenario (Section 3.2.1). Parts of the presented information model form the basis of the provenance information model (Section 5.3). Although valid for the discussed management cockpit, the presented bottom-up aggregation of metrics is not used in this thesis, as explained in Section 4.3.

- C. Straube and D. Kranzlmüller. **“A Meta Model for Predictive Analysis of Modifications on HPDC Infrastructures”** [4] – The conference paper bases upon the *Predictive Modification Effect Analysis* (PMEA) introduced in [5] (cf. below), and derives an UML meta model for describing IT infrastructures and their attributes in a way that facilitates PMEA. The paper covers the general notion of IT infrastructures (Section 5.4), consisting of the graph based notion, the black box approach, and the respective parts of the provenance information model (Section 5.4.5).

- C. Straube and D. Kranzlmüller. **“An Approach for System Workload Calculation”** [5] – The conference paper addresses the central role of workload for quantitative IT infrastructure attributes and introduces *Predictive Modification Effect Analysis* (PMEA). The paper discusses the basic notion of workload (Section 2.3), and load (Section 2.3.3), and it introduces IT infrastructure description terms (Section 5.4.4). The very specific focus of the presented workload calculation approach is not compliant to the process model objectives (cf. Section 4.1), what disqualifies its integration in the process model in this thesis. Instead, the workload calculation approach is referenced in Section 6.3.2 as a potential data source.

- C. Straube and D. Kranzlmüller. **“Model-Driven Resilience Assessment of Modifications to HPC Infrastructures”** [6] – The conference paper bases on the *Capability Effect Analysis* (CEA) introduced in [380] (cf. below), and it focuses on the quantitative IT infrastructure attributes reliability and availability. The paper covers the general understanding of reliability (Section 2.4.3), and provides groundwork for parts of the provenance information model related to quantitative IT infrastructure attribute modeling (Section 6.1.3).

- C. Straube and D. Kranzlmüller. **“A Generic Capability Model for Analyzing Modification Effects in HPC Infrastructures”** [380] – The poster introduces the *Capability Effect Analysis* (CEA), the predecessor of PMEA. Besides, it describes a first version of the provenance information model to describe an IT infrastructure (Section 5.4.5), and introduces the idea of decomposing quantitative IT infrastructure attributes into a generic concept and several instances (Section 5.6).

- C. Straube and D. Kranzlmüller. **“An IT-Infrastructure Capability Model”** [7] – The poster presents an early version of the addressed research question (Section 1.2), and discusses related research fields. The presented simulation based approach was not followed in this thesis, because it contradicts the process model objectives (cf. Section 4.1), particularly because it is not able to address the complexity and extend of the considered IT infrastructures, as explained in Section 6.2.4.

- C. Straube, W. Hommel, and D. Kranzlmüller. **"A Platform for the Integrated Management of IT Infrastructure Metrics (Best Paper Award)"** [9] – The paper is the basis for the aforementioned invited journal article [1]. It contains a result subset, which is extended and revised by the journal article. Consequently, the paper also deals with the management of IT infrastructure metrics, targeting a high-level management cockpit. The paper focuses on metrics regarding the quantitative IT infrastructure attribute energy efficiency (Section 2.4.1), and it highlights research questions related to the management cockpit, which partly contributed to the research question addressed in this thesis (Section 1.2).
- C. Straube, M. Schiffers, and D. Kranzlmüller. **"Determining the Availability of Grid Resources using Active Probing"** [10] – The paper presents the results of my diploma thesis [379] in order to make it available to a broader scientific public. The result is an active probing architecture to determine the availability of resources in a Grid IT infrastructure. The paper discusses Grid IT infrastructures (Section 2.2.3), and the quantitative IT infrastructure attribute availability (Section 2.4.3).
- C. Straube and A. Schroeder. **"Architectural Constraints for Pervasive Adaptive Applications"** [11] – The conference paper presents an internal *Domain Specific Language* (DSL) to define constraints about the components of mobile software. The paper is only indirectly related to the thesis, as it covers mobile IT infrastructures, which are excluded from consideration as mentioned in Section 1.3.1.

Contributing author

The first four publications in the following list describe aspects of the *Distributed Research Infrastructure for Hydro-Meteorology* (DRIHM) project [98]. For these papers, I developed together with the project partners the presented DRIHM Distributed Computing Infrastructure, formulated the "DRIHMification" process, and contributed to the described DRIHM science gateway. All publications collectively contribute to the DRIHM scenario (Section 3.2.2).

The fifth publication deals with the special IT infrastructure type of Exascale systems and contributes detail aspects to the terminology (Section 2).

- D. D'Agostino, A. Clematis, A. Galizia, A. Quarati, E. Danovaro, L. Roverelli, G. Zereik, D. Kranzlmüller, M. Schiffers, N. gentschen Felde, C. Straube, A. Parodi, E. Fiori, F. Delogu, O. Caumont, E. Richard, L. Garrote, Q. Harpham, H. Jagers, V. Dimitrijevic, and L. Dekic. **"The DRIHM Project: A Flexible Approach to Integrate HPC, Grid and Cloud Resources for Hydro-Meteorological Research"** [107].

- E. Danovaro, L. Roverelli, G. Zereik, A. Galizia, D. D'Agostino, G. Paschina, A. Quarati, A. Clematis, F. Delogu, E. Fiori, A. Parodi, C. Straube, N. Felde, Q. Harpham, B. Jagers, L. Garrote, L. Dekic, M. Ivkovic, O. Caumont, and E. Richard. **"Setup an Hydro-Meteo Experiment in Minutes: the DRIHM e-Infrastructure for Hydro-Meteo Research"** [109].

- A. Galizia, D. D'Agostino, A. Quarati, G. Zereik, L. Roverelli, E. Danovaro, A. Clematis, E. Fiori, F. Delogu, A. Parodi, C. Straube, N. Felde, M. Schiffers, D. Kranzlmüller, Q. Harpham, B. Jagers, L. Garrote, V. Dimitrijevic, L. Dekic, O. Caumont, and E. Richard. **"Towards an Interoperable and Distributed e-Infrastructure for Hydro-Meteorology: the DRIHM Project"** [2].

- A. Parodi, N. Rebora, E. Fiori, F. Delogu, F. Pintus, D. Kranzlmüller, M. Schiffers, N. Felde, C. Straube, A. Clematis, D. D'Agostino, A. Galizia, A. Quarati, E. Danovaro, O. Caumont, O. Nuissier, V. Ducrocq, É. Richard, L. Garrote, M. C. Llasat, Q. Harpham, H. R. A. Jagers, A. Tafferner, C. Forster, V. Dimitrijevic, L. Dekic, and R. Hooper. **"The DRIHM Project: Building on Cutting-Edge Information and Communication Technology to Advance Hydro-Meteorological Research"** [3].

- C. Straube, A. Bode, A. Hoisie, D. Kranzlmüller, and W. Nagel. **"Dagstuhl Manifesto – Co-Design of Systems and Applications for Exascale"** [8].

1.3.4 Arrangement of research results

→ *Fig. 1.1 (p. 6)*

Figure 1.3 surveys a comparison of existing approaches, labeled by solid gray bars, with results presented in this thesis, labeled by hatched bars (cf. Section 7.4). The subsequent list explains the depicted five comparison dimensions and highlights the limitation of existing approaches. The presented process model is not constrained in these dimensions due to compilation of an individual reasoning function that enables coverage of (theoretically) infinite sets, respectively.

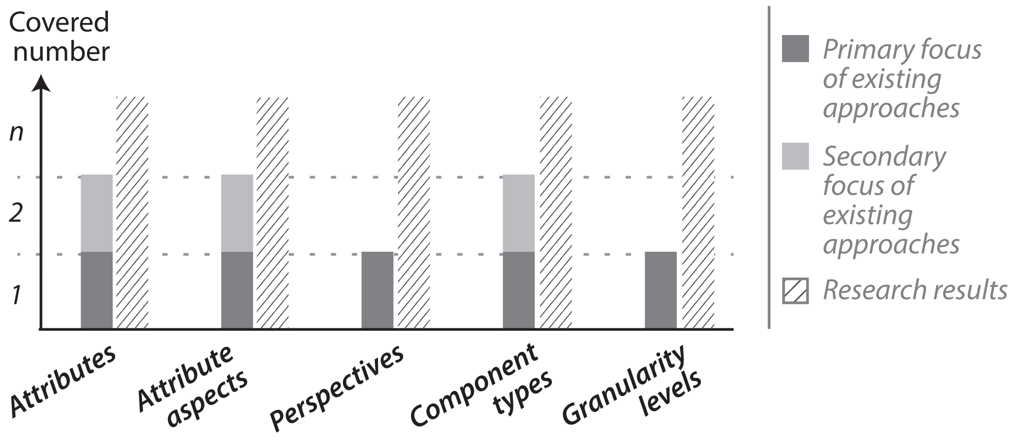


Figure 1.3: Comparison of existing approaches with presented research results.

- **Attributes** Number of simultaneously covered attributes and *inter*-attribute correlations. Most existing approaches support only one attribute, a small set also supports two attributes and their relations, e.g., the power consumption and response time in a virtualized server system [314].

- **Attribute aspects** Number of simultaneously covered aspects of a single attribute and *intra*-attribute correlations (cf. Section 2.4). Most existing approaches focus on only one aspect, like the I/O performance of HPC applications [360]. Only a small set also covers multiple aspects, e.g., by applying a top-down approach [362].

- **Perspectives** Number of simultaneously applicable perspectives during reasoning execution. A perspective describes considered elements and relevant attributes to emphasize differing focal points: the consumer perspective might grant major importance to workload execution (*elements*) and its performance (*attribute*), whereas the provider perspective might be mainly interested in IT infrastructure components (*elements*) and cost-reducing energy efficiency (*attribute*). Existing approaches apply mostly one perspective, e.g., provisioning related hard-disk power consumption [432], or use related HPC kernel power consumption [389].

- **Component types** Number of simultaneously covered IT infrastructure component types, prerequisited by the close collaboration of components to render an IT infrastructure's capabilities. Most existing approaches cover only a small set of IT infrastructure component types.

- **Granularity levels** Number of simultaneously covered granularity levels within the (reasoning) model. Existing approaches focus on only one level, either a (very) low or high granularity level, e.g., modeling power consumption of a processor [104] or a data center [138], respectively.

1.4 Research methodology and thesis structure

Based on Simon et al. [367], March et al. [266] split (IT) research activities in two (interacting) paradigms:

- **Natural Science** A descriptive discipline that discovers (natural) phenomena, compiles theorems, and justifies them [218] to explain how and why things are and to understand the nature of IT [183].

- **Design Science** A prescriptive discipline that builds and evaluates IT artifacts intended to attain goals and to meet or solve identified needs or problems, respectively [368, 187]. IT artifacts are *constructs* (vocabulary & symbols), *models* (abstractions & representations), *methods* (algorithms & practices), and *instances* (implemented & prototype systems) [266, 187].

The research question introduced in Section 1.2 calls for a solution that facilitates reasoning about quantitative IT infrastructure attributes. This recommends the *Design Science* paradigm as principle for the presented process model, especially due to its problem-solving nature, the resulting IT artifacts, and its general objective to create technical capabilities and products for accomplishing the analysis, implementation, and use of IT systems [117, 392]. Hevner et al. [187, 186] developed a framework for "understanding, executing, and evaluating [Design Science] research" [187, p. 79], which is applied as methodical course of research. Figure 1.4 (adapted from [186, Figure 1]) depicts the framework's three realms from left to right:

- **Environment** Defines the research project's application domain or problem space of the phenomena of interest [368]. It comprises interacting people, organizations, and existing or planned technical systems [365].
- **Design Science Research** Utilizes requirements identified in the *Environment* for iterative artifact building, evaluation, and refinement.
- **Knowledge Base** Comprises scientific theories and engineering methods for building and evaluation. Besides, it contains experiences and expertise about the state-of-the-art in the application domain and existing (meta) artifacts and processes, created in previous research [201, 186]. Both are provided to the other two realms.

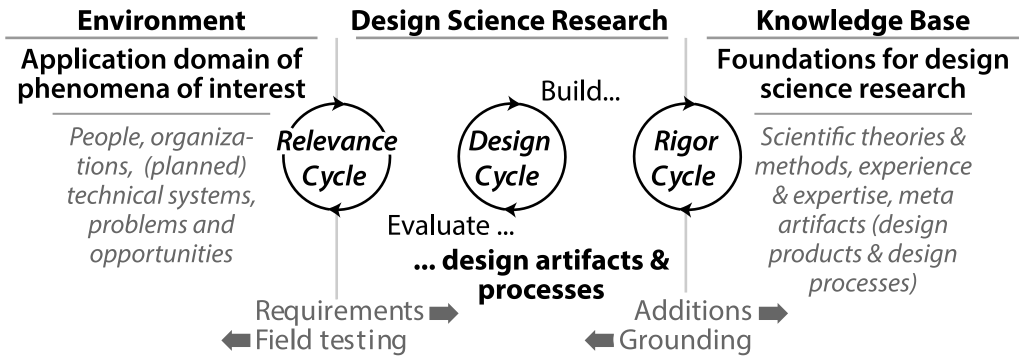

Environment
Design Science Research
Knowledge Base
Application domain of phenomena of interest
Build...
Foundations for design science research
People, organizations, (planned) technical systems, problems and opportunities
Relevance Cycle
Design Cycle
Rigor Cycle
Scientific theories & methods, experience & expertise, meta artifacts (design products & design processes)
Evaluate ...
... design artifacts & processes
Requirements
Field testing
Additions
Grounding

Figure 1.4: Elements of the applied Design Science paradigm (adapted from [186, Figure 1]).

Three cycles connect the introduced realms, namely the *Relevance Cycle*, the *Design Cycle*, and the *Rigor Cycle* (cf. Figure 1.4), that also structure the thesis. The iterative and parallel nature of the three cycles prohibits a simple mapping onto the sequential chapter structure. Instead, cycles are addressed by several (non-consecutive) chapters. Figure 1.5 overviews the thesis' chapters, summarizes chapter purposes and developed IT artifacts, respectively, and illustrates the mapping of the *Design Science* paradigm's iterative activities to the sequential chapter structure. The subsequent list explains the cycles and the related chapters, respectively:

- **Relevance Cycle** A (good) design science research undertaking should start with the **identification and examination of gaps relating to a goal** in an actual application environment [186] in order to ensure a goal-oriented research activity and relevance. Thus, the *Relevance Cycle* extracts and aggregates the (implicitly) contained goals, tasks, problems, and opportunities [187] within the *Environment*. It results in an application context consisting of requirements for the research and a set of acceptance criteria for the ultimate evaluation of the research results [186].

  Chapter 1 Introduces on an high abstraction level the research context, identifies gaps, and extracts the research question, each with relation to reasoning about quantitative IT infrastructure attributes.

  Chapter 2 Examines in an in-depth overview the *Environment* to ensure a common understanding, to foster a clear terminology, and to extract a morphological field (cf. Section 1.3.2) that scopes research in general and requirement specification in particular.

  Chapter 3 Conducts a Use Case analysis on real world scenarios (cf. Section 1.3.2) to provide a requirements set to the *Design Cycle*.

- **Design Cycle** Based on identified gaps, the *Design Cycle* **develops a set of IT artifacts to close identified gaps**. In particular, it generates design alternatives and evaluates them against requirements until a satisfactory design is achieved [368].

  Chapter 4 Motivates and explains the research main approach, the process model underlying design concepts and implementations, and outlines the presented process model (cf. Section 1.3.1).

  Chapter 5 Develops the process model's artifacts and procedures that form the basis for the reasoning methodology (cf. Figure 1.2).

  Chapter 6 Develops the process model's reasoning methodology that employs the artifacts and procedures (cf. Figure 1.2).

- **Rigor Cycle** For the *Design Science* paradigm, the "essence [...] lies in the scientific evaluation of artifacts" [201]. Thus, the *Rigor Cycle* **executes validation methods** on the presented process model to "rigorously demonstrate [its] utility [and] quality" [187, Table 1, p. 83].

  Chapter 7 Validates the presented process model, using 1) a *controlled experiment* to analyze the process model's feasibility and broad applicability, using 2) a *field study* to "monitor use of [the process model] in multiple projects" [187, p. 86], and using 3) a *related work analysis* to build arguments for the process model's utility.

Chapter 8 Concludes and summarizes the thesis and research results, respectively, and discusses the refinement and reassessment process of the developed IT artifacts [187].

The Knowledge Base relevant to the thesis is provided in Appendix A.

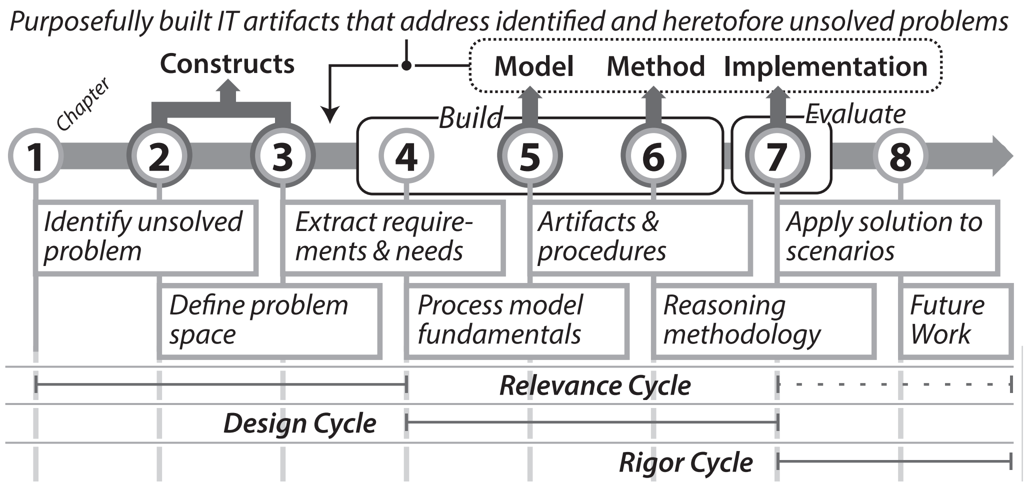

Purposefully built IT artifacts that address identified and heretofore unsolved problems
Constructs
Model
Method
Implementation
Chapter
Build
Evaluate
1
2
3
4
5
6
7
8
Identify unsolved problem
Extract require-ments & needs
Artifacts & procedures
Apply solution to scenarios
Define problem space
Process model fundamentals
Reasoning methodology
Future Work
Relevance Cycle
Design Cycle
Rigor Cycle

Figure 1.5: Overview of thesis' structure, chapter purposes, and developed IT artifacts, respectively.

Chapter 2

Environment – Terminology and research scope

This chapter discusses the research *Environment* and extracts a morphological field for scoping requirements analysis and research activities. Section 2.1 motivates the discussion and introduces the applied approach, before Section 2.2 and 2.3 consider IT infrastructures from a provider and consumer perspective, respectively. Afterwards, Section 2.4 discusses IT infrastructure attributes, and Section 2.5 presents the extracted morphological field.

2.1 Discussion objectives and approach

Extracting reasonable requirements in the *Relevance Cycle* (cf. Section 3), and guiding IT artifact development in the *Design Cycle* (cf. Section 4 to 6) mandatorily prerequisite a profound understanding and a clear notion of IT infrastructures, their exposed characteristics, attributes, and use, to avoid inappropriate inferences or artifacts. In order to achieve these objectives, the chapter performs two tasks:

- Provide a *discussion of the research environment* (its application domain), as further motivated and detailed in Section 2.1.1.
- Conduct an *extraction of a morphological field* for scoping the requirements analysis and research, as further motivated and detailed in Section 2.1.2.

Both tasks are achieved by gathering existing notions from multiple sources, since no source is superior to the others [426], and "different sources reduce possible biases during the research process" [90, p. 634]. The chapter's consideration and discussion of established, wide-spread, and current facts, experiences and expertise within the application domain might hypothesize to put this content in the Knowledge Base in Appendix A (cf. Section 1.4). Though, it is purposely discussed in this chapter to facilitate an extensive requirements analysis and provide foundation for successive chapters and sections, indicated by a right arrow and the targeted section, e.g., →Section 2.

2.1.1 Discussion of the research environment

The lack of a commonly accepted IT infrastructure definition, and manifold ways of defining IT infrastructure attributes require a thorough discussion of the research *Environment*.

The extent of IT infrastructures and their involvement in a variety of areas and provisioning paradigms cause a term overloading. Furthermore, each discipline employing and relying on IT infrastructures applies a (slightly) different notion. This situation results in a mass of definitions, covering several focal points and granularity levels (e.g., [131, 77, 335, 95, 176, 406, 141]). Yet, no definition is commonly accepted as standard or widely applied [244, 410]. This produces a lack of a domain-spanning, formal definition that could be used out-of-the-box for the presented process model [257].

The lion's share of definitions in common is the assignment of an "enabling foundation role" to IT infrastructures.[1] The role's point of reference is defined in a relative way, as Figure 2.1 depicts on its left hand side in the form of a containment hierarchy (*EG-2.1*), labeled as *Generic view*. In other words, "what is infrastructure depends on where in the organization you are placed" [410, p. 9]. Since this situation prohibits a fixed point of view, the chapter discusses IT infrastructures from two perspectives, as indicated at the very top and bottom of Figure 2.1:

- **Provider perspective** Covers component types, architectures, and provisioning paradigms, as further detailed in Section 2.2.
- **Consumer perspective** Covers workload that is executed on IT infrastructures, as further detailed in Section 2.3.

IT infrastructure attributes are orthogonal to the IT infrastructure layers, as depicted in Figure 2.1 on its left side. In contrast to the notion of IT infrastructures, there is a (high level) common sense about them. Yet, the variety and extent of IT infrastructure attributes and the complexity of their

[1]This notion is in line with the general term's origin, i.e., being composed of "infra" (lat. "beneath", "under") and "structure", leading to "beneath the structure" [244, 280].

assembly require a detailed discussion to ensure a common understanding, also on a low level, as further detailed in Section 2.4.

On its right hand side, Figure 2.1 depicts two exemplary instances of the relative IT infrastructure understanding. *Example A* (according to Laan [244, p. 36]) highlights the involvement of "several" IT infrastructures in application execution. For instance, the *software* IT infrastructure relies on security functionality provided by the *platform* IT infrastructure, which in turn uses the communication functionality of the *hardware* IT infrastructure in a transparent way, respectively [24]. *Example B* (according to Weill et al. [411, Fig. 1]) illustrates the deployment of an IT infrastructure at multiple levels. Both examples emphasize the compulsory consideration of an IT infrastructure relative to the applying IT system [410]. An example not covered by Figure 2.1 are *Content Delivery Networks* (CDN) that "place web server clusters across the globe [...] to improve [...] responsiveness and locality" [76, p. 14] of web content. Hence, a CDN builds on an IT infrastructure to assemble another one that is in turn used for website delivery. *EG-2.1*

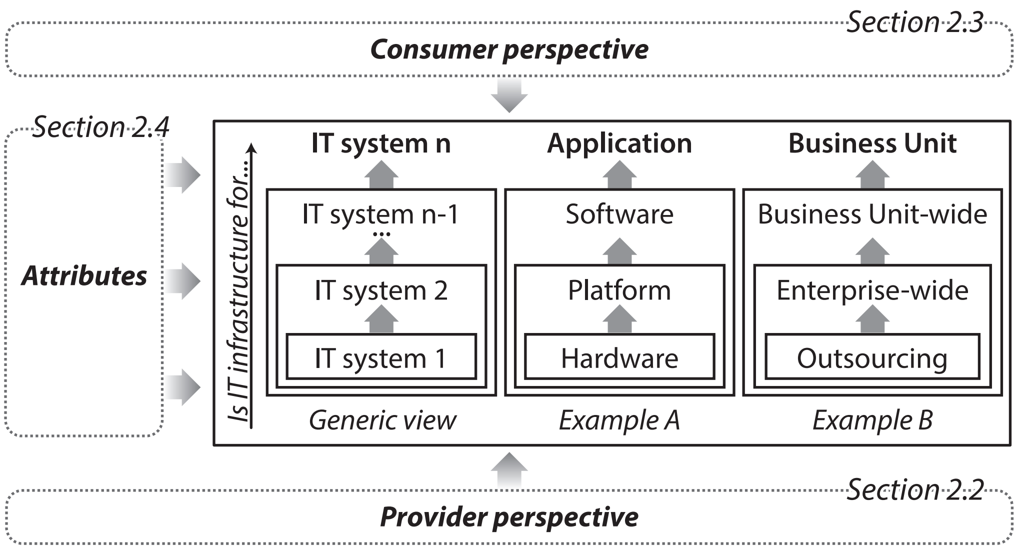


Figure 2.1: The notion of IT infrastructures being an "enabling foundation" requires a consideration from a provider and a consumer perspective. IT infrastructure attributes are orthogonal to the "enabling foundation" containment hierarchy.

2.1.2 Extraction of a morphological field

The introduced research *Environment* exposes countless characteristics, but requirements engineering and IT artifact development can reasonably use and handle only a (small) subset. The chapter addresses this situation by extracting a *morphological field*, an analytic method to capture a (complex) domain and consider all possible characteristics [356]. In particular, it is a list of dimensions and possible discrete values, respectively. Discretization of characteristics to dimensions and values allows a focused requirements analysis in Chapter 3, because only the discrete values have to be considered, instead of a theoretically countless set of aspects. Besides, it sets the research scope, since it explicitly defines, which IT infrastructure characteristics have to be covered by the *Design Cycle* and the developed IT artifacts, and which characteristics are out of scope. A characteristic extraction in the chapter's text is indicated by $\xrightarrow{MF}$*dimension name*. To give an idea of the final morphological field, Table 2.1 depicts the dimensions and values, the chapter will extract piece wise while considering the research *Environment*. Section 2.5 explains the morphological field and contained dimensions and values in detail, respectively.

Dimension	**Value**		
Perspective	Provider	Consumer	
Scale	Low	High	
Heterogeneity	Low	High	
Federation	No	Yes	
Dynamics	Low	High	
Distribution	Regional	National	International
Administrative Units	1	n	
Application Type	Parallel	Distributed	
Attribute	Performance	Energy efficiency	Reliability

Table 2.1: Morphological field, the chapter will extract piece wise while considering the research *Environment*.

2.2 IT infrastructures from a provider perspective

→ *Fig. 2.1 (p. 21)*

From a provider $\xrightarrow{MF}$*Perspective*, IT infrastructures differ in terms of contained components, component types, the arranging architectures, and provisioning paradigms. The section discusses these differences following a containment hierarchy and a decreasing granularity level. On each level, commonly contained components and prevalent applied architecture(s) are discussed. Figure 2.2 overviews the resulting section structure and arranges considered granularity levels and the containment hierarchy:

- Section 2.2.1 starts with a **single system** and contained components.
- Section 2.2.2 describes a group of linked single systems, called a **cluster**, and especially *High Performance Computing* (HPC) clusters and the sub type of so-called custom-built *supercomputers*.
- Section 2.2.3 describes **Grids**, which integrate, among others, all of the aforementioned systems and architectures.

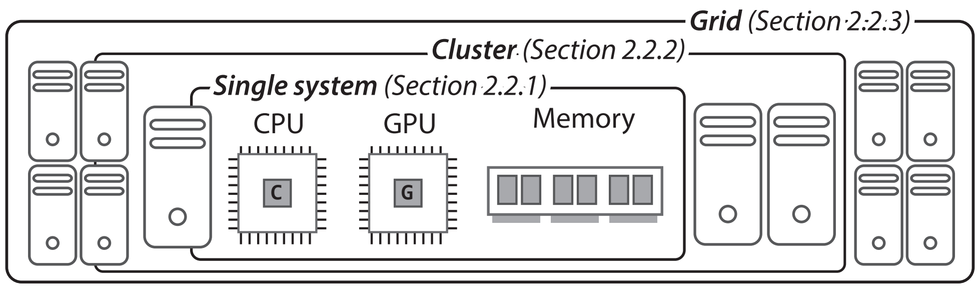


Figure 2.2: Containment hierarchy of discussed IT infrastructure elements from a provider perspective and resulting section structure.

According to the chapter's objective of considering details relevant to the research (cf. Section 2.1), the section purposely focuses on a set of details. For an extensive discussion the reader is referred to Hennessy et al. [184] and Tanenbaum et al. [384] for computer architecture, to Bauke et al. [38] and Buyya [75] for cluster computing, and to Foster et al. [147] for Grids. Despite their perceived omnipresence, Cloud systems are not considered, due to the manifold open research issues caused by the underlying virtualization concepts. Instead, virtual systems in general and Clouds in particular are discussed for future work in Section 8.2.3. Similar reasons constitute the non consideration of the building that houses the IT infrastructure as further detailed in Section 8.2.4.

2.2.1 Single system

→ *Fig. 2.2 (p. 23)*

The section covers a single system at the center of the containment hierarchy in Figure 2.2. According to the *Von Neumann architecture* (VNA) [294], Figure 2.3 (adapted from [384]) abstractly arranges the three logical components, a single system can be split in:

- **Processing Unit** Executes program code and calculations, mostly as *Central Processing Unit* (CPU) or *Graphics Processing Unit* (GPU).
- **Memory hierarchy** Provides storage for the processed and program data in differing capacities and rates.
- **Bus** Connects processing unit(s) and memory hierarchy elements.

Figure 2.3 particularly focuses on the *logical* component partitioning, instead of the physical design and assembly, in order to highlight the involved concepts. The depicted caches, for instance, are placed in the memory hierarchy, although they are physically part of a CPU or GPU.

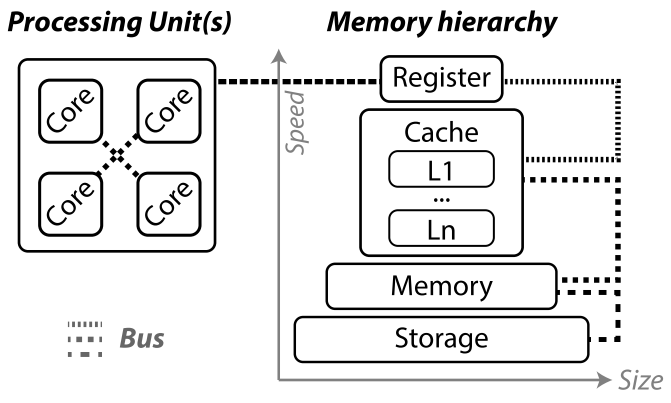


Figure 2.3: According to the *Von Neumann architecture*, a single system can be logically split in processing unit(s), memory hierarchy, and bus.

Processing Unit

→ *Fig. 2.3 (p. 24)*

A processing unit, also called microprocessor[2], is the hardware component that performs the basic arithmetical, logical, and input/output operations of the system. A microprocessor's characterization is twofold:

[2]Although its widespread use, the naming of *microprocessor* is not fully correct, since modern processing units contain additional components, like caches and bus control [54]. However, for the pursued chapter's level of detail, both terms can be used synonymously.

- **Instruction** Is the fundamental concept of a microprocessor's work, and describes an (atomic) operation the microprocessor carries out and exhibits to the programmer. An instruction, in turn, can be split in a couple of phases, depending on the processed data an the instruction's purpose (*EG-2.2*). Figure 2.4 arranges the correlation between time and the just outlined concepts: a *cycle* is the time span between two instruction execution starts, the *clock speed* specifies how many cycles per second are executed.

> The `add` instruction sums two Integer, Float or Decimal values, depending on the instruction set at hand. It consists of three phases: The **fetch** phase reads the instruction's input values from memory and places them in the processor's internal storage, which can be either a stack, an accumulator, or a register (cf. [184], →Section 2.4). The **execution** phase executes the operation `sum` on these values, the final **store** phase publishes the result back in memory.

EG-2.2

- **Architecture** Describes the microprocessor's design and can be characterized by the exposed instruction set (*Complex Instruction Set Computer* (CISC) and *Reduced Instruction Set Computer* (RISC)), by its parallelization capabilities (multi-thread and multi-/many-core microprocessors), or by its support of instruction pipelining, i.e., the parallel execution of different phases (cf. Figure 2.4). A differentiation that is of special importance to HPC systems (cf. Section 2.2.2) is the separation in CPUs and GPUs. The former consists of multiple (purely) general purpose microprocessors, the latter consists of thousands of specialized cores. Especially for computing intense scientific applications (cf. Section 2.3), many-core CPUs tend to be supplemented by special purpose accelerators such as GPUs [27]. A widespread embedded systems architecture is the *Advanced RISC Machine* (ARM) architecture (→Section 7.2).

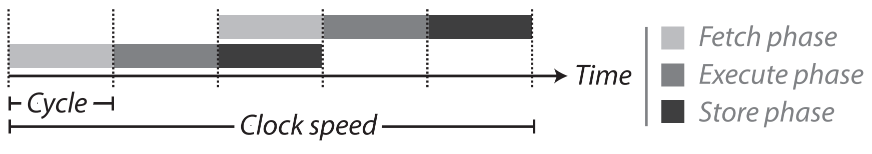


Figure 2.4: Mapping a microprocessor's instruction execution on a time axis to illustrate the correlation to cycles and clock speed.

→ *Fig. 2.3 (p. 24)*

Memory hierarchy

According to Figure 2.3, the memory hierarchy ranges from "small fast register memory to larger slower levels of cache memory to still larger and slower main memory" [73, p. 190]. In other words, the closer the memory to the microprocessor, the faster and (mostly) the smaller it is, a correlation that was shown empirically several times (*EG-2.3*).

EG-2.3

Browne et al. [73] analyzed the latency to different levels of the memory hierarchy in the SGI Origin 2000 machine and found out that CPU *registers* have a latency of 0 cycles, the *L1 cache* of 2-3 cycles, and *storage* hundreds of millions of cycles.

→ *Fig. 2.3 (p. 24)*

Bus

A bus is "a cost-saving and conventional connection structure [... being a] shared data path for a set of connected functional components" [182, p. 362]. The shared use of a bus implies a potential bottleneck, which is addressed by employing several types of specialized buses, e.g., a processor internal bus, a processor bus, a memory bus, and a system bus [182]. Bus selection and processor architecture have an affect on the overall system design, e.g., requiring a *Northbridge* and a *Southbridge* or substituting these chipsets (partially), e.g., by Intel's *Quick Path Interconnect* (QPI) [21].

→ *Fig. 2.2 (p. 23)*

2.2.2 Cluster

The section discusses the cluster, a group of linked single systems (cf. above), servers or workstations that operate collaboratively to achieve a certain goal. The term *cluster* primarily focuses on the architecture of a system and describes "a type of parallel and distributed system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource" [320, 75, 76]. The stand-alone elements, collectively called *nodes*, are mostly *commodity-off-the-shelf* (COTS) hardware [76] exposing the following characteristics:

- Highly homogeneous hardware that runs the same type of operating system [385], resulting in a low resource $\xrightarrow{MF}$*Heterogeneity.*
- Located at the same data center, exposing a regional $\xrightarrow{MF}$*Distribution.*
- Ownership by a single institution [76] and centralized managed by one $\xrightarrow{MF}$*Administrative Unit.*

Figure 2.5 illustrates a typical cluster architecture, consisting of multiple nodes that are connected by a dedicated (high speed) network [76], depicted as dashed line. Additionally, Figure 2.5 highlights the common distinction of cluster nodes according to their main objective:

- **Head node** One or multiple head nodes, also called *master nodes*, act as interface to the user as depicted in Figure 2.5 on the left hand side. Additionally, head nodes handle job organization and task submission to the worker nodes.
- **Worker node** A (big) set of worker nodes execute the tasks assigned by the head node(s) as depicted in Figure 2.5 by the *task* rectangle.

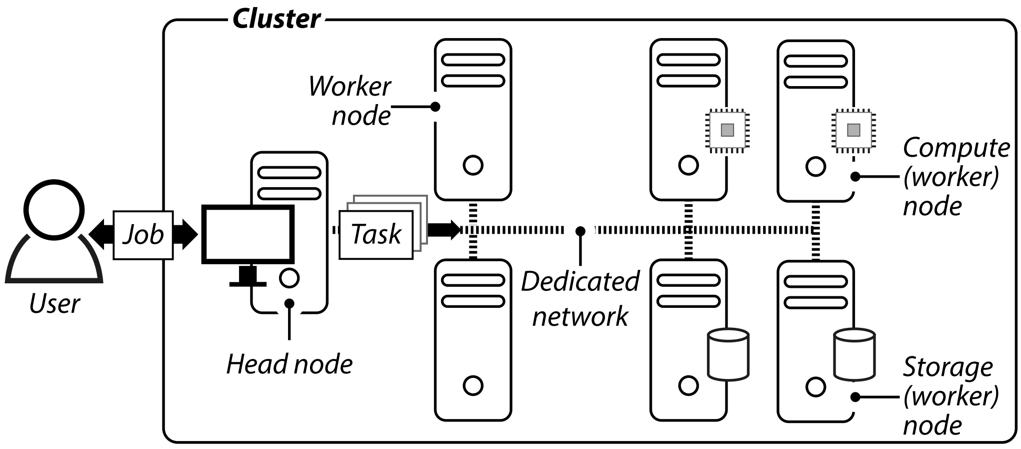


Figure 2.5: Typical cluster architecture, comprising a head node, multiple worker nodes, and a dedicated network.

The generic nature of the provided cluster definition allows a variety of comprised hardware, ranging from Beowulf clusters [40], built of mass market *Personal Computer* (PC) systems (→Section 7.2.1), up to Apple TV [157] or Raspberry Pi [396] clusters (→Section 7.2). Besides, clusters can be distinguished by their main objectives, e.g., achieving high availability or facilitating load-balancing, which results in differing configurations and framing specialized cluster types. According to the chapter's focus on aspects relevant to the presented research, two types are further detailed below:

- HPC clusters, and
- the so-called custom-built *supercomputer*, a sub type of HPC clusters.

HPC cluster

In general, the architecture of a generic cluster (cf. Figure 2.5) and an HPC cluster are very much the same [38]. In contrast, the main objective of HPC clusters (→Section 6.2.5) – attain a preferably high performance – implies the following four distinctive features (→Section 3.2.1):

- **Problem solving paradigm** The alluded main objective of HPC clusters is achieved by decomposing a compute-intense (big) problem into smaller problems that can be calculated by multiple worker nodes in parallel (→Section 2.3.1) [306, 38]. The paradigm's underlying driving force is the fact that executing a huge amount of operations on one processor, e.g., 10^{16}, is sometimes impossible and normally slower and more error-prone then executing the same amount of operations on multiple processors, e.g., 10^9 operations on 1000 processors [38].
- **Master node tasks** Decompose the compute-intense (big) problem in smaller pieces, and distribute them to the compute nodes afterwards. Distribution decisions are mostly made by a scheduler that applies a diversity of decision algorithms and tools.
- **Worker node tasks** Process the assigned small problem(s), and provide the processed data according to their *compute* and *storage* type, respectively.
- **Communication** The pursued high performance calls for a specialized network that is optimized for high performance environments, like *Gigabit Ethernet*, *Myrinet*, *Quadrics*, or *Infiniband.* Infiniband, for instance, is a point-to-point, bidirectional switched network that interconnects compute and I/O nodes [50], allowing throughput of up to 30 Gbit/s. These implementations differ in latency, bandwidth, and cost [38], as empirically investigated by Liu et al. [256]. The ever increasing speed of HPC cluster network systems, and in particular the network interfaces and links, tends to outperform the internal bus systems of the connected compute nodes [38]. Consequently, even hidden and sometimes neglected elements can have a strong impact on the overall behavior of an HPC cluster (→Section 3.4).

Supercomputer

Supercomputers (→Section 3.2.1) expose the same distinctive features as their HPC cluster super set (cf. above), while further narrowing specific characteristics. The most obvious one is the almost exclusive use of *FLOP/s* (cf. Section 2.4) to describe a supercomputer's (theoretical) peak performance, which is actually used for categorization (*EG-2.4*). A category label is composed of the performance unit and "scale", e.g., "Petascale".

The Top500 list [128] biannually collects the peak performance in *FLOP/s* of the 500 fastest supercomputers in the world. The list started in 1993 at 59.7 GigaFLOP/s and reached in 2014 33.8 PetaFLOP/s as the peak performance of the fastest supercomputer. At the time of writing, efforts in building "Exascale" systems were on their way [8, 43, 126]. *EG-2.4*

By focusing on the introduced performance description *FLOP/s*, supercomputers differ in two aspects to HPC clusters:

- **Compute nodes** To attain the extreme compute capabilities of a supercomputer, all system components and especially the compute nodes are mostly designed in a "balanced fashion to effectively match the resource requirements of the underlying application, as any architectural bottleneck will quickly render the platform intolerably inefficient" [59, Sec. 1] (→Section 3.4). Hence, a supercomputer's compute nodes are also located at the same data center, and administered by one unit (cf. above), but compared to an HPC cluster, the nodes are specifically assembled entities, instead of commodity hardware (*EG-2.5*). In particular, not only one, but up to three (slightly) differing hardware types are used, which is still considered as being homogeneous, due to the enormous scale of a supercomputer (cf. next bullet).

  The custom built nature of supercomputers takes shaping in many ways. The compute nodes of *Roadrunner*, a Petascale system at the *Los Alamos National Laboratory* (LANL), contain an equal number of conventional, general-purpose microprocessors, i.e., dual-core AMD Opteron processors, and special-purpose accelerators, i.e., two PowerXCell 8i processors [32]. The Terascale system IBM BlueGene/L [160] is based on the IBM system-on-a-chip technology and employs five interconnect networks for I/O, debug, and various types of inter processor communication. In contrast, the *Advanced Simulation and Computing* (ASCI) Purple system has "more of the look and feel of a traditional cluster, utilizing commodity processors, but uniquely designed networks" [192, p. 3]. Compared to the alluded systems, the Earth Simulator is supplied by a single manufacturer that provides a unique system [227]. *EG-2.5*

- **Scale** A supercomputer exposes a much higher $\xrightarrow{MF}$*Scale* and complexity than common HPC clusters (→Section 3.4). Especially the amount of cores, specialized interconnection networks, and storage capacities differ in several magnitudes between both (*EG-2.6*).

EG-2.6

> Instead of having about 16 processor cores, a single memory address space, and a simple, internal, all-to-all network connecting the cores, most supercomputers, like the *Roadrunner* or the *Jaguar* system at the *Oak Ridge National Laboratory* (ORNL) contain from tens to hundreds of thousands of cores, as many separate address spaces, and multiple interconnection networks with different features and performance characteristics [31].

2.2.3 Grid

→ Fig. 2.2 (p. 23)

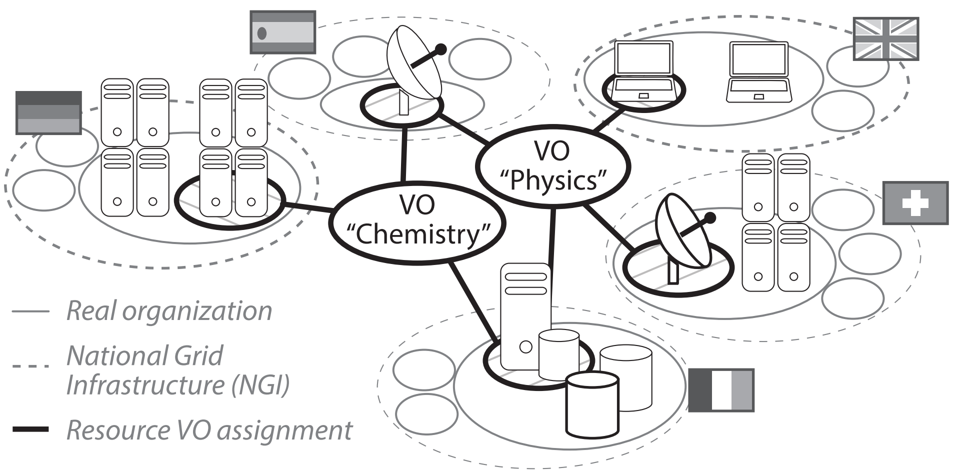
The section covers Grids as the outer level of the introduced containment hierarchy. In the beginning of Grid Computing [144] (→Section 3.2.2), HPC cluster systems being able to fulfill the usually high computation demands of scientific algorithms and questions were expensive and hard to get access to [149]. This gap initiated and drove the development of Grid infrastructures [436] that aim at "address[ing] large-scale computation problems using a network of resource-sharing commodity machines that deliver the computation power affordable only by supercomputers and large dedicated clusters at that time" [149, p. 3]. In achieving this objective, Grids integrate existing resources with their hardware, operating systems, local resource management, and security infrastructure [149], resulting in a high resource $\xrightarrow{MF}$*Heterogeneity.* For labeling this new type of infrastructure, the term *Grid* was chosen to emphasize the idea of a future in which "computing resources, compute cycles and storage, as well as expensive scientific facilities and software, can be accessed on-demand like the electric power utilities" [189, p. 1018] (cf. [188]). The common notion of Grids bases upon the definition of Foster et al., saying that Grids target at "[enabling] resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organizations" [146, 147]. Another popular, and semantically equal, but not that often cited definition is provided by Buyya et al., stating that a Grid is a "type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at run time depending on their availability, capability, performance, cost, and users' quality-of-service requirements" [76, p. 3].

Foster provides his definition's nucleus also as three point checklist [143], describing a Grid as a system exposing three characteristics (*EG-2.7*):

- **Coordination of resources not being subject to central custody** Deals with the Grid vision, i.e., "enable users to use resources from a diverse set of sites by leveraging a common set of job submission and data management interfaces across all sites" [328, p. 104]. Thus, different organizations from manifold countries use and provide resources, resulting in the following characteristic set:
  - Resources assemble an $\xrightarrow{MF}$*Inter-Organizational* environment that is based on $\xrightarrow{MF}$*Federation* and that can be organized in either a regional, national, or international geographical $\xrightarrow{MF}$*Distribution*, as done in life science disciplines since many years [241], for instance.
  - Resources are maintained by several $\xrightarrow{MF}$*Administrative Units* [333].
  - Resources expose a high resource $\xrightarrow{MF}$*Dynamics* due to the applied operations paradigm and in particular the non-centralized and autonomous administration.
- **Using standard, open, general-purpose protocols and interfaces** Addresses the high resource $\xrightarrow{MF}$*Heterogeneity* that ranges from data storage facilities, HPC clusters and supercomputers, sensors, and instruments to single notebooks and PDAs [339].
- **Delivery of nontrivial qualities of service** Requires manifold specialized areas to collaborate and interact in order to (successfully) operate and manage a Grid. The involved areas are structured in many ways according to the particular objectives and focus [148, 44, 18]. A widespread structuring is provided by the Globus Toolkit 4, dividing Grid operations in *Security*, *Data Management*, *Execution Management*, *Information Services*, and *Common Runtime* [142, Fig. 2]. According to the chapter's focus on aspects relevant to the presented research, only two operational aspects are detailed below (→Section 3.2.2). For further reading, the reader is referred to Foster et al. [146, 145, 148], and Berlich et al. [44].

  Virtual Organization (VO) An organizational structure, consisting of so-called *real-world entities* [284], like scientists, organizations, and other stakeholders, each potentially acting as resource provider and resource consumer [146] at the same time. The VO facilitates the controlled sharing of provided resources to enable real-world entities to collaborate by executing (computational) tasks on shared resources in order to achieve a common goal [418, 146, 405].

Information Service Stores and maintains meta data about Grid resources, covering resource types, access points, and (theoretically) available capabilities. In addition, it exposes a query interface to schedulers for resource discovery [431] and to monitoring tools like Nagios [202, 290] for report generation (→Section 8.2.4). A widespread information service implementation is the *Berkeley Database Information Index* (BDII) using the Grid specific *Grid Laboratory for a Uniform Environment-Schema* (GLUE) information model [22] developed by the GLUE Working Group [166] (→Section 3.3.2).



EG-2.7

The figure illustrates the covered Grid concepts and characteristics, especially resource heterogeneity, federated resource provisioning, and resource sharing within two virtual organizations, each pursuing a certain goal in the discipline of chemistry and physics, respectively. The figure also dumbs down the Grid infrastructure of Europe, called the *European Grid Infrastructure* (EGI, →Section 3.2.2), which consists of several *National Grid Infrastructures* (NGI), each resource providing country operates. The former American equivalent to EGI was TeraGrid, an "integrated portfolio of more than twenty HPC systems, several specialized visualization resources and storage archives, and a dedicated continental-scale interconnection network" [88, p. 226]. In addition to general purpose Grids like EGI and TeraGrid, there are several specialized ones, like the variety of medical grid projects [52, 238, 163, 241].

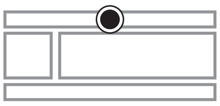
2.2.4 Morphological field for the provider perspective

While considering IT infrastructures from a provider perspective, the previous sections extracted a set of dimensions and values for the pursued morphological field, which are collected in Table 2.2.

Dimension	**Value**		
Perspective	Provider	Consumer	
Scale	Low	High	
Heterogeneity	Low	High	
Federation	No	Yes	
Dynamics	Low	High	
Distribution	Regional	National	International
Administrative Units	1	n	

Table 2.2: Morphological field of IT Infrastructures considered from a provider perspective.

2.3 IT infrastructures from a consumer perspective



→ *Fig. 2.1 (p. 21)*

The section investigates IT infrastructures from a consumer $\xrightarrow{MF}$*Perspective*, which focuses on the *workload* executed on the IT infrastructure and the caused *load* of the IT infrastructure or its components. Figure 2.6 pulls the elements together: at its top, Figure 2.6 depicts workload, which is the software to be executed on the IT infrastructure, causing load. According to literature, like Cheveresan et al. [93], workload is split in two groups:

- **Application** Is a software that solves a certain problem using computational facilities, as further detailed in Section 2.3.1.
- **Benchmark** Aims at substituting applications and at standardizing (computational) work to facilitate comparison or analysis of IT infrastructures, as further detailed in Section 2.3.2

The assignment of a piece of code to one of the two groups is quite often situation specific, as it depends on the employed code elements and objectives (*EG-2.8*). Consequently, the section introduces an individual set of distinctive features to structure IT infrastructure consideration from a consumer perspective in this thesis. In contrast, the section purposely does not present a generic structuring.

EG-2.8

The LINPACK benchmark can be understood as application if it is executed in the context of performance measurements for the Top500 list [128], or it can be understood as benchmark, if it is executed to cause load for energy consumption measurements.

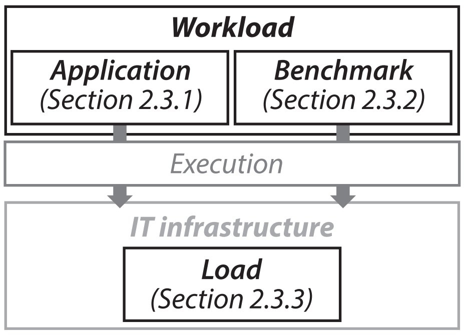


Figure 2.6: Consideration of IT infrastructures from a consumer perspective involves workload and the caused load.

2.3.1 Application

→ *Fig. 2.6 (p. 34)*

The generic notion of applications alluded in the introductory paragraph results in a theoretically infinite application set to consider, ranging from desktop tools to web platforms, and scientific problems. The section focuses on *scientific applications*, because of their challenging demands on multiple especially quantitative IT infrastructure attributes like performance, energy efficiency or reliability (cf. Section 2.4) [373]. However, the concepts, demands, and techniques described in this section apply (mostly) equally well to other application classes. Scientific applications cover a broad range of domains, like climate modeling [275, 3] (→Section 3.2.2), biological analysis [167], or computational physics [101] (cf. [212]), which is addressed by several structuring approaches that differ in focus and abstraction level (*EG-2.9*).

Employing the application's main scaling dimension, results in, amongst others, 1) data-intensive (large) scale data analysis, 2) (large) scale wide-area data transfer, 3) (large) scale amount of required computation, and 4) a (large) scale number of users [212]. Another approach splits applications according to the used source code and data types, e.g., in Floating-Point codes and Integer-Point codes [288].

EG-2.9

This section applies the structuring proposed by Jha et al. [212], because it builds upon conceptual characteristics of scientific applications, instead of implementation and programming specific details. This is of special importance to cover future developments, like potential new programming models in upcoming Exascale systems and applications [209] (cf. Section 2.2.2), and consequently, to come up with a structuring that is applicable for a rather long time horizon. Jha et al. structure applications based on made assumptions, which differ in terms of

- fault appearance and handling,
- performance measuring in time-to-solution or throughput,
- exclusive resource usage and resource sharing, and
- security threats.

Following their two $\xrightarrow{MF}$*Application types*, the section discusses *parallel* and *distributed* applications.

Parallel applications

Both in industry and science, advance and the emergence of new fields of interest cause increasingly demanding and compute-intense applications. These high performance applications are called *parallel applications*, if they can be decomposed in smaller problems to be executed (in parallel) on an HPC cluster [212] (*EG-2.10*). Hence, parallel applications mostly expose no distributed execution and can be handled by one single (big) machine, like a supercomputer (cf. Section 2.2.2).

So-called wavefront or "hyperplane" methods [193] break the computation of algorithms having recurrences into segments and pipeline (cf. above) them through multiple processors [321]. This is used for solving diverse (scientific) problems, like particle physics simulations [236], or the parallel solution of triangular systems of linear equations [325].

EG-2.10

Distributed applications

The problem solving approach in several research disciplines has changed during the last years in terms of complexity, distribution, and the amount of processed data [116]. It evolved from mostly simple (standalone) batch executions of data analysis tasks, to large scale and distributed executions of several elements, like heterogeneous and specialized programs, and data items, mostly provided by multiple disciplines [261, 282, 327, 98] (→Section 3.2.2).

Sir John Taylor [386] introduced the dedicated term "enhanced Science" (e-Science) to name the new situation as "global collaboration in key areas of science and the next generation of infrastructure that will enable it" [188, 189, 333]. In other words, the upcoming demand for collaboration, communication, and chained execution of multiple special purpose applications resulted in *distributed applications* that " need multiple resources, or would benefit from the use of multiple resources" [212, p. 1561]. Compared to parallel applications, distributed applications are not concerned with decomposing a problem in smaller tasks (cf. above), but with composing "units into a single application" [212, p. 1569]. Distributed applications and the advent of e-Science highlight two concepts:

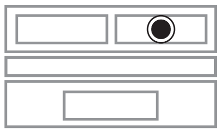
- **e-Infrastructure** There are several synonyms for the e-Science enabling "next generation infrastructures", like *cyber-infrastructures* (U.S.), or *e-Infrastructures* (Europe) [333, 328]. For a long time the (only) e-Infrastructures applied for scientific workflows were Grids (cf. Section 2.2.3) [189, 188], like the *Enabling Grids for e-Science* (EGEE), the *TeraGrid*, or the *Open Science Grid*. Most e-Infrastructures intend to integrate existing entities as well as new systems in a dynamic way [189], resulting in a global $\xrightarrow{MF}$*Distribution* [212] and a process of high $\xrightarrow{MF}$*Dynamics*, instead of an assembly in an "entirely top-down, orderly, and blueprint-like way" [133, p. 2]. This flexibility is required to be able to react on the needs of scientists, to incorporate continuity in the development of technologies, and to respect decisions often made upon a "complex web of socio-material relations" [401, Sec. 3.1].

- **Workflow** A workflow is "a collection of programs organized to accomplish some [specific goal]" [282, p. 587]. It consists of several, sometimes up to hundreds of tasks, also called *execution components* [254], being an "abstraction for the frequently occurring concept of an encapsulated and self-contained piece of work" [212, p. 4]. Besides the ordinary calculation, storage, and data transfer tasks, workflows require so-called *shimming* tasks mediating syntactically or semantically mismatching input and output data [20, 260] (→Section 3.4). Workflows are mainly grouped in two classes [214, 249, 261, 429, 420]:

Control flow Workflow dependencies describe a *control transfer* that triggers a consecutive task's execution [140]. It is built of control structures such as conditionals, loops, and iterations. Class members are mainly applied in business workflow systems [19, 140, 359].

Data flow Workflow dependencies describe a *data transfer* from a producing to a consuming task [116]. The availability of (all) input data triggers the consuming task's execution to ensure that the producer has finished before the consumer may start [116]. The movement of data through tasks determines the execution order of the whole program [140], a perfect fit to the data-intensive or even data-driven nature of scientific analyses [140] (→Section 3.2.2). Thus, data flow workflows prevail for scientific workflows [261, 415, 366].

The notion of workflows in terms of tasks and dependencies render *graphs* an efficient representation way. Consequently, most data flow oriented workflows are represented as graphs, mostly even as *Directed Acyclic Graphs* (DAG) [428, 429, 415]. Due to design errors or inaccurate workflow specifications, the execution might skip at an arbitrary point in time and require a rollback [359] (→Section 3.4). Since workflows facilitate the composition, orchestration, and management of distributed applications and their execution in manifold environments [438, 269, 114, 111, 395], they are increasingly employed to conduct complex analyses [62, 115].



→ *Fig. 2.6* *(p. 34)*

2.3.2 Benchmark

Benchmarks "emulate real-world computing tasks" [402, p. 340] to provide a standardized set of (computing) work. This standardized work is employed in several situations, ranging from analysis of a selected component set, and timing the execution of some clearly defined task [106], to assessing the relative performance of IT infrastructure components [244] (→Section 6.1.6). To cope with the variety and extend of situations, several (specialized) benchmark types apply differing levels of focus, pursue diverse objectives, and expose manifold behaviors [258, 407] (→Section 6.1.6). This section details *benchmark implementation concepts* that underlay most benchmarks, and distinguishes wide-spread *benchmark types* (*EG-2.11*).[3]

[3]Appendix B on page 277 details benchmarks that are used within this thesis.

EG-2.11

Kernel benchmarks substitute real-world applications and simulate typical usage profiles of usually executed applications (→Section 5.7), like Sweep3D and SAGE that represent typical computational and communication behavior of the ASCI workload in a balanced way [112]. Synthetic benchmarks aim at stressing selected components for further investigation, e.g., to measure power consumption, or to analyze a component's fail over rate, especially when executing a real application is too difficult or costly (→Section 6.2.4). Benchmarks employed for a particular goal are the LINPACK benchmark and the *NAS Parallel Benchmarks* (NPB) [29] for supercomputer comparison [13] (cf. Section 2.2.2).

Benchmark implementation concepts

The general intent of benchmarks to provide standardized (computational) "work" involves several tasks, particularly the formal, reproducible, and even (mathematical) provable definition of this work. Furthermore, it contains the work's description in a computer-readable way, and finally its execution on the examined IT infrastructure. Figure 2.7 arranges those tasks in four layers that form the basis for most benchmarks, consisting of the topmost *high level specification*, a set of *low level implementations*, their *compilation* resulting in a set of binaries, and finally the binaries' *execution* (→Section 6.1.6). Subsequently, the four layers are further detailed (*EG-2.12*):

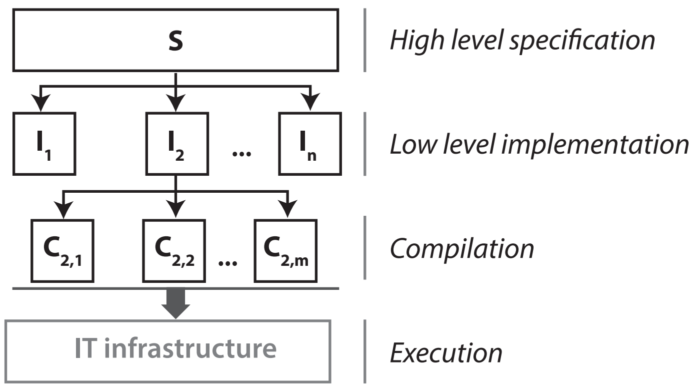


Figure 2.7: The four layers that form the basis for most benchmarks.

- **High level specification** (Semi) formally describes the overall objectives, guidelines, and limitations of the benchmark. In addition, it motivates and documents the benchmark's design, and it provides a set of arithmetic functions as well as initial values.

- **Low level implementation** Realizes the high level specification, (potentially) using multiple programming languages. The contingent strong impact of a language's inherent concepts and architecture on benchmark results calls for a thorough language selection (→Section 5.7). Further aspects that might bias the benchmark results are the language semantics, design, and general concepts (→Section 5.3).

- **Compilation** Creates an executable binary for a low level implementation. Most compilers apply differing optimization algorithms that might improve real applications, but simultaneously render component focused benchmarks useless, because "large parts of [a] program [are] not executed because they [are] not logically necessary" [106, p. 3] (→Section 6.1.7). This requires a clear-sighted compiler and configuration selection.

- **Execution** Executes the provided binaries. Execution also faces the problem of selections that might affect the benchmark results, e.g., problem size selection or input parameter files.

The high level specification of the Whetstone benchmark [106, 407] states that "the program should be complex enough to seem typical when presented to an intelligent compiler" [106, p. 3]. Consequently, it defines eleven modules in pseudo code consisting of typical code blocks, like conditional jumps, integer arithmetic, or trigonometric functions. The most widespread low level implementations are written in C and Pascal. Levy et al. [252] analyzed both and found out that "the C string search uses pointers to access characters while the Pascal program uses arrays and index variables" [252, p. 5], what underpins the required thorough programming language selection. The alluded impact of compilation was empirically evidenced by Wichmann [414], showing performance figures differing by up to 20% for two Pascal implementations, having replaced the print statements by variable checks (cf. [407]). Compared to the HPL benchmark that puts strong demands on the input parameters and the problem size, the Whetstone benchmark is very flexible in this regard. *EG-2.12*

Even though some work denotes the four layer approach as "best practice" [407], it is a trade-off: on the one hand, it supports separation of concerns, result reproducibility, various benchmark binaries, and taking compiler efficiency and machine architecture differences into account [106]. On the other hand, it entails several (difficult) decisions as discussed above.

Benchmark types

There are several benchmark types, whose characteristics can help to select a benchmark for a particular situation (→Section 6.1.6, →Section 6.2.4). Figure 2.8 depicts the five benchmark types (*EG-2.13*):

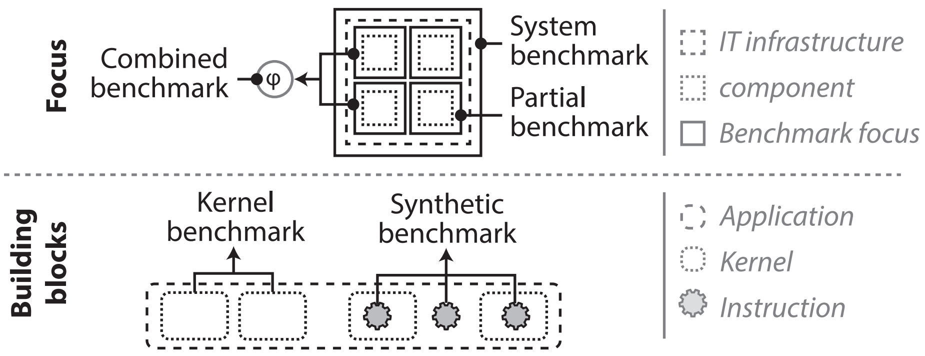


Figure 2.8: Benchmark types, derived from focus and building blocks.

- **Focus** Covers the benchmark's elements of interest. In its upper area, Figure 2.8 depicts an IT infrastructure and contained components by dashed rectangles, respectively (cf. Section 2.2). According to the considered elements, there are three benchmark types:

  System benchmark Considers the entire IT infrastructure without stressing a particular component sub set.

  Partial benchmark Focuses on a (small) sub set of IT infrastructure component types.

  Combined benchmark Employs an operation φ to combine two or multiple partial benchmarks to derive a new value.

- **Building blocks** Considers the artifacts that assemble the benchmark. In its lower area, Figure 2.8 depicts a real world application and contained kernels as dotted rounded rectangles, gear-wheels represent used instructions. According to the assembling elements, there are two benchmark types:

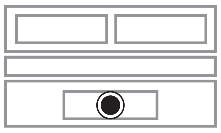
  Kernel benchmark Consists of real world production source code, also called *application kernel*. This source code focuses on a very specific field, like CFD, and was extracted by a (performance) analyst from a real application by getting rid of the rarely used 90% application source code [122]. This close relation to applications caused the advent of the synonym *mini-applications*, i.e., one or multiple kernels combined to a (small) self-contained program. A great advantage of kernel benchmarks is that they facilitate the analysis of architectural system performance under realistic I/O demands and communication patterns [59]. Besides, optimization insights gained from these studies can be fed back directly into the original application, and indirectly into applications with similar I/O requirements [59]. In contrast, the main challenge of most kernel benchmarks is that they are usually small, prone to compiler "attacks", fit in cache, and measure only CPU performance [122] (→Section 6.1.6).

Synthetic benchmark Consists of a set of (very) generic artificial code blocks, like matrix multiplications, that do not solve a (scientific) problem, but provide the tools to do so.

Exemplary **system benchmarks** are the already alluded NPB and the SPEC Benchmark suite [122], an exemplary **partial benchmark** is the *Sustainable Memory Bandwidth in High Performance Computers* (STREAM) benchmark [271] measuring sustainable memory bandwidth (in MB/s). The *memtime* benchmark underpins the potential extreme focus of partial benchmarks as it performs a memory walk with successive memory accesses [112]. Examples for **kernel benchmarks** are *Sweep3D*, implementing the main processing involved in deterministic particle transport computations [32], the *Microwave Anisotropy Dataset Computational Analysis Package* (MADCAP) [58] based *MADbench2* benchmark [59], stressing the HPC system with full computational complexity of *Cosmic Microwave Background* (CMB) data analysis, or the Livermore FORTRAN Kernels [387]. Representatives for **synthetic benchmarks** are the Dhrystone benchmark [407, 408], and the already covered Whetstone benchmark [106, 407].

EG-2.13

2.3.3 Load



→ Fig. 2.6 (p. 34)

The execution of applications and benchmarks (workload) on an IT infrastructure causes *load*. It is defined as the use level of an IT infrastructure component or the entire IT infrastructure in percent at a certain point in time in relation to its maximum capability or capacity (*EG-2.14:1*). This section investigates load and its characteristics, as the (strong) influence of load on nearly all attributes was empirically evidenced and underpinned in literature (*EG-2.14:2*).

EG-2.14

(1) A CPU load of 50% denotes that the CPU uses 50% of its floating-point operation capacity at the point in time t. *(2)* Contreras et al. state that a processor's power consumption "is greatly dependent" [104, Sec. 1] on its executing workload and hence, on its load.

A set of load values for a period of time assembles a *load profile* [336] (→Section 3.4, →Section 6.3.2), which can greatly support investigating and reasoning about quantitative attributes, as the profiles often exhibit informative "phases" [130, 203] (cf. [104]).

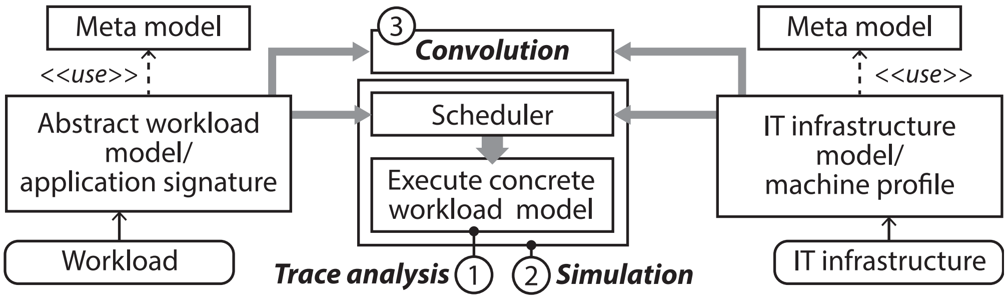

Meta model
<<use>>
Abstract workload model/ application signature
Workload
3
Convolution
Scheduler
Execute concrete workload model
Trace analysis 1
2 Simulation
Meta model
<<use>>
IT infrastructure model/ machine profile
IT infrastructure

Figure 2.9: Elements and concepts involved in analytic and physical load value derivation (adapted from [5, Fig. 1]).

Figure 2.9 pulls load derivation elements together (adapted from [5, Fig. 1]): On the left, a hardware and resource independent *abstract workload model* describes a concrete *workload*, on the right, an *IT infrastructure model* describes a concrete *IT infrastructure*. Both use language constructs provided by the correspondent *meta model*. Based on this basic notion, there are three approaches to derive load values (→Section 6.3.2):

- **Trace analysis and profiling** ① Bases upon workload execution [424, 219] while "the run time system captures important information in trace files using profiling libraries" [45, p. 3] (*EG-2.15*). In getting executable workload, the scheduler uses the abstract workload model and the IT infrastructure model to create a *concrete workload model* that binds application tasks to specific resources [429]. In other words, the scheduler maps workload tasks on services or (distributed) resources that can execute them [415, 429]. This mapping faces manifold challenges and pursues different goals [430, 416], e.g., resource availability, temporal and causality constraints, space requirements, consideration of multiple (parallel) workload, or execution costs in terms of financial expenditure, execution time, or energy consumption [359, 428, 327, 415, 113]. These challenges are addressed by several existing approaches, like operations research, constraint programming [206, 53], resource allocation constraints [358], temporal logic and specialized algebras [26, 370, 369], or Markov Decision Processes [428]. All these approaches in common is the necessity of capability and property information (cf. Section 2.4) to select appropriate resources [428] (→Section 2.4). Besides real workload execution, also recorded or synthetically generated workload [286] is employed, like the ASCI workload defined by specific engineering needs [193].

Prevalent profiling libraries are the *Portable Programming Interface* (PAPI) for performance evaluation on modern processors [73], as used in the AutoTune project [283], or the *MPI profiling API* (PMPI). In addition, several vendors provide product- or hardware-specific libraries, like the *Cray Performance Analysis Toolkit* (Cray PAT) that provides tools and an API to "collect profiling, tracing and hardware counter data as the application executes" [45, p. 2]. Hardware and performance counters are provided by modern processors exposing various data and performance events [162]. *EG-2.15*

- **Simulation** ② Is "an imitation (on a computer) of a system as it progresses through time" [334, p. 2]. The imitation aims at "predict[ing] the behavior of a system under particular circumstances when it is impossible, or at least undesirable, to experiment with the system itself" [72, p. 1]. In the context of load value derivation, simulation imitates the execution of workload, potentially including the scheduling step (cf. above), and mostly applying the same profiling algorithms on the gained data as "conventional" or physical trace analysis (→Section 3.3.2).

- **Convolution** ③ In a (pure) mathematical sense, convolution is an operation processing two functions f and g in a particular way to produce a third function h. In the context of load derivation, convolution is a "computational [algebraic] mapping of an application signature onto a machine profile" [82, p. 926] (→Section 3.3.2). An application signature is a "summary of the [fundamental] operations to be carried out by an application, including memory and communication access patterns, independent of any particular machine" [83, p. 337], a machine profile is a "characterization of the rates at which a machine can (or is projected to) carry out fundamental operations abstract from the particular application" [84, Sec. 2].

2.3.4 Morphological field for the consumer perspective

While considering IT infrastructures from a consumer perspective, the previous sections extracted a set of dimensions and values for the pursued morphological field. Table 2.3 collects the extraction results and extends the morphological field of Section 2.2, indicated by the gray and black font, respectively.

Dimension	**Value**		
Perspective	Provider	Consumer	
Scale	Low	High	
Heterogeneity	Low	High	
Federation	No	Yes	
Dynamics	Low	High	
Distribution	Regional	National	International
Administrative Units	1	n	
Application Type	Parallel	Distributed	

Table 2.3: Morphological field of IT Infrastructures summarizing a provider and consumer perspective.

2.4 IT infrastructure attributes

→ *Fig. 2.1 (p. 21)*

There is a variety of ways to describe an IT infrastructure, e.g., using its specification facts, like clock speed or allocatable amount of disk storage, or describing its application domains, e.g., solve massively parallel computational problems (cf. Section 1.1). Another and the research underlying way is the description of an IT infrastructure using three terms:

- **Capability** A qualitatively well-defined (low level) functionality that is exposed to the user or application, like data transfer, data storage, or computation. A capability can be used by a scheduler for mapping workload onto IT infrastructure components [5] (cf. Section 2.3.3).

- **Property** A quantitative and workload execution (cf. Section 2.3) agnostic description considering IT infrastructure components as white boxes at any time step, e.g., describing the theoretical maximum clock speed of a component offering computation capabilities. Property values can be gathered from vendor specifications, benchmarking, trace analysis, or any other (empirical) data source. Schedulers consume them for mapping workload onto IT infrastructure components [5] (cf. Section 2.3.3).

- **Attribute** A quantitative description of an IT infrastructure that considers its components as black boxes during workload execution at a particular time step, e.g., "power consumption in Watt per time step" or "time to completion of result calculation".

According to the research's focus (cf. Chapter 1.1), the section concentrates on quantitative IT infrastructure attributes, from now on simply called attributes, if naming conflicts can be precluded. The emphasis of on-going research activities and the explicit suggestion of several publications, like the Scientific Case [170] of the *Partnership for Advanced Computing in Europe* (PRACE), or industry surveys considering key issues for IT executives [262], call for considering the three following $\xrightarrow{MF}$*Attributes* that are consequently further detailed in alphabetical order:

- **Energy efficiency** Receives (massive) attention since power costs, especially cooling costs for supercomputers [318, 363], begun by trend to outclass (hardware) procurement costs [34, 162], and since IT infrastructure providers have to tackle the fact that in 2008 "people have begun to value the environmentally friendly attributes of IT" [289, p. 27]. In particular, "while in previous years [research] focused strongly and often solely on performance, energy efficiency has now become a second and even equally important metric" [354, p. 481].

- **Performance** Can be seen as an "old" IT infrastructure attribute that describes the success and quality of components, ranging from network devices to desktop systems and entire IT infrastructures, like HPC clusters (cf. Section 2.2.2).

- **Reliability** Describes the continuousness of delivering a correct service according to functional specifications under stated conditions over a specified period of time [137, 28]. The essential role of IT infrastructures for the everyday work in several scientific disciplines [401] and industry [156, 74] brings reliability into consideration's focus.

Although there are opinions about attribute ordering, e.g., Saini et al. state that “application performance is the ultimate measure of system capability” [349, Sec. 3], the section considers the three attributes equally. Noteworthy, they are only examples and are chosen because of their important role and prevalence. The presented research aims at covering arbitrary attribute sets (→Section 3.4, →Section 6.1.3), especially to be able to handle upcoming attributes in the context of Exascale systems (cf. Section 2.2.2). Section 8.2.1 details why *qualitative* attributes, like *Information Security* (IS) or *interoperability*, are not within the thesis' scope.

The remainder of this section details the above itemized quantitative IT infrastructure attributes by considering three aspects, respectively:

Instances Selects and discusses a (small) set of attribute instances. The extent of attribute figures implicitly leads to an abstract summarizing definition and a (theoretically) infinite set of concrete instances of this definition (→Section 5.6), e.g., performance can be boiled down to *work/time*, and prevalent instances are *FLOP/s* and *Byte/s*.

Factors and parameters Considers factors that might impact and parameters to influence the attribute in general and instance values in particular.

Role of load Discusses the role of load (cf. Section 2.3.3) for the attribute to respect the load's strong influence [173].

2.4.1 Energy efficiency

The attribute can be split in two parts: *power consumption* and the employing *energy efficiency*. The former describes the required power or electricity to perform a particular task, the latter uses the power consumption figures for comparing the efficiency of IT infrastructures and for evaluating power-aware techniques [363].

- **Instances** The instances *Cycles/Joule* and *FLOP/Watt* are detailed (→Section 3.2), because the former is highly relevant for microprocessor design, and the latter is of high relevance for the Green500 list [363] that ranks the energy efficiency of supercomputers, instead of their performance as the Top500 list [128] does (cf. Section 2.2.2).

  Cycles/Joule Counts processor cycles (cf. Section 2.2.1) that can be executed using one Joule of energy. This attribute instance is especially used in chip development.

  FLOP/Watt An instance related to the just described *Cycles/Joule* is the *FLOP/Watt* instance, counting the amount of executed FLOPs (cf. Section 2.2.1) per Watt. This attribute instance is used by the Green500 list, in the strict sense *MFLOP/Watt*, that aims at providing "a ranking of the most energy-efficient supercomputers in the world" [164] (→Section 3.2.1).

The *Power Usage Efficiency* (PUE) [264] describes "the ratio of the total energy used by the supercomputing site divided by the amount of energy that is consumed for pure computational work" [27, p. 20]. Hence, this energy efficiency instance focuses on the IT infrastructure surrounding and the housing building. Both aspects are not within the scope of this thesis as discussed in Section 8.2.4.

- **Factors and parameters** There are several approaches, partially under on-going research, dealing with energy efficiency improvement. They range from frequency adaption and system shutdown to cooling technologies [27] (*EG-2.16:1*). Most approaches are closely (and negatively) correlated to performance: energy efficiency improvements often induce a (time) delay and performance decrease [363] (*EG-2.16:2*), and performance improvements mostly decrease energy efficiency (*EG-2.16:3*).

(1) From the plurality of energy efficiency improvement approaches, two groups are exemplified. The first group deals with the adaption of microprocessor frequency to reduce "idle and dynamic power consumption of modern processors" [37, p. 4]. Group representatives are *Dynamic Voltage and Frequency Scheduling* (DVFS) [371] and Intel's SpeedStep technology [308]. Ge et al. [162] present a DVFS based scheduler that decreases CPU frequency to "dramatically reduce the CPU's power consumption" [162, p. 19]. The second group deals with partial system shutdown due to idling components. A group representative is PowerNap [277]. *(2)* Both groups induce the alluded time delay, as they either slow down the system or require a (short) component boot process. *(3)* Improving single-thread performance by applying speculative execution simultaneously decreases energy efficiency: power spent on following a speculative execution path is lost whenever the path is not taken [27]. *EG-2.16*

- **Role of load** The close correlation between energy efficiency and load is reflected especially by power consumption: the higher the load, the higher the power consumption normally is. This correlation is even manifested in some power consumption prediction models, e.g., the *Telecommunications Equipment Energy Efficiency Rating*

(TEEER) [383] defines the power consumption dependent on a component's load. Thus, on-going research calls for a detailed load consideration of all IT infrastructure component types in the context of energy efficiency [27, 162]. Schöne et al. substantiate this statement by saying that “new hardware generations introduce more and more energy efficiency features, resulting in a power consumption variation by at least a factor of four between idle and full load” [354, p. 481].

2.4.2 Performance

In its most basic form, performance describes achieved work per time. Both, work and time, are highly situation dependent, resulting in a lack of a fundamental definition of performance that would extend the just provided one. Instead, there is a plurality of ways to express IT infrastructure performance (→Section 6.1.3), e.g., distinguishing theoretical peak-performance and average performance over long-periods [212].

- **Instances** Subsequently, five wide-spread performance instances are considered: *Million Instructions per Second* (MIP/s), *Floating Point Operations per Second* (FLOP/s), throughput, *Time to Service Discovery* (TTSD), and *Time to Completion* (TTC) [178, 207]. Some instances are widely applicable, like throughput and TTC, others are focused on a specific field or provisioning paradigm (cf. Section 2.2), like FLOP/s and TTSD. The five examples are ascending ordered according to their level of abstraction and coverage. MIP/s, for instance, is a very low-level CPU performance attribute, whereas TTC is capable of describing the execution of workload on an entire IT infrastructure. The five discussed instances are only a small selection of possible ways to use the *work per time* understanding of performance. For further reading, please refer to Kruse [242], Koziolek et al. [239], or Hennessy et al. [184].

  MIP/s As the name implies, the instance fundamentally bases upon CPU instructions (cf. Section 2.2.1), counting how many instructions per second can be executed. According to Hennessy et al. [184], there are some drawbacks (cf. [242]). The most important one might be the potentially inverse correlation between an application's TTC and the MIP/s value: although a CPU might have a higher MIP/s value and execute more instructions per second, the application might run longer, because the exhibited instructions differ. For instance, a task that requires n_{Inst} instructions on CPU_A might require only $\frac{n_{Inst}}{2}$

on CPU_B. However, there are situations when MIP/s is a helpful performance instance, e.g., for comparing two CPU versions during chip development or for run-time DVFS techniques [196, 195].

FLOP/s Compared to MIP/s, the performance instance *FLOP/s* focuses on the executed floating point operations per second and hence, abstracts from CPU instructions. This performance instance is of special interest for scientific applications (→Section 3.2.1), as they mostly (exclusively) consist of floating point operations [242]. The today's default benchmark for FLOP/s comparison is the LINPACK benchmark (cf. Section 2.3.2).[4] Obviously, applications that do not employ floating point operations need a different performance understanding.

Throughput Can be defined as the "number of requests executed per time unit" [240, p. 233]. Since the request's type can be set individually, throughput is employed in manifold areas. For instance, in the networks area, throughput is the rate of successful message or payload delivery in a predefined time frame; in the storage area, it is the reading and writing speed in *bits/s*; batch systems use a *jobs/sec* notion [239]. Yet, throughput is a super set of MIP/s and FLOP/s.

TTSD Grids rely on the coupling of distributed resources (cf. Section 2.2.3) that provide a set of services. Using a service requires its detection, called *service discovery*, and binding. Since the time to discovery tends to strongly impact the overall execution time, TTSD is an important performance instance for Grids (→Section 3.2.2). Amongst others, the TTSD is influenced by service lifetime, information staleness, and caching [79].

TTC Can be defined as the "time between sending a request and the ending of the response by the server" [240, p. 233], producing the synonym *response time*. As TTC is mostly perceived by the user, it is of great relevance from a consumer perspective (cf. Section 2.3). For instance, web systems and enterprise applications are expected to provide "acceptable" response times to users [278] (cf. [240, 92]). When considering the discovery of a service as request, TTSD can be seen as a sub class of TTC. Figure 2.10 illustrates the term and arranges it in the context of time. The distinctiveness of TTC is that the generic performance notion of *work per time step* is inverted, and the work is atomically described by the time.

[4]At the International Conference of Supercomputing in 2014 (ISC'14), the potentially accompanying HPCG metric was presented: `http://www.hpcwire.com/2014/06/26/development-pushes-ahead-new-hpc-benchmark/`.

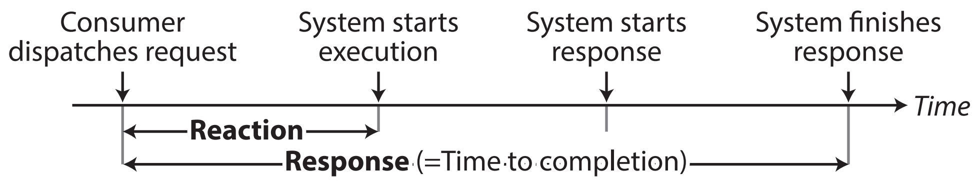


Figure 2.10: Response time and time to completion (based on [239, Fig. 2]).

- **Factors and parameters** Although the concrete influencing hardware components and source code blocks depend on the specific situation, it can be stated that performance is generally influenced by a wide range of aspects, as (empirically) evidenced several times (*EG-2.17:1*). This covers both the set of IT infrastructure component types (cf. Section 2.2) and the executed software, or in other words, the complete hardware and software stack. For instance, the performance of an HPC application is influenced from a software side by data structures, problem decomposition, employed algorithms, their implementations, used compiler(s), communication patterns, and the underlying operating system, from a hardware side from the system's architecture, the processors' architecture and speed, memory hierarchy, and interconnect technology [84, 30, 226, 192, 228, 227, 125] (→Section 6.1.3). Obviously, the consideration of influencing factors can arbitrarily be focused and refined (→Section 3.4, *EG-2.17:2*), e.g., focusing on the system computational noise, describing the "impact of the operating system on the achievable application performance" [112].

EG-2.17

(1) Kerbyson et al. [226] analyze the impact of decomposition approaches and the induced communication demands on application performance, Hoisie et al. [192] investigate communication patterns and behavior in the context of congestion. Carrington et al. [82] split an HPC application in a sequential part, described by memory traces, and a parallel part, described by communication traces. Carter et al. demonstrated the "complex interplay between the architectural paradigms, interconnect technology, and I/O filesystem" [86, p. 186] using the MADCAP package. *(2)* Performance consideration can go down to the microprocessor's internal storage architecture: a register storage might be faster than a stack storage, since the former can hold variables. This, in turn, might reduce memory traffic and finally speed up the program, as registers are faster than memory [184] (cf. Section 2.2.1).

This situation calls for the introductory alluded black box consideration [242] (→Section 3.4), as it enables providing a performance figure for an entire IT infrastructure, despite the aforementioned complexity and plurality of potential influencing factors. For instance, the Top500 list [128] is able to compare supercomputers by using the *FLOP/s* instance that “abstracts” from the outlined complexity.

The long scientific tradition of performance research, and the variety of influencing factors result in manifold approaches to improve and influence performance (*EG-2.18*). A long but certainly not exhaustive list contains [48, 363, 27]

- conditional compilation using pre-processor directives,
- architecture specific optimizations applied by compilers,
- static and dynamic polymorphism provided by some programming languages,
- compile-time, launch-time and run-time choice of library implementations, cache sizes, branch predictions, speculative execution, processor clock speed, or out-of-order execution.

However, nearly all influence parameters (indirectly) underline the close correlation of energy efficiency and performance alluded above.

The following equation demonstrates the involvement of several parameters in improving the TTC ($time_{exec}$) of an application:

$$time_{exec} = n_{inst} \cdot cpi \cdot clockSpeed$$

In particular, it highlights the involvement of a software element – the amount of instructions (n_{inst}) – , and of an hardware element – the clock speed ($clockSpeed$). Both elements are connected by the fixed value cpi – cycles per instruction – to come up with the TTC value. The TTC can either be improved by generating less instructions during compilation, or by executing more cycles per second by increasing the clock speed. *EG-2.18*

- **Role of load** An understanding of the executed workload, its resource utilization, and caused load is essential for most performance attribute instances [45, 227]. Especially throughput and TTC are usually strongly related to IT infrastructure load. In particular, throughput and load are roughly negatively correlated, TTC and load positively. Figure 2.11 depicts this situation (adapted from [239, Figure 3]).

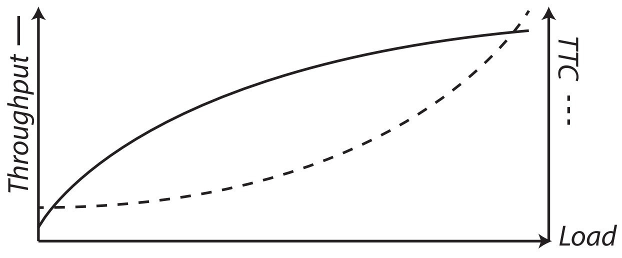


Figure 2.11: Illustrating correlations of load, throughput, and TTC (adapted from [239, Figure 3]).

2.4.3 Reliability

The following discussion about reliability bases upon the groundwork of Avizienis et al. [28] and employs current empirical data provided by Lu et al. [259]. Furthermore, the section focuses on the notion of *availability* and *reliability* that can be arranged in the wider field of resilience. In contrast, the section does not discuss outtake causes, e.g., the distinction of failures according to their causes, their severity, or their impact area. For a detailed and fundamental introduction of failures and errors the reader is referred to Avizienis et al. [28]. Figure 2.12 (taken from [6]) summarizes the considered terms and their correlations, respectively.

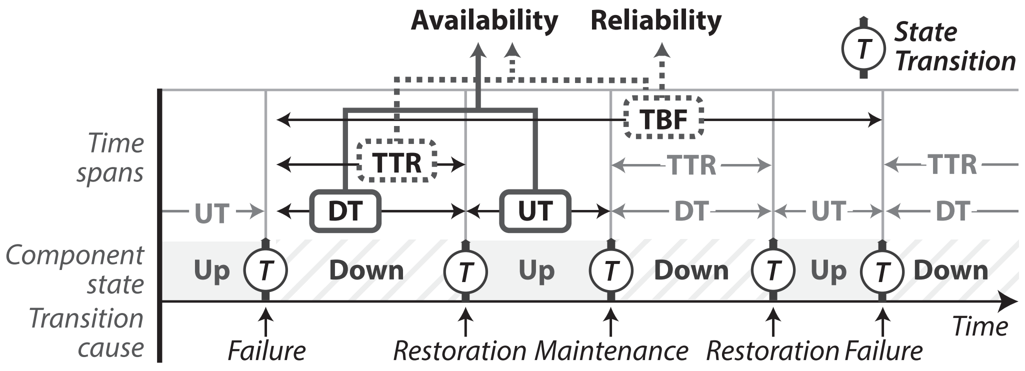


Figure 2.12: Correlation of component states, state transitions, and quantification to derive availability and reliability figures (taken from [6]).

An IT infrastructure component is in the *up state* whenever it delivers a *correct service* or a *system function* as it is described by the functional specification [28]. A *failure* is a temporary or permanent termination of this ability and it causes a transition from correct to incorrect service or to *down state* [28]. A *restoration* (also called *replacement* or *repair*) causes a transition from *down* to *up state* [400, 313]. The time period a component is in the up state or down state is described by the *Uptime (UT)* or *Downtime (DT)*, respectively. The time it takes to conduct a restoration is measured by the *Time to Repair/Restore (TTR)*. The time span between the occurrence of two consecutive failures is described by the *Time between Failures (TBF)* [259].

For all itemized measurements there are mean values labeled with a prefix *M*. They are used to describe or calculate a component's availability and reliability among other capabilities. Especially the *MTBF* plays a central role, because of its strong correlation to failure interpretation, .e.g, it describes the frequency of (hardware) failures [400, 61]. An example of the aforementioned neglected details is the equality of $DT = TTR$ and $UT = TBF$: it depends on the consideration of scheduled maintenance [259, 400], and does not influence the applied calculation methods in general, but only the calculation input values to attain availability and reliability values.

Component *availability* describes the delivery of being in the up state, component *reliability* describes the continuousness of being in the up state under stated conditions over a specified period of time [137, 28].

- **Instances** Based on this basic understanding, there are several specialized availability and reliability numbers (→Section 5.6).

  Steady-state availability Describes the equilibrium behavior of a component and is calculated by $A_s = \frac{MTBF}{MTBF+MTTR}$ [137, 391, 400].

  Point availability Describes the probability of finding a component in the up state at time t [391], and is also called *instantaneous availability*. It is calculated by an integral as described in [49, 391].

  Bath tube curve Describes a component's failure rate, i.e., the inverse of the component's availability, as a function of the component's lifetime, i.e., its age. Figure 2.13 depicts a typical bath tube: On the left hand side, there are several so-called "infant" failures, succeeded by a long production phase and a very low failure rate, which is closed by a phase of "wear-out" failures.

- **Factors and parameters** As the bath tube curve emphasizes, component life time strongly influences availability and reliability. Besides, they are indirectly related to performance, since the excessive heat generation of contemporary components, especially in an HPC cluster, impacts both [330, 363]. Arrhenius equation quantifies this correlation, stating that "failure rate of a compute node in a supercomputer doubles with every 10°C rise in temperature" [363, p. 2]. There are basically two improvement directions:

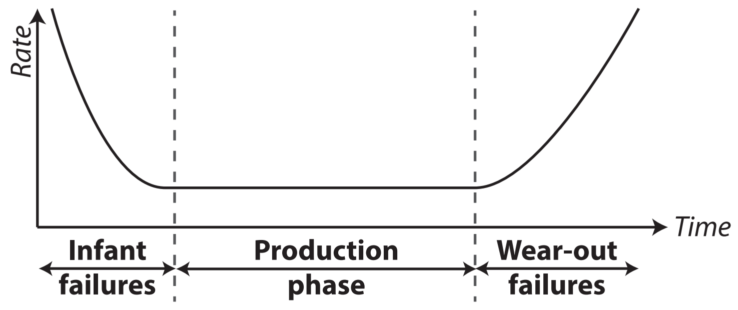


Figure 2.13: Exemplary bath tube curve describing the expected failure rate of a component depending on its lifetime (taken from [137, Fig. 4]).

Redundancy Is one of the major improvement tools [134], as it "allows a function to be performed on more than one node" [322]: in case a failure causes a transition to non availability of a particular component, the system is able to continue operation using a redundant component instead [215]. A common way to describe IT infrastructure redundancy are boolean functions that state "component c_i is available" dependent on the contained sub components and their availability, respectively [137] (*EG-2.19*). An alternative is the *fault tree*, a graphical representation of a redundancy structure. Its leaves represent availability (boolean variables), and inner nodes contain functions to calculate the overall system availability by processing the leave values up to the root.

EG-2.19 The boolean function $\varphi = (c_1 \vee c_2) \wedge c_3$ expresses that a system consisting of two redundant compute nodes c_1, c_2 and a network c_3 is available if the network and at least one of the two compute nodes is available [137].

Checkpointing Aims at recovering from application failure [216] by "writing a set of files required to restart the application after an interrupt" [108, p. 304]. The set of files, collectively called a *checkpoint*, can be selected and processed on a variety of abstraction levels, e.g., on user level or system level [134].

- **Role of load** Two central correlations between reliability and load (cf. Section 2.3.3) are the average component lifetime, and the appearance of errors [165]. The higher the average load, the lower is the component's expected lifetime and the higher the expected error rate, respectively.

2.4.4 Morphological field for IT infrastructure attributes

While considering IT infrastructure attributes, the previous sections extracted a set of dimensions and values for the pursued morphological field. Table 2.4 collects the extraction results and extends the morphological field of Section 2.3, indicated by the gray and black font, respectively.

Dimension	**Value**		
Perspective	Provider	Consumer	
Scale	Low	High	
Heterogeneity	Low	High	
Federation	No	Yes	
Dynamics	Low	High	
Distribution	Regional	National	International
Administrative Units	1	n	
Application Type	Parallel	Distributed	
Attribute	Performance	Energy efficiency	Reliability

Table 2.4: Morphological field of IT Infrastructures summarizing a provider and consumer perspective as well as IT infrastructure attributes.

2.5 Morphological field for the research environment

Section 2.1.2 motivates the assembly of a morphological field, an analytic method for capturing a (complex) domain [356], to address the extend of the research *Environment* and particularly to guide requirements engineering in Chapter 3 and to scope the research in Chapters 4 to 6. The previous sections extract a set of dimensions and values, and iteratively assemble them to the targeted morphological field, which is depicted in Table 2.5.

The itemization below provides a (non formal) definition for each dimension pursuing two guidelines:

- Do not target a general and long-term definition, but purely ensure a common understanding for the presented process model and the thesis. The situation dependent nature of most contained dimensions renders this guideline very important, because a narrowed and casuistic consideration would be a severe problem for formulating generic definitions, but is valid for the context of the thesis.
- Assume for all dimensions exposing an ordinal or metric scale (↗KB p. 267) that approaches capable of tackling the high(est) value are automatically also capable of tackling lower values. For instance, it is assumed that an approach that covers high heterogeneity also covers low heterogeneity (cf. Section 3.2).

The morphological field's dimensions are ordered according to the chapter's structure, starting with the general applied perspective, followed by IT infrastructure (provider) and workload (consumer) related dimensions, and finalized by the IT infrastructure attribute related dimensions.

Dimension	**Value**		
Perspective	Provider	Consumer	
Scale	Low	High	
Heterogeneity	Low	High	
Federation	No	Yes	
Dynamics	Low	High	
Distribution	Regional	National	International
Administrative Units	1	n	
Application Type	Parallel	Distributed	
Attribute	Performance	Energy efficiency	Reliability

Table 2.5: Morphological field describing the *Environment* of the presented research.

- **Perspective** Particular point of view to consider an IT infrastructure. Perspectives differ in their objectives: consumers are mainly interested in IT infrastructure performance and especially short TTC, providers are also interested in a cost-saving and regulations compliant operation.

- **Scale** The size of an IT infrastructure on a quantity basis, spanning all component types. The dimension is relative and closely coupled to the applied level of granularity. For instance, a Grid exhibits a higher scale than a single HPC cluster, but within an HPC cluster, there might be up to thousands of cores (cf. Section 2.2.2). However, the dimension does not aim at comparing IT infrastructures, but at classifying situations for requirements analysis.

- **Heterogeneity** A counter describing the amount of differing IT infrastructure component types. Like *Scale*, the dimension is relative and situation dependent, e.g., in terms of *UNIX systems* a Grid tends to be homogeneous, in terms of CPUs and architectures a Grid tends to be highly heterogeneous. However, just as the *Scale* dimension, the dimension aims at classifying situations and not at comparing IT infrastructures in general.

- **Federation** A binary criterion describing whether IT infrastructure components are provided by several resource providers or only by one.

- **Dynamics** The number of changes for a period of time in the IT infrastructure resource landscape in terms of adding and removing resources and configuration modifications. Although the definition of both, the *change* and the *period of time*, are situation dependent, the dimension arranges the considered IT infrastructure types (cf. Section 2.2). For instance, it assigns a supercomputer that is built for a couple of years a low value, compared to a Grid's high value due to the involvement of several autonomous administration units.

- **Distribution** A classification of an IT infrastructure's geographical distribution. Compared to the other dimensions, the distribution does not depend on a particular situation, since the classification in regional, national, and international bases upon universal borders.

- **Administrative Units** A counter describing the number of administrative units that are involved in IT infrastructure management, operations, and maintenance. There are only two values *1* and *n* to emphasize the differences between a single HPC cluster or supercomputer (cf. Section 2.2.2) and a Grid (cf. Section 2.2.3).

- **Application Type** A distinction of application types being executed on the IT infrastructure.
- **Attribute** An itemization of the three most prevalent quantitative attributes to describe an IT infrastructure.

Chapter 3

Relevance cycle – Requirements specification

This chapter illustrates the research's *Relevance Cycle* execution (cf. Section 1.4), resulting in a *Requirements Specification* (RS) for model integration and reasoning about quantitative IT infrastructure attributes. The employed RS development methodology inherits from requirements engineering methods used in software engineering, because of the their elaborated and broadly accepted nature, as the Knowledge Base motivates on page 264.

Amongst others, result reproducibility requires the explicit description of the applied methodology. Hence, Section 3.1 starts the chapter with introducing and explaining the applied RS engineering methodology that is implemented by the remaining sections: Section 3.2 describes two real world scenarios that collectively cover relevant dimension value combinations of the morphological field extracted from the research *Environment* (cf. Section 1.4) in Chapter 2. Based on scenario descriptions, Sections 3.3 and 3.4 identify a set of functional and non-functional requirements, respectively, and Section 3.5 summarizes both in a so-called evaluation tool for use during the *Design Cycle* execution in Chapters 4 to 6.

3.1 Requirements engineering methodology

A successful and systematic development process starts with *requirements* [342, 233] (↗KB p. 264), because they serve as foundation for project planning [12, 99] and quality assessment [397]. *Requirements engineering* is the process of requirement compilation, analysis, and documentation, based upon studied user needs [200]. Requirements engineering results in a *Requirements Speci-*

fication (RS), a “specification for a particular [...] product [...] that performs certain functions in a specific environment” [374, p. 3]. The RS defines what the completed product is expected to do, but it does not cover the production process comprising cost, delivery status, or reporting procedures.

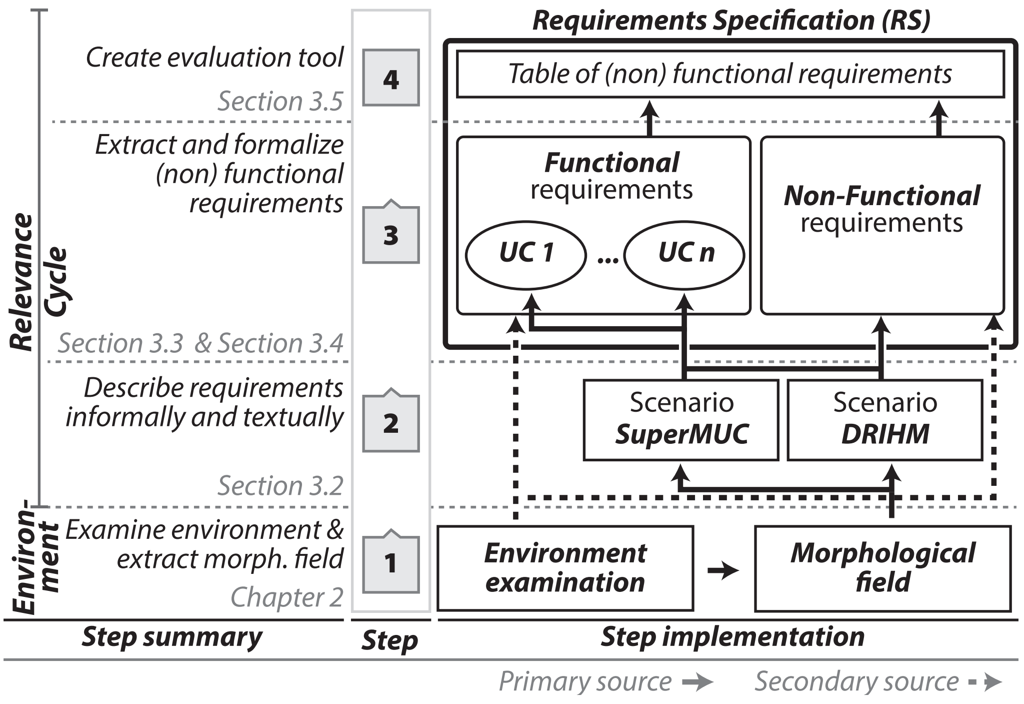
The discipline of software engineering has been developing requirements engineering methods since decades. As a result, these methods are time-tested, mature, and elaborated, what recommends them as basis for a methodology to derive a RS regarding model integration and reasoning about quantitative IT infrastructure attributes. The research’s objective of achieving a process model but a software product (cf. Chapter 4) calls for an adaption that keeps generic aspects but omits software specific ones, like user interface responsiveness.

The applied RS development methodology bases upon the *Use Case analysis* introduced by Jacobson et al. in 1992 [205], and contains adaptations from Hennicker [185] and Rupp et al. [344, 342]. The methodology is chosen, since it answers all questions that are important to the identification of requirements and the successive evaluation [344], e.g., “Which stakeholders are involved in the reasoning process?”, “What is the main functionality?”, and “How are they related to the stakeholders?”.

Figure 3.1 depicts the methodology that results from the alluded adaptions and that underlies the described *Relevance Cycle* execution. In addition, Figure 3.1 highlights the methodology’s four steps bottom-up:

- **Step 1** Starts the RS development with a thorough examination of the research *Environment*, and with an extraction of a morphological field, an analytic method to capture a (complex) domain and consider all possible characteristics [356] for scoping requirements analysis and research activities. In particular, *Step 1* confines the aspects that should be covered by the system, which aspects are part of the system’s surrounding, and which aspects are in relation to the planned system [342, 343]. This consideration is discretized in a morphological field (cf. Section 2.5) to assure a preferably high coverage of characteristics, as carried out in Chapter 2.

- **Step 2** Describes a set of real world scenarios within the research *Environment* to substantiate the targeted RS. Using multiple scenarios aims at covering a preferably wide spectrum. Nevertheless, an *overarching* consideration is hard to achieve and even harder to prove. One step towards a complete scenario set and thus, requirement completeness, is scenario selection based on the morphological field provided by *Step 1*. In particular, scenario selection aims at addressing preferably all



Requirements Specification (RS)
Create evaluation tool
Section 3.5
4
Table of (non) functional requirements
Extract and formalize (non) functional requirements
3
Functional requirements
UC 1
...
UC n
Non-Functional requirements
Relevance Cycle
Section 3.3 & Section 3.4
Describe requirements informally and textually
2
Scenario SuperMUC
Scenario DRIHM
Section 3.2
Environ-ment
Examine environment & extract morph. field
1
Environment examination
Morphological field
Chapter 2
Step summary
Step
Step implementation
Primary source
Secondary source

Figure 3.1: Applied methodology for RS development.

relevant dimension value combinations of the morphological field. In case a dimension exposes an ordinal or metric scale (↗KB p. 267) it is assumed that scenarios formulating requirements about the high(est) value implicitly also formulate requirements about all lower values. For instance, considering a scenario exposing several administrative domains assumes that derived requirements implicitly cover also one administrative domain. This scenario selection procedure identifies two real-world scenarios:

1. The world-class supercomputer SuperMUC from a provider perspective. The *SuperMUC* labeled scenario is considered in Section 3.2.1.
2. The *Distributed Research Infrastructure for Hydro-Meteorology* (DRIHM) that enables European-wide chained execution of *Hydro Meteorology* (HM) simulations, from a consumer perspective. The *DRIHM* labeled scenario is considered in Section 3.2.2.

Step 2 describes for both scenarios the contained IT infrastructure, applications, and reasoning aspects in an informal and textual way. In preparation of the abstracting derivation of (non) functional requirements in the successive *Step 3*, a small black circle and a number, like ❶, flag relevant scenario details to ease referencing.

- **Step 3** Formalizes the non-formal scenario descriptions in a requirement set, in particular *functional requirements* or Use Cases in Section 3.3, and *non-functional* requirements in Section 3.4. Use Cases are extracted and abstracted in four steps [185] from real world scenarios to employ a structured, systematic, and goal-oriented process, and to ensure that the derived Use Cases correspond to practice's demands:

  1. determine involved Actors,
  2. determine Use Cases,
  3. create Use Case Diagrams, and
  4. describe and refine Use Cases.

  The Knowledge Base explains all employed artifacts, in particular the Use Case template, *Unified Modeling Language* (UML) Use Case diagrams, and used terms like "System", "Actor", and "Use Case" on page 264. The software engineering discipline usually conducts a distinguished step to extract functional requirements from Use Cases. Nevertheless, according to the research's objective to develop a process model but a software product, this additional step would not result in an improvement of requirements. Together with the usually high costs of functional requirements extraction from Use Cases [233], this recommends to disregard the additional step, and "Use Case" and "functional requirement" are considered synonymously. The term functional requirement is used from now on as topic to underpin the counterpart role to non-functional requirements.

- **Step 4** Summarizes the identified and formalized (non) functional requirements in a table to ease evaluating developed artifacts in the *Rigor Cycle* in Section 7.3, and to ease analyzing related work in Section 7.4.

Step 2, *Step 3*, and *Step 4* are collectively considered as the *Relevance Cycle* execution (cf. Section 1.4), since they extract and aggregate the goals, tasks, problems and opportunities that are (implicitly) contained in the research *Environment*. The extraction and aggregation process based on scenario consideration is the primary source for (non) functional requirements, *Environment* examination (cf. Chapter 2) is the secondary source, labeled by solid and dashed arrows in Figure 3.1, respectively.

The *IEEE Recommended Practice for Software Requirements Specifications* [374] defines criteria, a requirements specification, its structure, and contained elements should meet. The following list alphabetically summarizes the criteria and highlights why the explained methodology is compliant:

- **Complete** Contain all significant requirements, e.g., functionality or external interfaces, and define the responses to all classes of situations.
  → Requirements are collected from real-world scenarios, which in turn cover relevant value combinations of the morphological field.

- **Correct** Ensure that every requirement is one the final system shall meet.
  → Requirements are extracted from demands identified in real-world scenarios.

- **Modifiable** Structure and style the RS in a way that makes applying changes to the requirements ease, complete, and consistent.
  → The applied templates and diagram types clearly delimit a particular requirement and support its replacement and adjustment.

- **Traceable** State every requirement's origin clearly.
  → The numbered black circles ❶ in the scenario descriptions clearly identify requirement origins.

- **Unambiguous** Ensure for every requirement exactly one interpretation.
  → The applied templates and diagram types clearly describe and formalize a particular requirement.

- **Verifiable** Provide some finite cost-effective process for every requirement with which a person or machine can check that the resulting product meets the requirement.
  → The evaluation tool is a list of check boxes evaluation has to process.

3.2 Scenario descriptions

The section details *Step 2* of the RS development methodology (cf. Figure 3.1) and describes real-world scenarios that act as foundation for (non) functional requirement identification. Scenario selection aims at covering related characteristics in the morphological field. Coverage of the power set of the morphological field's dimensions is not possible, especially because some combinations simply do not exist in reality. Instead, the most common combinations are targeted, resulting in two scenarios. The first one describes the world-class supercomputer *SuperMUC*, the second one covers an IT infrastructure that enables the European-wide execution of *Hydro Meteorology* (HM) simulation chains in the *Distributed Research Infrastructure for Hydro-Meteorology* (DRIHM) project. Throughout scenario descriptions, correlations to morphological field dimensions are indicated by $\xrightarrow{MF}$*dimension*. For each scenario, the following outline is applied:

General overview Overviews the scenario context and arranges it in the morphological field.

Details Describes selected aspects of the scenario in three groups:
- IT infrastructure, consisting of hardware, operations, and planning,
- Applications, and
- Attributes.

Reasoning Discusses reasoning about quantitative IT infrastructure attributes in the scenario.

3.2.1 World-class supercomputer

The scenario examines the *SuperMUC* supercomputer from a provider $\xrightarrow{MF}$*Perspective* and focuses on the $\xrightarrow{MF}$*Attributes* performance and energy efficiency, as justified in following scenario's attribute consideration. Table 3.1 arranges the scenario in the morphological field and highlights the realized dimension values, respectively.

Dimension	**Value**		
Perspective	**Provider**	Consumer	
Scale	Low	**High**	
Heterogeneity	**Low**	High	
Federation	**No**	Yes	
Dynamics	**Low**	High	
Distribution	**Regional**	National	International
Administrative Units	**1**	n	
Application Type	**Parallel**	Distributed	
Attribute	**Performance**	**Energy efficiency**	Reliability

Table 3.1: Arranging the *SuperMUC* scenario in the morphological field.

General overview

The SuperMUC is an award-winning [309] 3 PetaFlop/s (cf. Section 2.4.2) HPC supercomputer (cf. Section 2.2.2) operated by the *Leibniz Supercomputing Center* (LRZ) in Munich, exposing a regional geographic $\xrightarrow{MF}$*Distribution*. SuperMUC is one of the largest HPC systems in Europe, and since its inauguration in mid 2012, it resides in the top 10 of the Top500 list [128], and has been and is employed in manifold scientific projects.[1] For instance, for three dimensional numerical simulations to analyze magnetic fields during the formation of the first stars in the Universe, or for structural modeling of the influenza virus A's matrix protein M1 and its oligomerization. Amongst others, LRZ is a Tier-0 center of the *Partnership for Advanced Computing in Europe* (PRACE) ❶, an HPC infrastructure for researchers and industrial institutions throughout Europe [311]. Besides, LRZ is part of the *Gauss Center for Supercomputing*, which "boosts computational science and engineering by offering a world-class computing and networking infrastructure" [161].

IT infrastructure

The subsequent paragraphs highlight SuperMUC's *hardware*, *operations* aspects and *planning*.

- **Hardware** Figure 3.2 (adapted from [381]) schematically depicts SuperMUC's architecture. In particular, it emphasizes SuperMUC's low resource $\xrightarrow{MF}$*Heterogeneity*, and illustrates its constitution according to the common elements of an HPC system (cf. Section 2.2.2) ❷:

  Compute elements Are built of 18 identical *IBM System x iDataPlex* thin node islands. An island comprises 512 nodes, each employing two *Sandy Bridge-EP Intel Xeon E5-2680 8C* processors having 8 cores each. Summing up with the fat node island, this results in a high $\xrightarrow{MF}$*Scale* of 155.656 cores ❸.

  Storage elements Are split in three areas according to their intent ❹, respectively. The temporary disk storage for compute job execution runs IBM's *General Parallel File System* (GPFS), a high-performance clustered file system. The permanent storage is located on a *Network Attached Storage* (NAS) based disk storage.

  Dedicated network Is arranged in a switched fat tree topology, and also split in different areas and employs different technologies. Islands

[1]For a current list see `www.lrz.de/projekte/hlrb-projects/`, last visited 25.08.2014.

and their nodes as well as the temporary disk storage are connected by an *Infiniband interconnect* (cf. Section 2.2.2), which is operated at a *Fourteen Data Rate* (FDR)-10. SuperMUC's size requires the employment of several switches, in particular 20 big island switches and several smaller switches within an island. The archive and backup system is connected via a slower 10 Gb Ethernet.

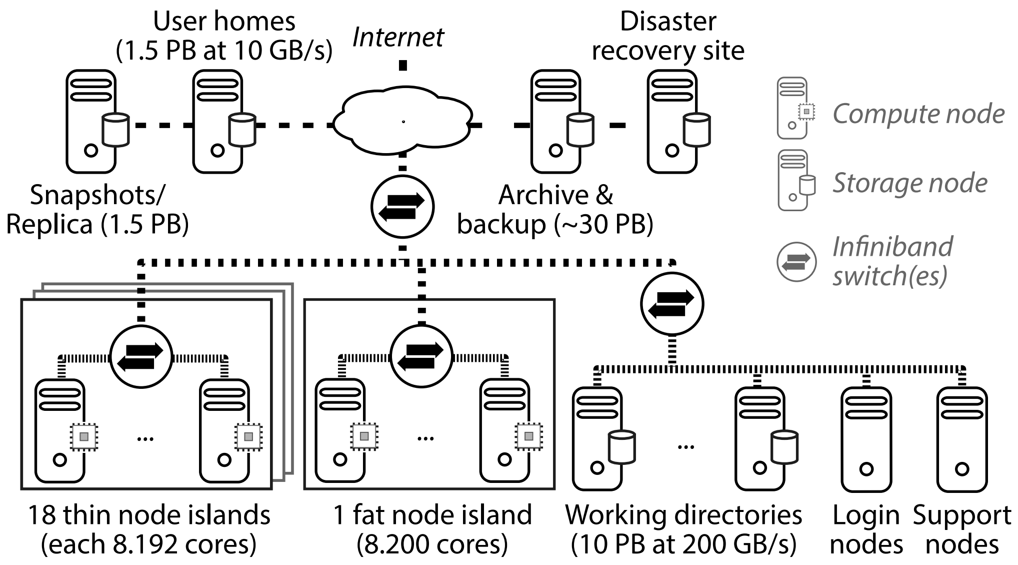


Figure 3.2: Schematic view of SuperMUC's architecture (adapted from [381]).

Due to their maturity and effectiveness, some parts of the outlined SuperMUC's architecture are equal to the architecture of other supercomputers. For instance, the Columbia supercomputer, formally located at the *National Aeronautics and Space Administration* (NASA), also uses two communication fabrics: an "Infiniband switch provides low-latency MPI communication, and a 10 Gb Ethernet switch provides user access and I/O communications" [50, p. 2]. This underpins the central role of communication components ❺, and recommends the achievable communication performance being one of the main objectives of overall system design [32].

- **Operations** The LRZ exclusively provides and operates the SuperMUC without any $\xrightarrow{MF}$*Federation.* Figure 3.3 condenses the LRZ's organizational structure for discussion below and to emphasize the involvement of multiple stakeholders in the reasoning about quantitative IT infrastructure attributes, caused by the involvement of several groups in SuperMUC's operation. Besides the *Board of directors*, four groups of the department *High performance systems*, labeled by bold rectangles in Figure 3.3, interactively operate and maintain the SuperMUC and its surrounding ❻. Each group acts in a delimited range on its own authority, resulting in several $\xrightarrow{MF}$*Administrative Units* that are responsible for SuperMUC's provisioning:

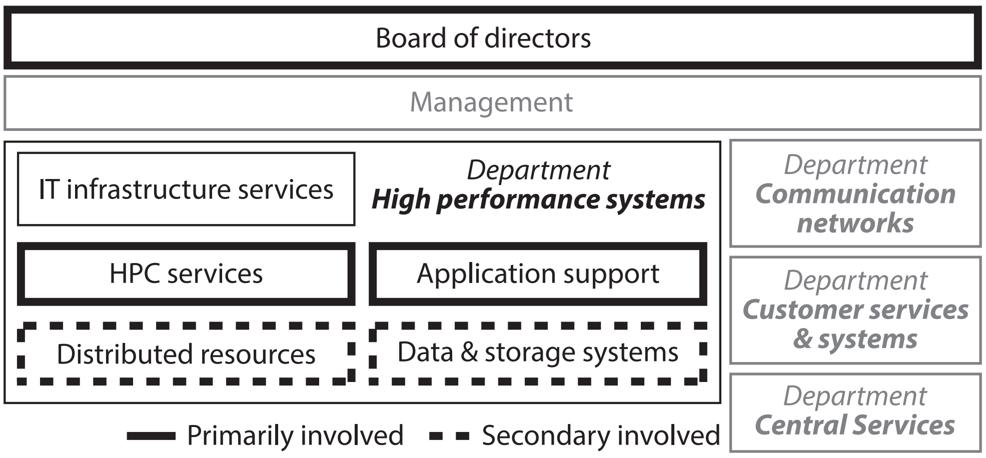


Figure 3.3: Condensed overview of the LRZ's organizational structure, emphasizing the involvement of several groups in SuperMUC's operations.

HPC services Is concerned with operational aspects of SuperMUC's compute nodes and Infiniband interconnect. Besides, the group is responsible for low-level software, like programming environments (MPI, OpenMP, etc.), compiler, performance tools, and libraries.

Application support Covers consumer related aspects ❼, especially the execution and code-wise improvement of (scientific) applications, software-scaling issues, and consumer support in general.

Data & storage systems Responsible for the attached storage elements, i.e., the temporary disk storage, archive, and backup facilities.

Distributed resources Develops interfaces that are exposed to external infrastructures, like PRACE or EGI (cf. Section 2.2.3).

Board of directors Is responsible for the SuperMUC in general ❽.

- **Planning** Components of an extreme scale IT infrastructure like the SuperMUC are highly specialized, balanced, and harmonized to achieve maximum performance. In contrast, flexibility is neglected and restricted to (broken) hardware replacement. Hence, SuperMUC has a planning-horizon of several years, as the lack of flexibility bans any radical hardware changes ❾. In contrast, only small changes in the resource landscape can be accomplished, resulting in low $\xrightarrow{MF}$*Dynamics*.

Applications

The alluded high specialization and extreme scale of a supercomputer apply not only on its hardware, but also on the supported applications. The *Tianhe-2*, for instance, a supercomputer developed by China's National University of Defense Technology, is said to be "at the world's frontier in terms of calculation capacity, [...] but lacks software support" [94]. Even though this specialization enables world-class performance for a specific application category, other categories are comparatively slow or cannot be executed at all.

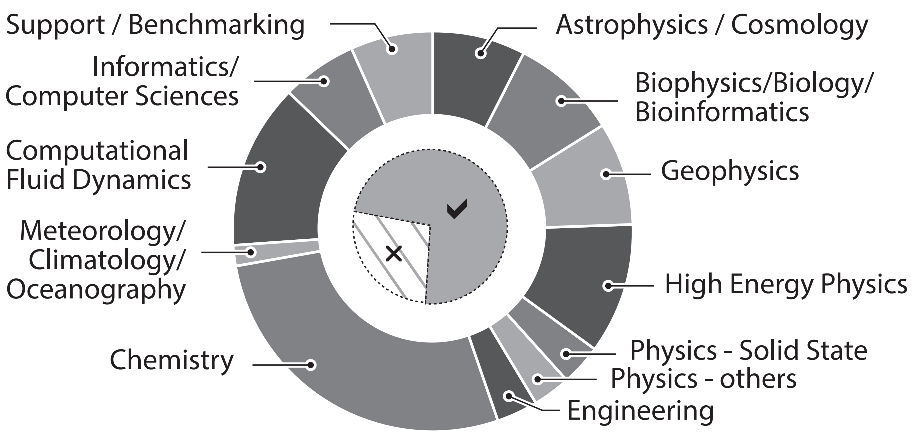


Figure 3.4: Distribution of scientific disciplines that used SuperMUC in 2013, collected from SuperMUC's accounting database.

The SuperMUC is designed as general purpose HPC system that allows the execution of manifold application types to avoid such an extreme focus. As a consequence, more than 150 different applications run on SuperMUC per year, ranging from medical and engineering to energy and astrophysics applications ❿. Figure 3.4 overviews the distribution of executed jobs per scientific discipline for the year 2013.[2] Obviously, Figure 3.4 provides only a snapshot for the considered time range, and the particular portions might differ compared to other years. Nevertheless, Figure 3.4 and its discussion below give a time-agnostic impression of executed applications and especially of the variety of code.

[2]Data were collected from SuperMUC's accounting database.

Software from each discipline differs in many aspects and causes different types of workload ⓫. Astrophysics, for instance, often employs the *Free Fast Fourier Transform* (FFTW) library [153] for computing discrete Fourier transformations, Chemistry employs the *Basic Linear Algebra Subprograms* (BLAS) library [51] for basic linear algebra operations. Further examples are the communication behavior and employed general-purpose libraries, like MPI, or the employed programming languages, like C, C++, and multiple FORTRAN versions ⓬.

The execution time of applications averages 5,51 hours, and ranges from a couple of seconds to 80 hours, taken by an application of CFD, in particular for the numerical investigation of complex multiphase flows with Lagrangian particle methods ⓭. 73.1% of jobs were completed, 26.9% were canceled ⓮, labeled by a checkmark and a cross in Figure 3.4, respectively.

Most of the disciplines using the SuperMUC tend to *apply* computer science instead of dealing with it directly ⓯. The implied differing (technical) user backgrounds and skills sometimes result in disproportional attribute demands and urgently call for supporting application development and execution, as done by LRZ's *Application support* group.

Attributes

The subsequent explanation justifies, why *performance* and *energy efficiency* (cf. Section 2.4) are of primary interest for SuperMUC's operation ⓰:

- **Performance** The compute-intense nature of applications executed on SuperMUC, the comparison with other high-class supercomputers in the Top500 list [128], and the participation in the performance-focused PRACE, emphasize performance as primary attribute, especially the two following performance instances ⓱ (cf. Section 2.4.2):

  FLOP/s The nearly exclusive execution of floating point operation based [242] scientific applications (cf. Figure 3.4) on SuperMUC renders *FLOP/s*, a counter of executable floating point operations per second, a suitable and important performance instance. In addition, *FLOP/s* figures are used for public proposal applications in general, and for determining SuperMUC's peak performance and its Top500 list position in particular, an important *Key Performance Indicator* (KPI) for scientific supercomputers.

  TTC The execution of big, complex, and demanding (scientific) applications foreground *TTC* especially for SuperMUC's users, as they are interested in preferably short waiting times. In the context of

SuperMUC's performance, the user's request that starts the TTC time span (cf. Section 2.4.2) is the *job submission.* Consequently, TTC can consider queuing time of the employed IBM LoadLeveler batch system or exclude it (18). TTC values are widespread, ranging from minutes to several hours, as mentioned in the section above about applications executed on SuperMUC.

Both performance instances are combined and influenced by multiple component types, even within a single domain, e.g., system performance includes computing cores as well as communication, interconnect, and I/O performance [91, 191, 281] (19).

- **Energy efficiency** Energy efficiency is of paramount importance for supercomputers having SuperMUC's size, and it has become a serious concern to HPC data centers [71], because energy costs tend to outclass (hardware) procurement costs [34] and "in many situations power consumption is becoming the determining factor of the system size" [71, p. 135]. Furthermore, steadily increasing electricity prices evolved to be a severe problem, especially in the context of upcoming Exascale systems [8] and expected consumption levels of hundreds of megawatts in the future [211, 190] (cf. [1]). In particular, for sustainable multi-Petascale and Exascale systems, energy efficiency is mandatory [417], not only since the Exascale Computing study [43] set a definite power limit of 20 *Megawatt* (MW) as affordable. Finally, social pressure for "Green IT" and reducing the emission of CO_2 (cf. Section 2.4.1), underpinned by a study of the *German Federal Ministry for the Environment, Nature Conservation and Nuclear Safety* (BMU) [135], call for energy efficiency. Besides those generally applicable reasons, energy efficiency is one of SuperMUC's distinctiveness, as it is claimed to be IBM's first supercomputer using warm-water cooling.

  Despite the attribute's central role, there are no concrete attribute instances focusing solely on SuperMUC (20). Instead, the PUE (cf. Section 2.4.1) is an important factor, as it also covers the special cooling system of SuperMUC. However, to improve energy efficiency of HPC systems, the fine-grained assessment of the power consumption of the entire HPC system encompassing compute nodes, interconnect networks, and storage devices is required [27] (21).

Reasoning about quantitative IT infrastructure attributes

Within the last years, modern data centers like the LRZ emerged from research facilities to flexible business entities. This shift is an answer to the diversity of changes in their environments, e.g., cooperation with and participation in projects and infrastructures like PRACE (cf. Section 2.4), arising governance duties, or continuously changing hardware and electricity prices. This situation poses several demands on quantitative IT infrastructure attributes of a data center's IT infrastructure, like the LRZ's SuperMUC 22, from several perspectives:

- **Consumer perspective** Attribute demands are mainly formulated by the relationship between the LRZ and scientists, nowadays treated like customers. The relationship's parameters are defined in *Service Level Agreements* (SLA), describing provisioning guidelines about manifold IT infrastructure attributes like performance or reliability 23. Depending on the particular project's or scientist's needs, SLAs heavily differ, e.g., due to different emphasis. For instance, PRACE aspires to provide the "apex of HPC technology" [311] to Europe's researchers, which prioritizes performance and related aspects 24. Since most SLAs are formulated in an individual way, reasoning about quantitative IT infrastructure attributes is mostly interested in *descriptive statistics* 25, e.g., to identify correlations or commonalities.

- **Provider perspective** LRZ's management and provisioning have to face consumer agnostic factors in terms of *operations* and *procurement*:

  Operations Covers the day-by-day use of SuperMUC and related attributes, e.g., run SuperMUC at 50% during the night to save energy or to address differing electricity prices between night and day 26. As motivated above, one of the most important attributes to operations is energy efficiency 27. Neglecting or even ignoring these factors can cause severe issues, e.g., high energy consumption might produce "unaffordable" electricity prices. The potential collision of interests from a consumer and provider perspective might result in conflicting attribute objectives 28. This, in turn, recommends *optimization algorithms* 29 for reasoning about quantitative IT infrastructure attributes, because LRZ's management aims at finding a (local) optimal trade-off between all parties, or in other words, it aims at "quantify[ing] impacts prior to implementation" [31, p. 43]. To support application developers, also *What-if* analysis 30 is helpful for LRZ's

Application support group to investigate planned impacts of hardware modifications or configuration changes to the IT infrastructure and applications [84].

Procurement Covers aspects related to the procurement of new IT infrastructure components, e.g., in the context of SuperMUC's extension phase in 2015, which targets doubling SuperMUC's performance to a peak performance of up to 6.4 PetaFlop/s. The long-term planning horizon related to SuperMUC's hardware calls for an eminently thorough analysis of IT infrastructure attributes, because the very low flexibility of supercomputers bans every radical modification for improvement or error correction during operation. Reasoning in the procurement 31 context wishes for *What-if* analysis capabilities 30 to compare competing systems and solutions, and to enable investigation of varying configurations, like the trade-off between a higher performance against increased power consumption. *What-if* analysis is especially useful to system designers, as it quantifies the benefits between alternatives [228], and it "help[s] computing centers select the best system in an acquisition" [30, p. 194]. Reasoning targets a "careful balance between the availability of existing components and the need for technological advancement" [32, p. 2].

- **Both perspectives** SuperMUC provides functionality through a complex qualitative and quantitative component interplay to deliver sophisticated and non-trivial computation and storage functionality, as other contemporary HPC systems do. This close interplay hardens identifying the specific contribution of a single component to SuperMUC's functionality [139, 268] 32, which is of special importance for both perspectives during SLA negotiation, because a SLA focuses on the attribute value, and particularly not on value compilation. In addition, effects induced by local modifications on a single component can quickly and easily cascade and affect the HPC infrastructure partly or completely in an unpredictable way.

The participation in and contribution to multiple differing projects and the surrounding factors (cf. above) interdict a narrow consideration of only one attribute. Instead, a flexible attribute list appropriate to the specific situation is highly required 33. In addition, correlations between attributes must be covered, because each modification usually induces at least two groups of effects, the (by definition) positive intended effects and the mostly negative but unavoidable side effects. For instance, increasing a typical HPC infrastructure's redundancy to address short-time breakdown and to improve reliability [134], simultaneously increases energy consumption and degrades performance due to redundancy overhead [137]. Subsumed, avoiding (inadvertent) SLA violations requires consideration of all relevant attributes as well as their correlations.

3.2.2 European-wide execution of hydro meteorology simulations

The *Distributed Research Infrastructure for Hydro-Meteorology* (DRIHM) scenario applies the consumer $\xrightarrow{MF}$*Perspective*, and focuses on the $\xrightarrow{MF}$*Attributes* performance and reliability, to address the complexity of computational demanding hydro-meteorological models and the negative effects of IT infrastructure breakdown during model execution. The attribute energy efficiency is neglected, since it is only relevant from a provider's perspective. Table 3.2 arranges the DRIHM scenario in the morphological field and highlights the realized dimension values, respectively.

Dimension	**Value**		
Perspective	Provider	**Consumer**	
Scale	Low	**High**	
Heterogeneity	Low	**High**	
Federation	No	**Yes**	
Dynamics	Low	**High**	
Distribution	Regional	National	**International**
Administrative Units	1	**n**	
Application Type	Parallel	**Distributed**	
Attribute	**Performance**	Energy efficiency	**Reliability**

Table 3.2: Arranging the *DRIHM* scenario in the morphological field.

General overview

During the last years, IT-based research activities in several disciplines evolved from a stand-alone and separated principle of operation to a chained and collaborative one, e.g., to investigate physical phenomena [345] (cf. Section 2.3.1). This is also true in the *Hydro Meteorological Research* (HMR) discipline: the proprietary software components that analyzed (also proprietary) data sets separately on systems owned and maintained by the scientist or its home institution shall nowadays be combined to chains and workflows. In addition, the workflows shall be executed on IT infrastructures having a European-wide or even global geographic $\xrightarrow{MF}$*Distribution*.

The DRIHM project [98] addresses this upcoming demand in the realm of HMR by aiming at the implementation and operation of a distributed and inter-organizational IT infrastructure that supports the execution of complex HMR workflows. One of the project's professed objectives is the use of existing large-scale and powerful resources and other IT infrastructures.

Figure 3.5 summarizes the above outlined transition for the DRIHM project: on the left hand side, the initial situation is characterized by HMR scientists fulfilling several duties simultaneously, being an end user, a model developer, and an administrator. In addition, each tool is executed separately on proprietary resources without any data exchange. On the right hand side, these duties and related topics are distributed and assigned to multiple institutions. Furthermore, the tools are executed on a federated IT infrastructure according to a chain that is (graphically) defined in a portal.

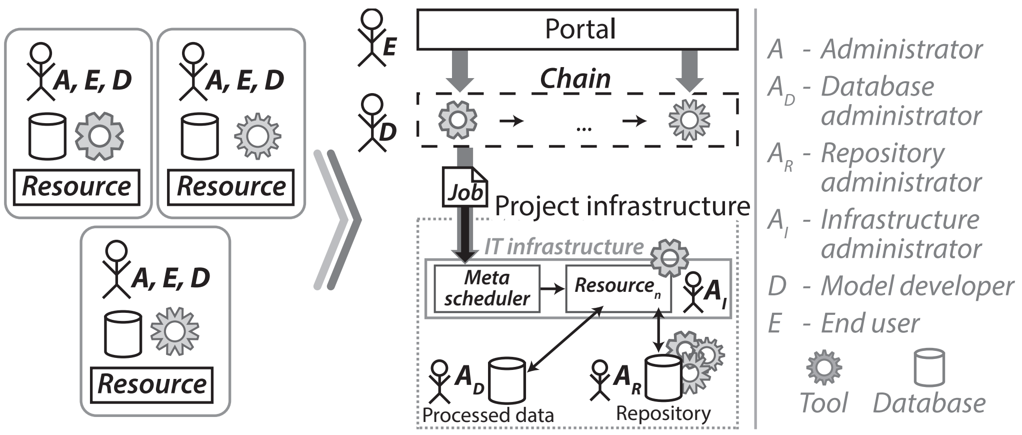


Figure 3.5: Transition from a stand-alone and separated science paradigm to a chained and collaborative one in the HMR discipline.

IT Infrastructure

The subsequent paragraphs highlight the DRIHM IT infrastructure's *hardware*, *operations* aspects and *planning* of modifications.

- **Hardware** Figure 3.6 (taken from [107]) schematically depicts the DRIHM IT infrastructure and emphasizes its high resource $\xrightarrow{MF}$*Heterogeneity*, a direct consequence of the introduced objective of using existing resources. On its left hand side, Figure 3.6 itemizes several Grid resources, provided by single organizations as well as *National Grid Initiatives* (NGI) (cf. Section 2.2.3). All resources belong to the EGI. On its right hand side, Figure 3.6 collects all non-Grid resources, covering additional resources types, like a set of PRACE machines, Cloud systems, and proprietary components. Based upon $\xrightarrow{MF}$*Federation*, all resources collectively build the DRIHM IT infrastructure.

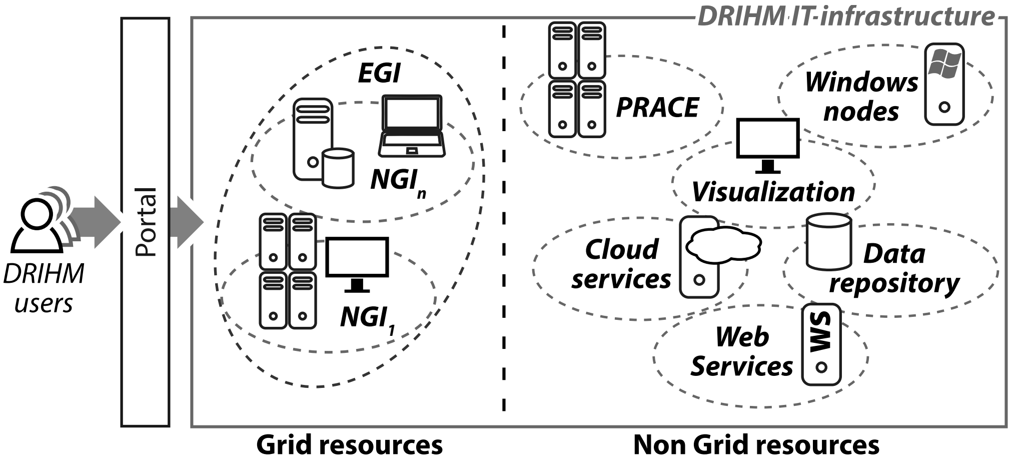


Figure 3.6: Schematic view of the DRIHM IT infrastructure (taken from [107]).

- **Operations** According to the federated and multi institutional paradigm underlying the DRIHM IT infrastructure, there is no central entity being solely responsible for operations and maintenance of resources in the *drihm.eu* VO.[3] Instead, the contributing resource providers from nine countries 34 are responsible only for their own resources. This causes several $\xrightarrow{MF}$*Administrative Units* and high $\xrightarrow{MF}$*Dynamics*, since resource providers apply individual maintenance cycles and policies

[3]https://wiki.egi.eu/wiki/EGI-DRIHM:Infrastructure

35. Information about Grid-based resources within the DRIHM IT infrastructure are stored in a shared BDII 36 service (cf. Section 2.2.3), which is updated on a daily base by the resource providing institutions and exhibits a queryable interface.[4]

- **Planning** Compared to the SuperMUC scenario, there is no detailed long-term planning process, but at most, abstract architectural decisions made at the beginning of the DRIHM project. This has two reasons:
  - The DRIHM IT infrastructure is a typical e-Infrastructure (cf. Section 2.3). Thus, short-term scientist needs drive its assembly and configuration, which prohibits a long-term planning horizon.
  - The Grid-like nature and organizational structure of the DRIHM IT infrastructure result in individual maintenance activities that are rarely synchronized with all resource providers, and the outtake of a particular resource is (theoretically) covered by another resource exposing the same or similar capabilities.

Applications

The set of applications that are executed on the DRIHM IT infrastructure was defined upfront in a first version at the beginning of the DRIHM project [119]. During the project, this application set was slightly extended. Figure 3.7 (adapted from [107]) overviews the considered applications and structures them in four HMR-related groups: *Meteorological*, *Hydrologic*, *Hydraulic*, and *Impact*. Even though HMR-driven nature of the structuring, it also affects the workload to consider during reasoning execution, because each application group differs in terms of generated load and demands on quantitative IT infrastructure attributes 37. In addition, nearly each model is developed and maintained by a different scientist or group of scientists 38. For instance, *WRF-NMM* and *Meso-NH* 39 are highly scaled and compute intense parallel applications (cf. Section 2.3.1) that require powerful HPC systems and cause (heavy) inter-node communication. In contrast, *RainFARM* is a comparatively frugal Python application that runs on a single node while achieving acceptable TTC figures.

Attributes

The consumer perspective of the DRIHM scenario focuses on performance and reliability 40:

[4]Exemplary command to get all CREAM resources:
`lcg-infosites -is bdii.ipb.ac.rs -vo drihm.eu cream`.

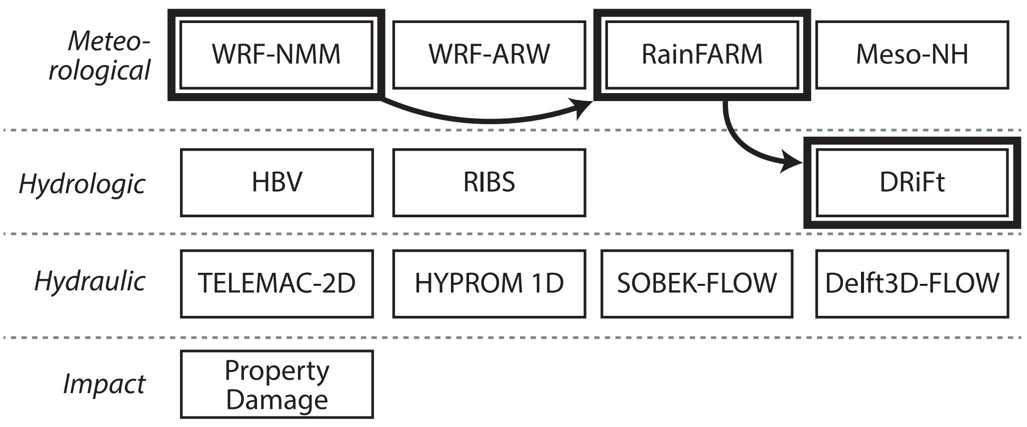


Figure 3.7: DRIHM model structure and a chain (adapted from [107]).

- **Performance** Most HMR models are computationally intensive and require an HPC cluster or even a supercomputer, like SuperMUC. Consequently, performance is an important attribute from a consumer perspective, as it reflects the time a scientist has to wait for computational results. Compared to the SuperMUC scenario, the performance instances in the DRIHM scenario are more general:

  TTC Considers the execution of both, a single model and a model chain (cf. above). As in the SuperMUC scenario, the triggering activity is the job submission.

  TTSD Addresses the (theoretical) employment of multiple services implementing the same HMR model 41.

- **Reliability** The complexity of most HMR models causes long run times of up to several hours or days, as discussed in the SuperMUC scenario (cf. Figure 3.4). Thus, IT infrastructure reliability is of great importance, as the continuousness of being in the up state under stated conditions over a specified period of time [137, 28] (cf. Section 2.4.3) directly impacts the smoothness of an application's execution. For instance, an application running 24 hours also requires an IT infrastructure that is available for 24 hours. Otherwise, application execution is interrupted and in the worst case, (partial) results are lost.

Reasoning about quantitative IT infrastructure attributes

The distributed and geographically scaled nature of the DRIHM IT infrastructure, and the absence of a central (strategic) IT infrastructure management cause a much higher level of abstraction when reasoning about quantitative

IT infrastructure attributes, compared to the SuperMUC scenario (cf. Section 3.2.1). This means that reasoning in the DRIHM scenario deals with contained systems, like the SuperMUC, as black boxes and considers only exposed characteristics, instead of investigating hardware components down to a single core ㊷. In addition, reasoning aims at defining SLAs that guide a more focused reasoning, as carried out in the SuperMUC scenario ㊸.

3.3 Functional requirements

This section discusses functional requirements for the considered system that should provide functionality for reasoning about quantitative IT infrastructure attributes. The section extracts the functional requirements from scenario descriptions, and abstracts and formalizes them in functional requirements or Use Cases[5] within *Step 3* of the RS development process (cf. Figure 3.1). In facing the extend of extraction results, Section 3.3.1 and 3.3.2 overview the gathered eleven actors and 18 Use Cases, respectively. Appendix C provides complete details, consisting of a description, correlations, and abstraction sources, employing the black circle flags ❶ for referencing specific aspects in the scenario descriptions.

3.3.1 Actors

An actor describes a *role* that interacts with the considered system (↗KB p. 264). Noteworthy, an actor can be realized by multiple persons or in other words, one person can realize multiple actors. For referencing in the RS, each actor is labeled by the acronym *ACT* and a numbering, e.g., ACT-7.

Figure 3.8 overviews actors extracted from the scenario descriptions in an (object-oriented) inheritance hierarchy, instead of an organizational hierarchy describing the constitution of a company or business division, to emphasize the abstraction level of each actor, respectively. In addition, Figure 3.8 provides at its bottom the scenario that mainly origins an actor, and it highlights the three actor groups that differ in available knowledge, experience, responsibilities, and objectives:

- **Provider** The group is experienced in and responsible for maintaining the considered IT infrastructure and domain specific problems. It contains the following actors that are mainly abstracted from the SuperMUC scenario according to the applied provider perspective:

[5] As established in Section 3.1, the terms are used and understood synonymously.

ACT-1	Subsumes actors being experienced in and responsible for IT infrastructure operations, maintenance, and provisioning.
ACT-2	Responsible for and realizes all physical operation activities on the IT infrastructure.
ACT-3	Concerned about all non physical aspects in IT infrastructure operations, covering attribute and workload modeling and prediction.
ACT-4	Responsible for architectural and technical long-term decisions regarding the IT infrastructure, and for the evaluation of technological trends and innovations in terms of suitability for the IT infrastructure.
ACT-5	Executes concrete operation and maintenance activities, like hardware replacements or incorporation of (reviewed) modifications.
ACT-6	Highly experienced in modeling and measuring one or multiple quantitative IT infrastructure attributes and their influence factors, especially for the IT infrastructure at hand.
ACT-7	Highly experienced in the development and execution of workload and identifying and predicting expected load, especially for the IT infrastructure at hand.

- **Management** The group contains only one actor that initiates reasoning tasks and defines reasoning activities in the context of strategic decision making. The actor belongs to both scenarios, because he acts as linking element between provider and consumer actors:

ACT-8	Initiates and interprets reasoning activities in the context of his responsibility for the IT infrastructure in general, and SLA negotiation.

- **Consumer** The group executes (scientific) software on the IT infrastructure. It contains the following actors that are mainly abstracted from the DRIHM scenario according to the applied consumer perspective:

ACT-9	Subsumes actors using the IT infrastructure to solve a particular (scientific) problem.
ACT-10	Represents a (scientific) project's head, who negotiates SLAs to define attribute value ranges for the considered IT infrastructure that are sufficient for the project.
ACT-11	Develops workload and especially real world applications that run on the IT infrastructure.

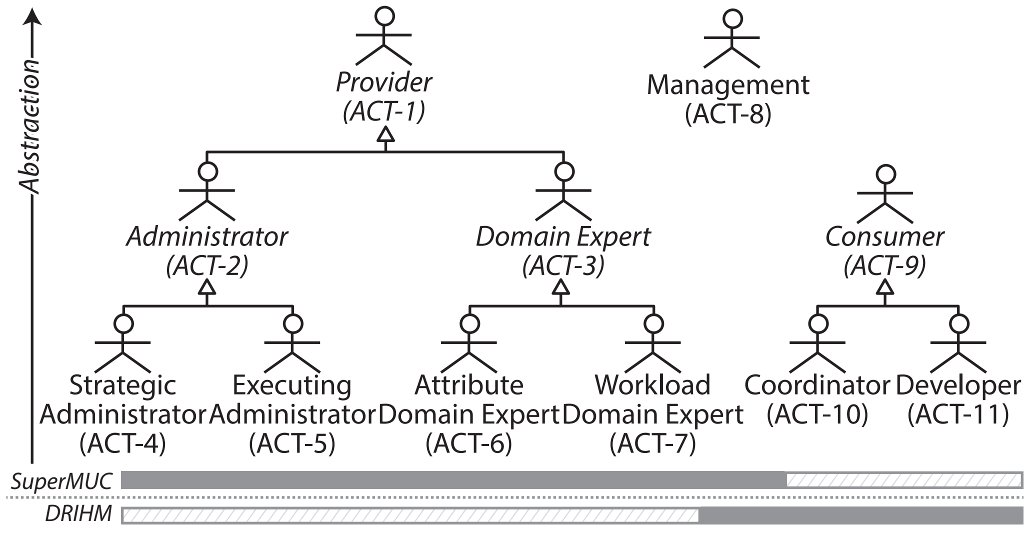


Figure 3.8: Overview of actors extracted from the real-world scenarios.

3.3.2 Use Cases

A Use Case is a description of a system's behavior under various conditions (↗KB p. 264). For referencing in the RS, each Use Case is labeled by the acronym *UC* and a numbering, e.g., UC-1. The subsequent itemization overviews the Use Cases that were extracted from the scenario descriptions in Section 3.2. The overview is structured in three sub systems grouping related aspects:

- **A – Reasoning objectives** Continuously changing user demands, manifold external factors like electricity prices or national law, and economic purposes assemble a (theoretically) infinite set of potential reasoning aspects and combinations. Achieving a cost-saving and effective reasoning in this context requires a clear-sighted initiation and justification for a reasoning activity, and call for the identification, extraction, and specification of reasoning objectives from the theoretical infinite set of options and directions. Figure 3.9 overviews six Use Cases and related actors addressing this situation:

UC-1	Initiate reasoning activity, e.g., to address a special customer need or external factor changes.
▹ UC-1.1	Negotiate SLA and attributes.
UC-2	Define reasoning objectives and formalize them for reasoning tool creation in sub system *B* and reasoning execution guidance in sub system *C*.

▹ UC-2.1 Define attribute(s) to consider.
▹ UC-2.2 Select workload to consider and (potentially) use for load generation.
▹ UC-2.3 Select IT infrastructure component(s) to consider.

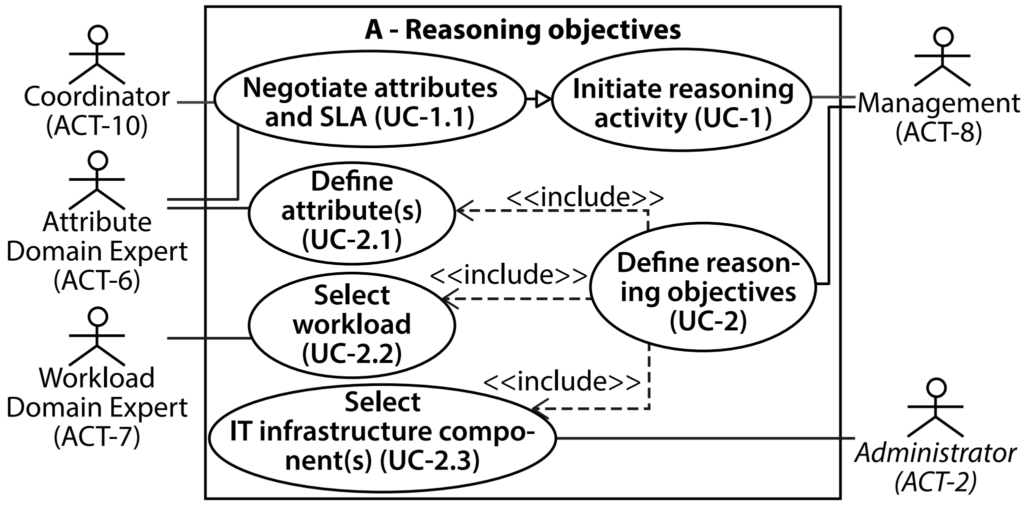

A - Reasoning objectives
Coordinator (ACT-10)
Attribute Domain Expert (ACT-6)
Workload Domain Expert (ACT-7)
Negotiate attributes and SLA (UC-1.1)
Initiate reasoning activity (UC-1)
Management (ACT-8)
Define attribute(s) (UC-2.1)
<<include>>
Define reasoning objectives (UC-2)
Select workload (UC-2.2)
<<include>>
<<include>>
Select IT infrastructure component(s) (UC-2.3)
Administrator (ACT-2)

Figure 3.9: Use Cases of sub system "A – Reasoning objectives".

- **B – Reasoning tools** Three reasons require an elaborated model of the IT infrastructure and its attributes to execute reasoning compliant to the reasoning objectives defined in sub system *A*:

  IT infrastructure unavailable The steady use of IT infrastructures and their central role prohibit physically performing pursued reasoning methods, like *What-if* analysis, on the considered IT infrastructure, since it is mostly in production mode [78, 244, 337] (cf. Section 1.2). In the procurement context, considered IT infrastructure components might not be available yet [228].

  Model integration Section 2.4 introduces the plurality of attribute instances, each potentially implemented by one or multiple (mature) models that cover a small aspect of the reasoning objectives. In order to benefit from this situation, a model should be prepared, existing attribute models can be incorporated in.

  Parameter flexibility Reasoning based on an elaborated model facilitates a much higher flexibility in terms of reasoning parameters, e.g., for procurement.

  Figure 3.10 overviews seven Use Cases and related actors addressing this situation:

UC-3	Create a model of the considered IT infrastructure.
▹ UC-3.1	Split IT infrastructure modeling and delegate tasks in case the IT infrastructure is highly scaled or complex.
▹ UC-3.2	Import modeling related data from third-party models.
▹ UC-3.3	Update the IT infrastructure model to reflect changes.
UC-4	Select a model for an attribute and related IT infrastructure component(s).
▹ UC-4.1	Create a model proxy in case no model can be integrated.
▹ UC-4.2	Create a load profile for workload selected in UC-2.2.

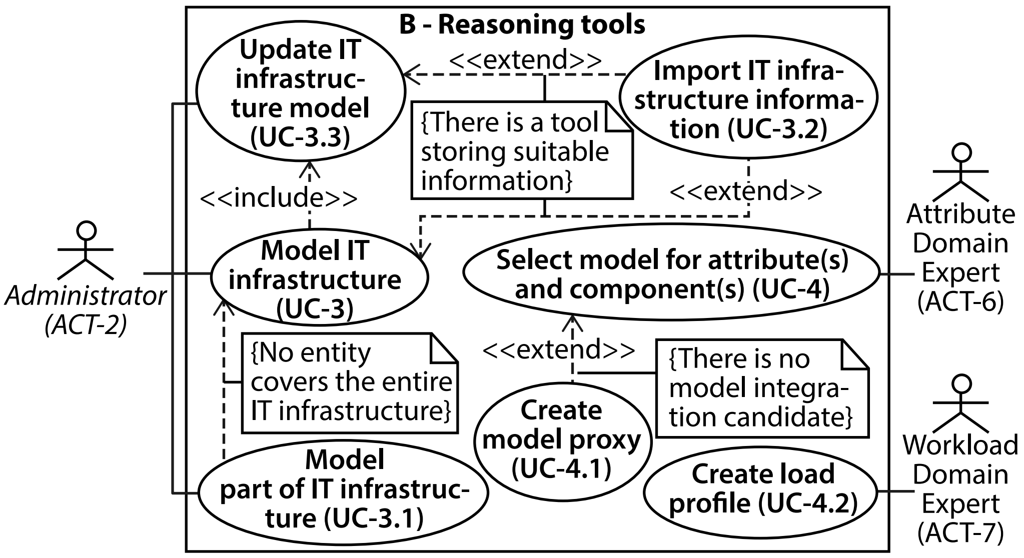

B - Reasoning tools
Update IT infrastructure model (UC-3.3)
<<extend>>
Import IT infrastructure information (UC-3.2)
{There is a tool storing suitable information}
<<include>>
<<extend>>
Attribute Domain Expert (ACT-6)
Model IT infrastructure (UC-3)
Select model for attribute(s) and component(s) (UC-4)
Administrator (ACT-2)
{No entity covers the entire IT infrastructure}
<<extend>>
{There is no model integration candidate}
Create model proxy (UC-4.1)
Workload Domain Expert (ACT-7)
Model part of IT infrastructure (UC-3.1)
Create load profile (UC-4.2)

Figure 3.10: Use cases of the sub system "B – Reasoning tools".

- **C – Reasoning execution** Guided by the reasoning objectives and using the reasoning tools defined and prepared in sub system *A* and *B*, respectively, the reasoning investigates possible outcomes for a variety of input parameters. The potential variety of reasoning goals require the application of different (overlapping) reasoning approaches, e.g., quantifying alternatives or finding a (local) optimum for a predefined parameter set. Depending on the reasoning results, an activity trigger might be required, e.g., conducting a modification to improve the mode of operation. Figure 3.11 overviews five Use Cases and related actors addressing this situation:

UC-5	Execute reasoning based on defined objectives and using the prepared tools.

▹ UC-5.1	Execute What-if analysis based reasoning.
▹ UC-5.2	Execute optimization based reasoning.
▹ UC-5.3	Execute descriptive statistics based reasoning.
UC-6	Trigger a specific activity if required by the reasoning results.

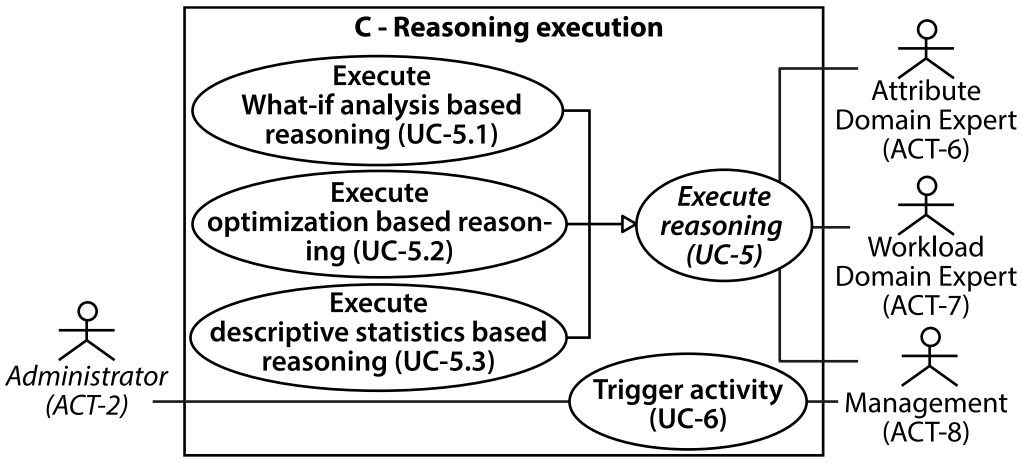

C - Reasoning execution
Execute What-if analysis based reasoning (UC-5.1)
Execute optimization based reasoning (UC-5.2)
Execute descriptive statistics based reasoning (UC-5.3)
Execute reasoning (UC-5)
Trigger activity (UC-6)
Attribute Domain Expert (ACT-6)
Workload Domain Expert (ACT-7)
Management (ACT-8)
Administrator (ACT-2)

Figure 3.11: Use Cases of the sub system "C – Reasoning execution".

3.4 Non-functional requirements

This section discusses non-functional requirements extracted from scenario descriptions within *Step 3* of the RS development process (cf. Figure 3.1). In facing the extend of extraction results, the section overviews the gathered eight non-functional requirements. Appendix C provides complete details, consisting of a description, correlations, and abstraction sources, employing the black circle flags ❶ for referencing specific aspects in the scenario descriptions.

In contrast to functional requirements that describe a concrete, delimited, and implementable behavior or (partial) function [342] (cf. Section 3.3), non-functional requirements cover the system's overall characteristics, its use, and quality criteria [233]. In other words, non-functional requirements are those requirements that are not functional. A typical non-functional requirement in the software engineering discipline is the responsiveness of user interfaces or robustness. Non-functional requirements on the methodology for reasoning about quantitative IT infrastructure attributes cover all aspects that impact several elements, functional requirements, and areas. For instance, covering all types of IT infrastructure attributes is of high importance for attribute and workload selection in Use Case UC-2.1 and UC-2.2, but also for IT infrastructure modeling in Use Case UC-3.

For referencing in the RS, each non-functional requirement is labeled by the acronym *NFR* and a numbering, e.g., NFR-7. Subsequently, the identified non-functional requirements are itemized.

NFR-1	Individual component type sets
NFR-2	Individual attribute sets
NFR-3	Multiple granularity levels
NFR-4	Workload consideration
NFR-5	Job cancellation
NFR-6	Development over time
NFR-7	Simplicity
NFR-8	Efficient use

3.5 Evaluation tool

This section details the final *Step 4* of the RS development process (cf. Figure 3.1) that collects all extracted (non) functional requirements in an evaluation tool to ease and support evaluation of research results (Section 7.3) and related work (Section 7.4) in the *Rigor Cycle*. The evaluation tool consists of two tables for the functional and non-functional requirements, respectively. A table entry consists of four elements:

- **Fulfillment indicator** Uses a □, ✓, or × to illustrate a not yet analyzed entry, a fulfilled entry, or a not fulfilled entry, respectively.
- **Identifier** Contains the requirement's identifier for referencing.
- **Title** Extends the identifier with the requirement's name to ease employment of the evaluation tool.
- **Justification** (Optionally) explains why the particular requirement is (not) fulfilled.

□	UC-1	**Initiate reasoning activity**
□	UC-1.1	**Negotiate SLA and attributes**

continued on next page

continued from previous page

☐	UC-2	**Define reasoning objectives**
☐	UC-2.1	**Define attribute(s)**
☐	UC-2.2	**Select workload**
☐	UC-2.3	**Select IT infrastructure component(s)**
☐	UC-3	**Model IT infrastructure**
☐	UC-3.1	**Model part of IT infrastructure**
☐	UC-3.2	**Import IT infrastructure information**
☐	UC-3.3	**Update IT infrastructure model**
☐	UC-4	**Select model for attribute and component(s)**
☐	UC-4.1	**Create model proxy**
☐	UC-4.2	**Create load profile**
☐	UC-5	**Execute reasoning**
☐	UC-5.1	**Execute What-if analysis based reasoning**
☐	UC-5.2	**Execute optimization based reasoning**
☐	UC-5.3	**Execute descriptive statistics based reasoning**
☐	UC-6	**Trigger activity**

Table 3.3: Validation tool – Functional requirements.

☐	NFR-1	**Individual component type sets**
☐	NFR-2	**Individual attribute sets**
☐	NFR-3	**Multiple granularity levels**
☐	NFR-4	**Workload consideration**
☐	NFR-5	**Job cancellation**
☐	NFR-6	**Development over time**
☐	NFR-7	**Simplicity**
☐	NFR-8	**Efficient use**

Table 3.4: Validation tool – Non-functional requirements.

Chapter 4

Design cycle – Process model fundamentals

The chapter introduces the developed process model for the integrated reasoning about quantitative IT infrastructure attributes and prepares its extensive detailing in the successive Chapters 5 and 6. Section 4.1 motivates the development of a process model, Sections 4.2 and 4.3 explain the process model's design concepts and implementation approaches, respectively. Section 4.4 overviews the attained process model, whose execution instance is from now on named a *reasoning project*.

4.1 Process model motivation and objectives

The *Requirements Specification* (RS) in Chapter 3 implies that reasoning about quantitative IT infrastructure attributes can and should be model-based. Three reasons disqualify a *one-size-fits-all* approach, an approach that commonly tries to cover a problem space in its entirety:

- **Extend of problem space** Reasoning deals with IT infrastructures from a provider and consumer perspective, and with quantitative IT infrastructure attributes. Considered IT infrastructures are "very diverse architecturally" [192, p. 1], as they differ in the underlying design philosophy, the deployed hardware, and applied configurations (cf. Section 2.2 and 2.3). Quantitative IT infrastructure attributes expose a plethora of (calculation) formulas, as the (small) set of abstract attribute definitions can be realized in manifold ways (cf. Section 2.4). This high diversity and scale produce a tremendous problem space, a one-size-fits-all approach cannot manage [255].

- **Individual objectives and parameters** Reasoning intents tend to pursue individual set of objectives, constraints, and trade-offs [226, 122] (*EG-4.1:1*), e.g., "make the data collection as accurate as possible, and the predictions as conservative as reasonable" [112, p. 9]. Besides, reasoning target values "cannot be static and may change over time" [76, p. 5], and the respective application domain might require specific parameter sets reasoning has to process and respect, e.g., differing load values dependent on the executed workload [226] (cf. Section 2.3.3). A one-size-fits-all approach would inevitably omit details that theoretically might be insignificant or uninteresting for most reasoning intents, but of high importance to a specific one.

- **Accuracy vs. portability** The trade-off states that it is challenging and rarely possible to achieve high accuracy and portability at the same time (*EG-4.1:2*): an accurate model is mostly highly fitted to a specific situation or system, a portable model is mostly less accurate due to its generic nature. In other words, a one-size-fits-all approach can be either sufficiently accurate or widely applicable.

EG-4.1

(1) When considering performance (cf. Section 2.4.2), computer architects might be interested in improving a new machine's design in terms of FLOP/s, application developers and end users in achieving a preferably short TTC [63]. *(2)* Analyzing the performance of an HPC cluster based on generic communication patterns is *portable* to a variety of HPC clusters. Yet, it is less *accurate* than a cluster specific model that is able to respect the characteristics of employed hardware, like an individual Infiniband setup (cf. Section 2.2.2), which is not the case for a portable model.

The outlined drawbacks of a one-size-fits-all reasoning model motivate the development of a process model that generically prescribes and formalizes *how* to compile an *individual* and casuistic reasoning model, suitable for a specific reasoning intent. The targeted reasoning model is called a *reasoning function*. It is a mathematical mapping of a parameter set on a vector of quantitative IT infrastructure attribute values. Equation 4.1 depicts the reasoning function f in its most generic form, its building blocks are described subsequently:

- **Domain** Parameters used for attribute vector calculation, and relevant for the specific reasoning intent. Although used in the same way in a mathematical sense, the domain is split in two sub sets to emphasize differences in terms of interference, focus, and origin:

Modification parameters Describe aspects being subject to change in the context of a reasoning intent or a planned modification. Modification parameters $mod_i \in Mod$ implicitly describe IT infrastructure internals from a provider perspective (cf. Section 2.2), especially contained hardware.

Configuration parameters Describe IT infrastructure externals, particularly from a consumer perspective (cf. Section 2.3), like application scaling behavior or the electricity price during the day. Configuration parameters $conf_j \in Conf$ influence the reasoning outcome, but are not subject to change.

The introduced semantic domain decomposition is advantageous for reasoning efficiency and result interpretation, because both sets can be handled separately (*EG-4.2*): modification parameters are examined in more detail, e.g., in terms of analyzed value ranges, and potentially every modification parameter is covered by a dedicated person or group being responsible for the affected IT infrastructure components or aspects. In contrast, configuration parameters are more or less taken for granted and a small value range constraints the reasoning activity. Besides, decomposition enables consideration of cost to analyze the "true merits of any technology or product"[34, p. 49], as it can be handled in the role of a configuration parameter. At the same time, semantic domain decomposition does not disadvantage reasoning methodologies, like mathematical optimization (↗KB p. 262) or descriptive statistics (↗KB p. 268), because both parameter sets are treated in the same mathematical way. Noteworthy, both sets, modification and configuration parameters, are not compulsory mandatory and can be set as required for the respective reasoning objectives.

- **Co domain** An attribute value vector (cf. Section 2.4), computed by the compiled reasoning function according to the reasoning objectives.

$$f(\underbrace{\underbrace{mod_1, ..., mod_n}_{\text{Modification parameters}}, \underbrace{conf_1, ..., conf_m}_{\text{Configuration parameters}}}_{\text{Domain}}) = \underbrace{\underbrace{\begin{pmatrix} attr_1 \\ ... \\ attr_z \end{pmatrix}}_{\text{Attribute values}}}_{\text{Co domain}} \tag{4.1}$$

EG-4.2

An exemplary reasoning activity deals with optimizing the overall electricity price and performance of an HPC cluster dependent on the number of its worker nodes. The number of nodes can be altered to influence the cluster's performance/energy price trade-off. Thus, it is treated as modification parameter mod_1. Although the electricity price and the notional fixed load value can also be altered in some way, e.g., negotiation with power contractors or user groups, they have a different reach, and are treated as configuration parameters $conf_1$ and $conf_2$, respectively. This decomposition of the reasoning's three-element input parameter set implicitly highlights, which parameters can be adapted and should receive special attention, and which parameters must be taken as given. This, in turn, reduces the required efforts for the pursued optimization, because only the modification parameter is in the optimization's search space (↗KB p. 262), whereas configuration parameters can be fixed values or at most a very small value range.

4.2 Design concepts

The subsequent design concepts underlie the process model and its development, aiming at fulfilling the RS defined in Chapter 3, and at achieving a sustainable and widely applicable solution:

- **Integration of existing artifacts** The high relevance of IT infrastructure attributes (cf. Section 2.4) and the strong impact on day-by-day work of IT infrastructure management and operations produce an abundance of specialized (and mature) models, instrumented components, and gained measurements, covering a variety of partial aspects of a reasoning intent. The process model bases upon integration to benefit from this situation and to enable the use of elaborated, established, and especially validated artifacts. In particular, the process model employs integration by

  - describing the selection of suitable models and their integration in the reasoning function, and
  - using already instrumented components for the creation of model proxy functions, in case no model can be integrated.

  In other words, the process model builds upon (important) work done and results achieved by previous researchers, but also makes a logical progression by combining these results in a new way [212].

- **Iterative function refinement** Out of the variety of attribute influencing factors, reasoning might be only interested in a (small) subset. The process model copes with these individual objectives by beginning reasoning function compilation with a coarse-grained function skeleton, and iteratively refining it until the reasoning function (co) domain comply to the reasoning objectives (*EG-4.3:1*). Besides, several other research transactions use iterative refinement [31, 319, 158] (*EG-4.3:2*).

(1) Focusing on the power consumption of Example *EG-4.2*, a fictive coarse-grained reasoning function is $f(n) = 9.4 \times n + 4$. It calculates the cluster's power consumption by multiplying the node consumption of 9.4 Watt by the node number n and adding a basic consumption of 4 Watt. An iterative refinement would first replace 9.4 by a model $g(l)$, describing the power consumption of one node at a load of l. Second, it would replace the fixed value 4 by a model $h(t)$, describing the basic consumption of the cluster at a point in time t. The final reasoning function would be $f(l, n, t) = g(l) \times n + h(t)$. *(2)* Barker et al. [31] start the modeling of application performance with the coarse-grained function $T_{total} = T_{comp} + T_{comm} \times T_{overlap}$ and iteratively replace function elements by finer-grained formulas. The same procedure is applied by Pfeiffer et al. [319] for $t = t_{comp} + t_{comm} + t_{io}$, and by Gamell et al. [158] for the energy modeling formula $E = E_{system} + E_{comm}$. *EG-4.3*

- **Flexible attribute binding** The plurality of factors influencing a particular attribute (cf. Section 2.4), and the variety of pursued attribute objectives cause manifold specialized attribute instances. The process model addresses this situation by
  - decomposing the notion of a quantitative IT infrastructure attribute in a generic *concept* and one or multiple concept *instances*, and by
  - fostering flexible attribute concept and instance binding to IT infrastructure components on all granularity levels (*EG-4.4*).

The performance instance *FLOP/s* (cf. Section 2.4.2) matches HPC clusters and supercomputers, but not network components. In contrast, energy efficiency (cf. Section 2.4.1) is less constrained and can be considered for a wide field of component types, ranging from a single Intel PXA255 processor [104] to a data center [138]. *EG-4.4*

4.3 Implementation approaches

Guided by the design concepts discussed above, implementation targets a process model that is capable of generating significant statements in a good or even optimal time- and cost scale, or in other words, a process model that achieves a good trade-off between statement quality and result gaining costs. In particular, it aims at a

- flexibly applicable, and easy to use process model,
- cost- and effort-efficient, goal-oriented reasoning function compilation,
- far-ranging support of reproducible and traceable results, and
- reduction of error-proneness by standardizing, formalizing, and automating the reasoning activity.

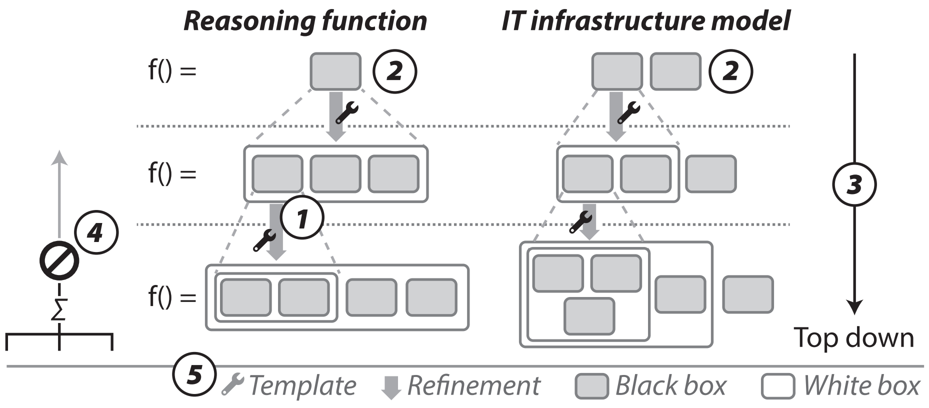


Figure 4.1: Schematic overview and interrelation of the presented process model's implementation approaches.

Figure 4.1 arranges the implementation approaches applied to realize the above listed items and design concepts. The following list describes the figure elements referencing them using the symbol ⓝ:

- **Reasoning objectives driven refinement** ① States that the objectives of a reasoning intent strictly constrain function refinement, which means that a refinement is executed if and only if it
  - adds an additional parameter (*domain*),
  - adds an additional attribute (*co domain*),
  - improves the accuracy of one or multiple co domain values.

  In particular, refinement is not bound to or driven by the physical hardware's shape, as a hardware-driven refinement tends to cause dispensable efforts for the aspired reasoning (*EG-4.5*). Instead, driving function refinement exclusively by reasoning objectives means to "begin with simple models and few parameters, then add complexity only as needed [...]" [84, Sec. 2]. Thus, it results in a focused reasoning function that is as compact as possible and as extensive as necessary, as illustrated in Figure 4.1: the second refinement extends only the outer left function element, because the others wouldn't produce a result that complies to the reasons itemized above. Other research transactions apply this approach, too, e.g., Carrington et al. model parallel application performance by "adding complexity only as needed to explain observed performance" [84, Sec. 2].

Simply stated, a hardware driven approach would refine a cluster node down to the CPU architecture, memory, and bus to comply to the physical hardware, irrespective of the reasoning objectives at hand. This might be meaningful for reasoning interested in a CPU architecture. In contrast, if reasoning deals with application TTC, the interplay of cluster nodes is of relevance. This renders the alluded refinement obsolete, as a consideration of the CPU architecture might be too detailed [217]. *EG-4.5*

- **Black box approach** ② Initially considers every reasoning function element and IT infrastructure component as *black box* that is refined to a *white box* whenever knowledge about the black box internals is required. This approach has two advantages:
    1. A coarse-grained model can be created very quickly on an abstract level and afterwards punctually refined, supporting a fast, effort-efficient, and goal oriented reasoning activity.
    2. System characteristics can be evaluated at the necessary level of detail and accuracy [337] while not extending the entire model, because only relevant parts are punctually refined, and all other parts remain on an abstract level.
- **Top-down refinement** ③ Highlights the refinement direction being top-down with reference to the granularity level of the 1) compiled reasoning function, the 2) covered IT infrastructure components, and the 3) attributes. This direction has two advantages:
    1. The substitution of already existing formulas in the reasoning function implicitly dictates refinement guidelines. For instance, a

refinement that replaces the discrete value 4 in Example EG-4.3 by a model is implicitly guided, because the refining model must apply the same scale, semantics, and role as the replaced discrete value.

2. The refining element can directly be compared to the replaced function element. In case the delta is not within a (predefined) target range, the refinement can be improved or overdone directly.

- **No bottom-up aggregation** ④ Emphasizes that a bottom-up aggregation is explicitly not applied due to potentially severe "by-design" problems. As carried out in [9, 1], a prevalent problem is to preserve semantics during aggregation in terms of scale and meaning (*EG-4.6*). Possible solutions face either the difficulty of formulating a generic aggregation process, or the implicit loss of portability. In contrast, this is not the case for the above detailed iterative top-down refinement using a black box approach, because the alluded aggregation problems are implicitly solved: whenever a part of the reasoning function is refined (replaced), it is implicitly ensured that no semantic issues appear, because otherwise, the refinement would be invalid.

EG-4.6

A generic aggregation formula that processes performance relevant figures bottom-up would have to tackle the difficult comparability of CPU, memory, and interconnect performance (cf. Section 2.4).

- **Template-based formalization** ⑤ Describes, dictates, and formalizes all artifacts and actions of a reasoning project to address its comprehensive and versatile nature, as particularly the compilation of an individual reasoning function covers manifold tasks and areas. Addressing the alluded variety, template-based formalization uses five template types, each specialized for a certain situation. The decision template, for instance, aims at supporting and accelerating a selection process and at documenting the decision about a selection in a way that fosters reproducibility, since a selection has the same consequences independently from the decision maker.

4.4 Overview of the process model

The section pulls the chapter's elements together and overviews the presented process model for the integrated reasoning about quantitative IT infrastructure attributes. Figure 4.2 highlights the process model's composition of *artifacts and procedures*, and a *reasoning methodology*:

- **Artifacts and procedures** Define and formalize notions, models, and methods that are employed by a reasoning project in seven parts:

  Roles and actors Abstract stakeholders to clarify responsibilities and to enable task assignments throughout a reasoning project.

  Template meta model Provides concepts and language elements for the template-based formalization.

  Provenance information model Provides classifiers for storing partial results and tracing reasoning project execution to facilitate reproducibility.

  IT infrastructure notion Describes the process model underlying understanding of IT infrastructures.

  Measuring Formalizes the gathering of measurements.

  IT infrastructure attribute notion Describes the process model underlying understanding of IT infrastructure attributes.

  Application proxy workload selection Formalizes the selection of workload alternatives whenever the original one cannot be executed.

- **Reasoning methodology** Formalizes the reasoning activity as a "comprehensive integral series of techniques and methods creating a general system theory" [199] (cf. [315]), considering a method as a "system of rules and guidelines for a consistent procedure" [315, p. 41]. Figure 4.2 illustrates in its upper part the methodology's three phases:

  Phase A Identifies, collects, and specifies objectives, elements, assumptions, and constraints of the reasoning project and formalizes them in a *reasoning suite* that steers the reasoning project's execution, and aligns the distributed and decoupled task execution. Besides, the phase produces the reasoning function skeleton (cf. Equation 4.1).

  Phase B Conducts the iterative top-down refinement of the reasoning function skeleton as motivated in Section 4.3. Guided by the reasoning suite, the phase iterates as long as elements of the reasoning function's (co) domain remain undefined, or as long as any other requirement stated by the reasoning suite remains unfulfilled.

  Phase C Uses the reasoning function to execute the actual reasoning. It selects an appropriate reasoning tool, analyzes different parameter combinations, and triggers a direct reaction on the reasoning results, if necessary. Compared to *Phase A* and *Phase B*, the description of *Phase C* is less extensive, since most decisions to make and activities to execute in a reasoning project are covered by the first two phases.

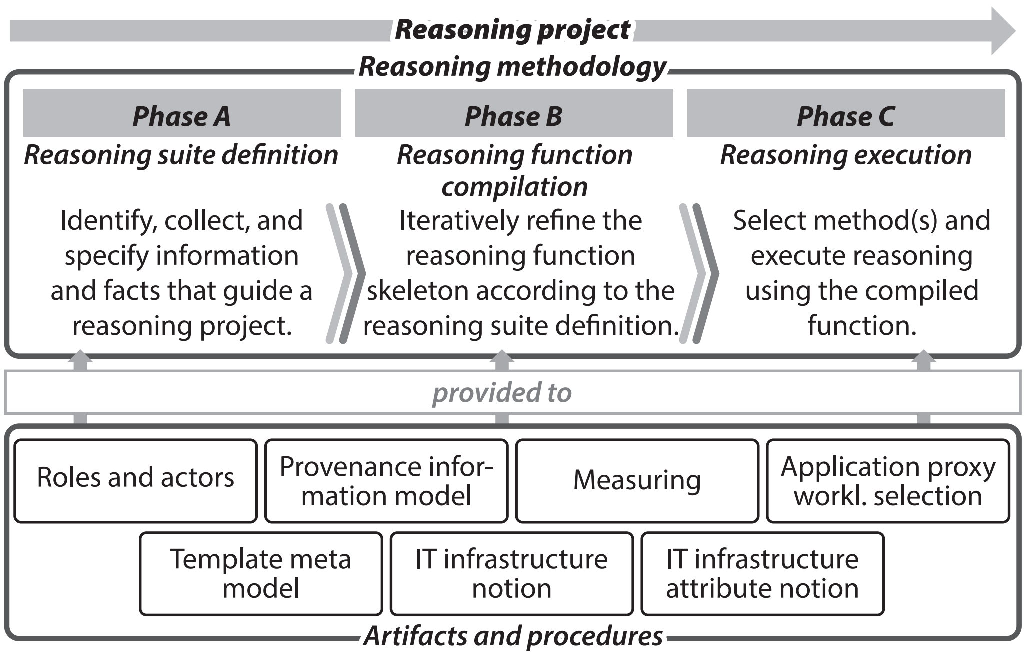

Reasoning project
Reasoning methodology
Phase A
Reasoning suite definition
Identify, collect, and specify information and facts that guide a reasoning project.
Phase B
Reasoning function compilation
Iteratively refine the reasoning function skeleton according to the reasoning suite definition.
Phase C
Reasoning execution
Select method(s) and execute reasoning using the compiled function.
provided to
Roles and actors
Provenance information model
Measuring
Application proxy workl. selection
Template meta model
IT infrastructure notion
IT infrastructure attribute notion
Artifacts and procedures

Figure 4.2: Overview of the presented process model.

The following Chapters 5 and 6 describe the process model's *artifacts and procedures* and *reasoning methodology*, respectively.

Chapter 5

Design cycle – Process model artifacts and procedures

The chapter details the process model's artifacts and procedures that are provided to the reasoning methodology (cf. Section 4.4). Figure 5.1 highlights in its lower part the chapter's focus on the contained seven parts, the chapter details in a cascading order according to their mutual use:

- **Roles and actors** Abstract stakeholders in Section 5.1 to clarify responsibilities and to enable task assignments in a reasoning project.
- **Template meta model** Provides concepts and language elements in Section 5.2 for describing and formalizing process model elements.
- **Provenance information model** Provides classifiers in Section 5.3 for storing partial results and for tracing reasoning project execution in order to foster and ensure reproducibility.
- **IT infrastructure notion** Describes in Section 5.4 the process model underlying understanding of IT infrastructures.
- **Measuring** Formalizes in Section 5.5 the gathering of measurements, built of measuring objectives definition and measuring setup design.
- **IT infrastructure attribute notion** Describes in Section 5.6 the process model underlying understanding of IT infrastructure attributes.
- **Application proxy workload selection** Formalizes in Section 5.7 workload replacement selection if the original one cannot be executed.

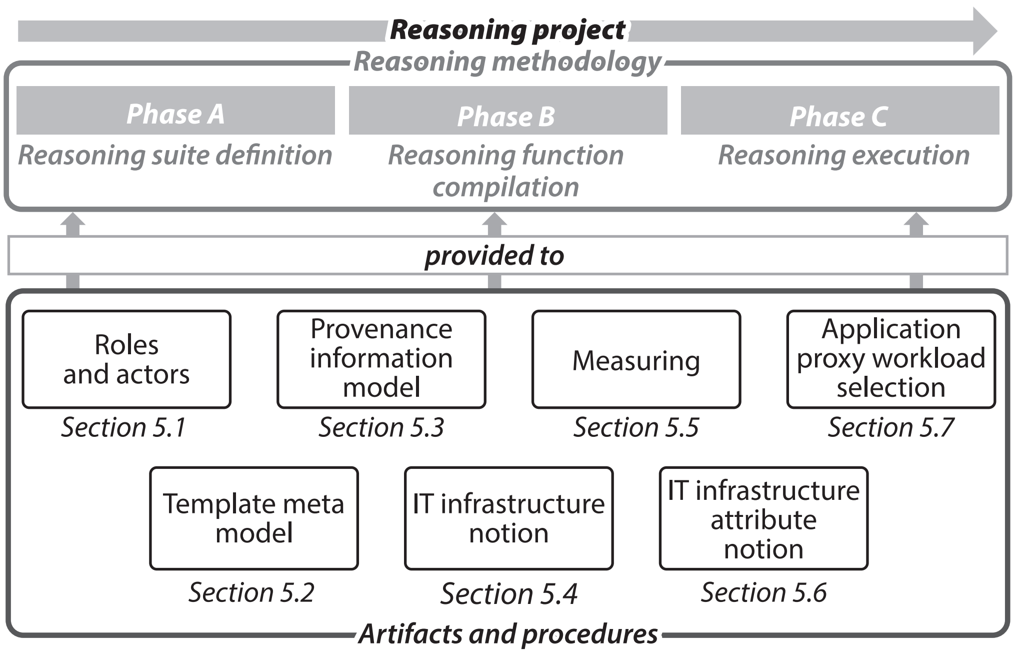
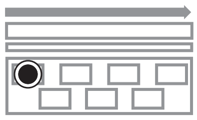


Figure 5.1: Overview of the developed process model emphasizing the focus of Chapter 5 on the *artifacts and procedures* part.



→ *Fig. 5.1 (p. 98)*

5.1 Roles and actors

Assigning tasks and responsibilities in a reasoning project and in particular the reasoning methodology requires a set of roles and actors the duties can be assigned to. The set of actors is not defined from scratch, but taken from the *Requirement Specifications* (RS) that already extracted a set of actors in the research *Relevance Cycle* (cf. Chapter 3). Besides facilitating a cost-saving development, this also complies to NFR-8 that requires an efficient use, because all defined actors are identified as being necessary in real-world scenarios. Noteworthy, only the leafs of the extracted actor hierarchy (cf. Figure 3.8 on page 80) are used, as they were directly extracted from the scenarios. In contrast, the abstraction of commonalities to more generic actors is necessary for the RS, but not for a reasoning project. Elements of the resulting set are named *roles* but actors to avoid confusion with entities of the RS, although the underlying concepts are the same (↗KB p. 264). The roles are detailed below and labeled by a unique identifier, e.g., R1 for *Role 1*:

R1 – Strategic Administrator Is responsible for architectural and technical long-term decisions regarding the IT infrastructure. It evaluates technological trends and innovations in terms of suitability for the IT infrastructure. For small IT infrastructures, it might overlap with role R2.

R2 – Executing Administrator Executes the concrete operation and maintenance of the IT infrastructure, e.g., replacement of broken hardware, incorporation of (reviewed) modifications, maintenance of management databases, or implementation of required software. Compared to role R1, R2 doesn't make any (long-term) decisions about the IT infrastructure's architecture. In contrast, it acts according to externally given decisions and orders. For small IT infrastructures, it might overlap with role R1.

R3 – Attribute Domain Expert Is experienced in one or more attributes, e.g., performance or energy efficiency (cf. Section 2.4), especially for the IT infrastructure at hand. In particular, the role profoundly knows about the modeling, measuring, and other influencing factors of a set of attributes.

R4 – Workload Domain Expert Is experienced in the development and execution of workload (cf. Section 2.3), especially for the IT infrastructure at hand. Additionally, it is skilled in identifying and predicting the load that is about to be generated by executing workload on the IT infrastructure.

R5 – Management Is responsible for the IT infrastructure in general, communicates with (potential) consumers, and negotiates SLAs. In order to fulfill its responsibility for the IT infrastructure, to address the urgent need to consider influencing external factors like electricity prices or national law, and to respect the consumer demands, role R5 initiates and interprets reasoning activities and, if necessary, triggers follow-up activities.

R6 – Developer Develops workload and especially real world applications (cf. Section 2.3.1) that are executed on the IT infrastructure.

5.2 Template meta model

→ *Fig. 5.1 (p. 98)*

A reasoning project exposes a comprehensive and versatile nature, as particularly the compilation of an individual reasoning function covers manifold tasks and areas. The template meta model addresses this situation by describing, dictating, and formalizing all artifacts and actions of a reasoning project in order to

- force traceability,
- foster reproducibility,
- reduce error-proneness, and to
- support automation.

Figure 5.2 depicts the process model underlying meta model hierarchy (↗KB p. 259) top down: the *template meta model* provides language elements and concepts in five template classes to define *concrete template models* in the process model, which result in *template instances* that formalize and guide reasoning project execution. Templates were chosen as tool, because

- they ease work,
- they are "accepted and employed by most stakeholders" [342, p. 162],
- they support the creation of similar elements according to a common building plan, and
- they ensure that elements of the same class have the same structure.

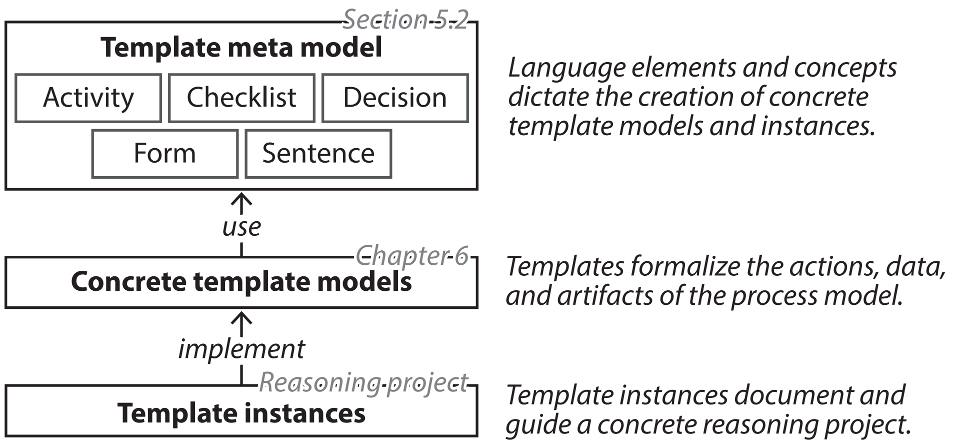

Section 5.2
Template meta model
Activity
Checklist
Decision
Form
Sentence
Language elements and concepts dictate the creation of concrete template models and instances.
use
Chapter 6
Concrete template models
Templates formalize the actions, data, and artifacts of the process model.
implement
Reasoning project
Template instances
Template instances document and guide a concrete reasoning project.

Figure 5.2: Template meta model hierarchy applied to formalize and guide the execution of a concrete reasoning project.

Section 5.2.1 itemizes the meta model's design objectives and intention, Section 5.2.2 explains fundamental concepts and the five template classes.

5.2.1 Objectives and intention

The following six objectives underlie the template meta model:

- **Build on existing results** There are manifold tools and approaches addressing partial aspects of a reasoning project and also of template meta model development, e.g., statistical regression methods (↗KB p. 268) derive model proxy functions from measurements, and UML activity diagrams (↗KB p. 270) formalize activities. Building on top of these solutions is beneficial in terms of development and validation efforts, understandability, and acceptance.
- **Enforce reproducibility** A main principle of a scientific mode of operation is reproducibility of conducted steps, partial results, and applied procedures and methods. Applying the same template for the same task(s) enforces fulfilling this profound, because it dictates the execution of the same steps to achieve a certain goal.
- **Ensure coverage** Rupp et al. [342, Sec. 7.1] claim specifying *all* aspects that are important to the current situation. Despite the objective's overlap with making information explicit and with fostering reproducibility, it is mentioned separately due to its fundamental role.
- **Foster expandability** The variety of reasoning project objectives produces an innumerable amount of potential focal points and interests. Fostering expandability aims at enabling adaptions of the process model to the resulting specific demands. Besides, templates entitle predefined extension points and label tasks or entries being optional to reduce costs, where appropriate. In contrast, reducing templates would contradict the objective of ensuring coverage (cf. above).
- **Make explicit** A scientific mode of operation requires making information, procedures, and notions explicit, especially aspects that are normally taken for granted [80]. The involvement of multiple stakeholders and roles in a reasoning project further emphasizes this need, since each role comes from a (potentially) different area and applies differing terms, concepts, and notions [342] (cf. Section 3.3.1).
- **Support automation** Automation is an eligible tool to reduce error-proneness, especially caused by manual execution. Although a first step towards automation is usually programming and scripting, template design focused on data exchange and code generation (cf. Section 8.2.5), because the extend and complexity of a reasoning project tend to overburden (simple) script approaches.

5.2.2 Template classes

The section explains the template meta model building blocks in the topmost box in Figure 5.2. In particular, it explains the following details:

- **Common namespace** Sets naming conventions and rules for labeling and identifying templates and their elements, respectively.
- **Class containment hierarchy** Defines how templates are related.
- **Template classes** Formalize a reasoning project in its entirety. The template classes **"Activity"**, **"Checklist"**, **"Decision"**, **"Form"**, and **"Sentence"** were derived to address typical needs and implications of a set of situations in a reasoning project. Template detailing is supported by a non computer science example to focus on template characteristics but on the example's technical details and correctness: A (simplified) airplane manufacturing process is chosen to highlight template characteristics and use, because it has several aspects in common with a reasoning project, e.g., involving several steps concurrently executed by manifold stakeholders and roles. Obviously, airplane manufacturing is much more complex as illustrated, but it is a very striking example. Due to the lack of stronger structuring parameters, the illustrative example orders template class detailing.

Common namespace

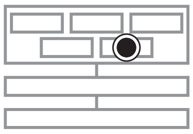
All templates are identified by a string containing the template class, an ascending numbering, and a label (*EG-5.1:1*). Their elements are referenced by a unique identifier, concatenating the parent template's identifier and the element identifier (*EG-5.1:2*). Element numbering scope is within a template, starting at *1* for each template. To confine the depiction of certain template elements from figures and tables, the label "Template element" is used.

EG-5.1 *(1)* T-A7 – Measuring entitles a *template* (T) of the class *activity* (A), numbered 7. *(2)* T-A7:D4 is the fourth *decision point* (D) in activity T-A7.

Class containment hierarchy

There is no strict containment hierarchy for templates. In contrast, a template of class A can be used by a template of class B and vice-versa, e.g., an activity template could employ a form template in a particular action, and a form template could require the explicit description of an executed activity by providing an activity template based description.

"Sentence" template class



→ *Fig. 5.2 (p. 100)*

Sentence templates (T-S) formalize objectives and details about a delimited task in a compressed and shareable way (*EG-5.2*). This addresses especially the involvement of several roles and stakeholders (cf. Section 5.1) in a reasoning project what forcefully requires ensuring that "different [stakeholders] use the same words for phrasing the same circumstances" [342, p. 168]. Formalization is provided in *one* sentence (template's eponym) to

- describe information in a tangible and intuitive way,
- implicitly limit the information's extend, and to
- facilitate the specification of a grammar for each sentence template to support a flexible, compact, formal, and powerful way of definition.

Sentence templates are provided as grammars (↗KB p. 255) in order to build on existing results (cf. Section 5.2.1). Compliant to NFR-7, grammars are context-free, since the expressive power of context-sensitive grammars is not required, and they use the highly compact notation of Backus and Naur [355] (↗KB p. 255). A sentence template is split in two parts:

- **Grammar part** Formulates a sentence in natural language, describing a situation, objective, or goal to achieve. Variables ⟨V⟩ represent aspects to set when instantiating the sentence template during a reasoning project. Production rules below the sentence define valid value ranges for each variable and name the value providing action or template, respectively, indicated by a ⇽ symbol.
- **Production part** "Executes" the defined production rules. The complexity of a reasoning project can render the execution an extensive endeavor, resulting in a description in paragraphs or even sections.

The following production rules describe generic (G) data types that are used by several sentence templates:

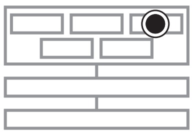
$$\begin{aligned}
\langle\text{GScale}\rangle &\vDash \langle\text{GScaleCategory}\rangle\langle\text{GQuantity}\rangle\\
\langle\text{GScaleCategory}\rangle &\vDash \text{nominal} \mid \text{ordinal} \mid \text{metric } (\nearrow\text{KB p. 267})\\
\langle\text{GQuantity}\rangle &\vDash \text{Mathematical quantity } (\nearrow\text{KB p. 263})\\
\langle\text{GPercent}\rangle &\vDash 0 \le x \le 100 | \text{x} \in \mathbb{R}\\
\langle\text{GIdentifier}\rangle &\vDash \text{Unambiguous, meaningful, preferably short string}\\
\langle\text{GIdentifierSet}\rangle &\vDash \langle\text{GIdentifierSet}\rangle | \langle\text{GIdentifier}\rangle | \epsilon\\
\langle\text{GRange}\rangle &\vDash x, y | x < y \wedge \text{x} \in \mathbb{R} \wedge \text{y} \in \mathbb{R}\\
\langle\text{GValue}\rangle &\vDash \text{x} \in \mathbb{R}
\end{aligned}$$

EG-5.2

> Manufacturing assembles an ⟨AirplaneType⟩ distance airplane at ⟨Factory⟩ [according to ⟨Regulations⟩].

⟨AirplaneType⟩ ⊨	Short | Medium | Long	⇽ T-D1
⟨Factory⟩ ⊨	Unique identifier of factory	⇽ T-A1
⟨Regulations⟩ ⊨	⟨GIdentifierSet⟩	⇽ T-C1

Sentence template T-S1 defines the objectives and ancillary conditions of airplane manufacturing. The template's grammar part above stipulates the selection of an airplane type, an assembling factory, and an optional set of regulations to fulfill. The template's production part is provided within the next template meta model examples.



→ *Fig. 5.2 (p. 100)*

"Decision" template class

Decision templates (T-D) support decision making by highlighting (potential) implications and affects of available options on reasoning project actions and artifacts (*EG-5.3:1*). In conjunction with the documentation of causes for and results of made decisions, this fosters reproducibility, because a decision can be traced and produces the same consequences, independently from the decision maker. Decision templates consist of three parts:

- **Implication table** Enumerates options and points up respective (potential) implications to the actions and elements of a reasoning project. The implication table consists of two columns: the left column names decision options, the right column explains the options and itemizes respective implications, labeled by a ↦ symbol. Implications mainly concern template elements, e.g., a particular action or decision. All other affects are placed in the decision tool (cf. next bullet).

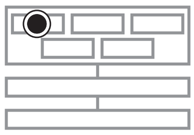
- **Decision tool** Provides a set of aspects that are relevant to the selection and might support its taking. For instance, a decision support could compare the difficulty on an ordinal scale of applying a set of tools, it could be a decision tree guiding the selection by a set of questions to answer, or it could be a comparison of pros and cons of selection options. The diversity of selection situations bans a fixed shape of the decision tool and a mandatory use (*EG-5.3:2*). Though, the decision tool is optional and not included in all decision templates.

- **Documentation** Stores a decision's reasons and results to enable tracing and reproduction at a later point in time. Documentation employs classifiers of the provenance information model (cf. Section 5.3) and is optionally assisted by a form template (see description on page 107).

Opt.	Explanation and implications
I1	**Short** – Airplane for distances of up to 1.000 km. ↦ Requires a new factory (action T-A1:A1). ↦ Requires no checklist fulfillment (variable T-S1:⟨Regulations⟩).
I2	**Medium** – Airplane for distances between 1.000 and 9.000 km. ↦ Requires no new factory (action T-A1:A1). ↦ Requires fulfillment of T-C1 (variable T-S1:⟨Regulations⟩).
I3	**Long** – Airplane for distances of more than 9.000 km. ↦ Requires no new factory (action T-A1:A1). ↦ Requires fulfillment of even stricter checklist.

(1) Decision template T-D1 processes T-S1:⟨AirplaneType⟩ (cf. *EG-5.2*). Its implication table above enumerates available options, explanations, and selection implications, respectively. *(2)* The higher *Return of Investment* (ROI) for medium distance airplanes is an exemplary implication to be placed in the decision tool, as it does not impact other templates, but only non-technical and economic aspects. The alluded diversity renders a decision tool example dispensable.

EG-5.3


→ Fig. 5.2 (p. 100)

"Activity" template class

Activity templates (T-A) decompose a task in manageable smaller elements by explicitly defining actions, execution ordering, partial results, and their use to achieve a certain goal (*EG-5.4*). They model a process considering concurrency, alternative ways, and specific behavior, using two parts:

- **Action flow** Formalizes actions and their flows, partial results, and responsibilities of a reasoning project. Modeling action flows employs UML activity diagrams (↗KB p. 270), because they
  - are closely related to the outlined intention of activity templates,
  - cover process modeling, task splitting, execution ordering, loops and concurrency, which is rather difficult in free text [344].

An action flow allocates the following meanings and elements of a reasoning project to activity diagram nodes (↗KB p. 270):

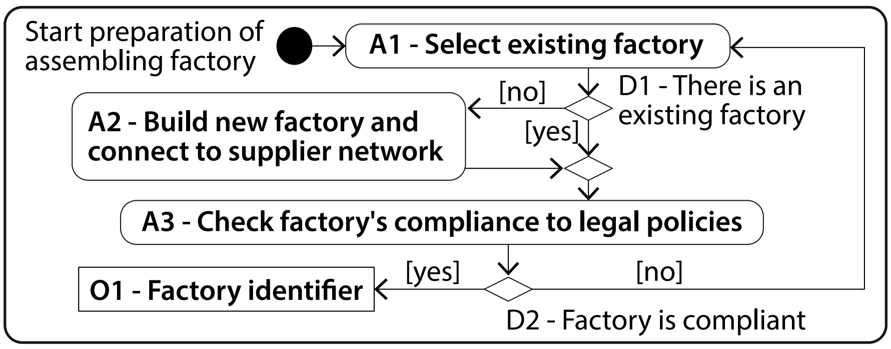
Action Single task or step (A prefix).

Object (Partial) result or static artifact (O prefix).

Decision Decision taken during execution (D prefix).

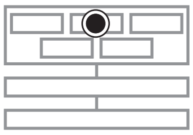
Action input Parameter(s) given by the calling action. The parameter's name in the calling action is provided in brackets.

- **Action details** Describes action flow part elements, mostly by an itemization or a set of sections. Usually, each action employs further templates to achieve its goal, e.g., an action taking a selection or documenting results might employ a decision or form template, respectively.


Start preparation of assembling factory
A1 - Select existing factory
[no]
D1 - There is an existing factory
A2 - Build new factory and connect to supplier network
[yes]
A3 - Check factory's compliance to legal policies
[yes]
[no]
O1 - Factory identifier
D2 - Factory is compliant

EG-5.4

Activity template T-A1 formalizes the preparation of a factory for airplane assembly. The following simplified action details describe its action flow depicted above: selecting an existing factory in action T-A1:A1 appropriate for airplane assembly is strongly affected by the value of T-S1:⟨AirplaneType⟩ (cf. implication table in *EG-5.3*). Depending on the result in T-A1:D1, action T-A1:A2 builds a new factory. Action T-A1:A3 checks the used factory's compliance to legal policies employing a checklist template. Depending on the result in T-A1:D2, the factory will be used and its identifier in T-A1:O1 is filled in T-S1:⟨Factory⟩ (cf. *EG-5.2*).



→ Fig. 5.2 (p. 100)

"Checklist" template class

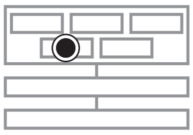
Checklist templates (T-C) enforce the fulfillment of a requirement set and can be used in a variety of situations (*EG-5.5*), e.g., describing a set of capabilities or formulating non-functional requirements (cf. Section 3.4), which dictate ancillary conditions and quality criteria of a system [342].

□ **Wiring is compliant to fire security policies** – Wires are encased by a special, heat-resistant material, which is produced with machine “MM-200:1”. Material production and installation are monitored during the entire airplane assembly process.

□ **Inter-seat space is compliant to client specification** – The seat-positioning robot can handle every non-standard distance between seats as defined by a client.

Checklist template T-C1 describes the regulations mandatory to the airplane manufacturing process, required by T-S1:⟨Regulations⟩ (cf. *EG-5.2*). The checklist template evaluates the factory newly built or selected in activity template T-A1 (cf. *EG-5.4*), as both checklist entries deal with the factory's assembly capabilities. For instance, the heat-resistant material encasing the wires must be provided by suppliers, integrated in the airplane, and monitored before airplane delivery. Hence, a fulfillment of the first entry depends on the entire manufacturing process and the factory capabilities.

EG-5.5



→ Fig. 5.2 (p. 100)

“Form” template class

Form templates make results explicit and traceable by dictating a set of form fields to fill (*EG-5.6*), each described by four entries:

- **Unique identifier** Unambiguously identifies the form field and enables its external referencing. It is built according to the template meta model's namespace, consisting of the prefix F and a numbering.
- **Title** Shortly summarizes the field's content and purpose.
- **Valid data** Specifies the form field's valid data (range). It can cover simple data types, like numbers, dates or strings, but also individual (complex) data types, like a template identifier or an UML diagram type. In case the valid data type is string, the entry is omitted. Optionally, a justification for narrowing the data range can be provided.
- **Filling objectives** Describes the field's purpose to support stakeholders in filling the field in terms of information extend, coverage, and focus.

Every documentation and action requires a different set of fields. Hence, the only field the form template class defines mandatorily is the *form* *identifier* F1 that distinguishes several instances of the same form template. All other fields depend on the template's deployment and are defined as required.

EG-5.6

F1 – Form identifier

Filling objectives and rules – Unambiguously identify a particular instance, i.e., a filled-out version of the form template.

F2 – Responsible engineer

Valid data – Name and email address
Filling objectives and rules – In case there are questions regarding the airplane blueprint, provide sufficient contact information.

F3 – Blueprint

Valid data – AutoCAD diagram at scale 1:2000
Justification – Only AutoCAD diagrams are allowed, because all involved stakeholders use this tool.
Filling objectives and rules – The AutoCAD diagram should depict the entire airplane and highlight wiring interfaces.

An airplane is assembled according to a given blueprint. Form template T-F1 formalizes the documentation of this blueprint. In particular, it specifies an unique identifier for the blueprint, calls for information about the responsible engineer, and documents the blueprint itself, provided as AutoCAD diagram at scale 1:2000.

5.3 Provenance information model

→ Fig. 5.1 (p. 98)

Provenance enforces reproducibility, ensures immutability of produced results, and eases result interpretation [151, 304] by keeping track of a data item's (entire) creation history [304]. Hence, provenance is seen as a critical component of the scientific method [140, 116]. In the context of a reasoning project, the alluded data items are all (partial) results assembled throughout reasoning project execution and especially during the reasoning function compilation process, like integrated models, measurements for model proxy creation, or attribute calculation aspects.

The section details the process model's *provenance information model*, an "abstraction of real-world entities into formal constructs that can be represented in computer systems" [121, p. 298]. It facilitates provenance for a reasoning project by providing capabilities to describe and store information, data types, and relationships [181] that document the aforementioned reasoning project data items. The provenance information model supports all common provenance concepts, like storing meta information, enabling integrity enforcement, ensuring non-repudiation, and fostering data tracing [327], as explained throughout this section.

Besides, the provenance information model exposes a broad applicability (cf. Section 4.1) and expandability (cf. Section 5.2.1), as its underlying object orientation allows individual extensions by sub classing, e.g., to describe specialized IT infrastructure components as proposed by Sharapov et al. [362]. Employing object orientation for expandability support is a common approach, which is applied by wide-spread information models, like the *Grid Laboratory for a Uniform Environment-Schema* (GLUE)[1] (cf. Section 2.2.3) that even recommends to "create specializations of the conceptual model and to implement them by extending the concrete data models" [22, p. 6]. In addition, object orientation is chosen due to its understandability as well as its modularity and reusability [390], which are important for NFR-7 and NFR-8, respectively. According to the applied object orientation, UML class diagrams (↗KB p. 271) formalize the provenance information model.

Section 5.3.1 explains the provenance information model package structure, the successive Sections 5.3.2, 5.3.3, and 5.3.4 detail packages consisting of generic classifiers that are used throughout a reasoning project. All other classifiers and packages are discussed in the remaining sections and chapters, accordingly. For instance, Sections 5.5 and 5.6 discuss classifiers modeling measuring aspect and IT infrastructure attributes, respectively.

5.3.1 Package structure

The extend and complexity of a reasoning project require numerous classifiers to cover relevant details sufficiently. UML (↗KB p. 274) addresses this extend by bundling all classifiers that are related in some way [344] in so-called *packages*. In addition to this general structuring, the provenance information model's packaging pursues particularly two aspects:

- **Flexibility** Addresses the differing development pace and the probability of adaptions, e.g., IT infrastructure classes might need adjustment more often than scale definitions.

[1]Section 7.4.5 explains why existing information models are not used or extended.

- **Reusability** Reflects the involvement of several stakeholders in a reasoning project (cf. Section 5.1) and especially the need for diverse provenance information model slices for diverse tasks (cf. NFR-8).

Hence, packaging decouples classifiers at those points that have differing use [344] and that ease distribution of self-contained parts of the provenance information model. Bundling classifiers in packages and simultaneously providing clear UML package diagrams calls for describing classifier correlations in class attributes, instead of the more common associations (↗KB p. 271). Although there is no semantic difference, the latter produces several association lines that tend to make diagrams confusing.

Figure 5.3 overviews the provenance information model in an UML package diagram, the fully detailed and complete UML class diagram is depicted in Figure C.2 on page 312 in Appendix C. The subsequent itemization shortly summarizes the intent of each package and contained classifiers by anticipating details that are introduced and explained in later sections.

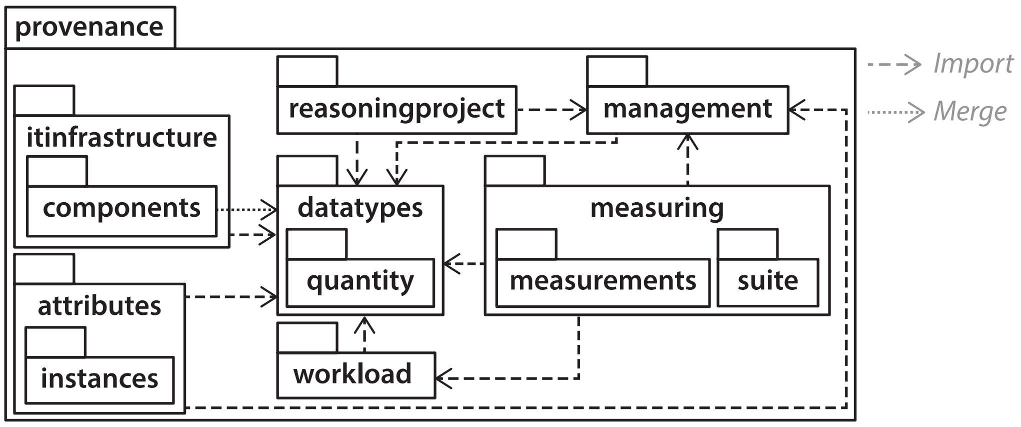


Figure 5.3: UML package structure of the provenance information model.

`reasoningproject` Describes a reasoning project in general and all aspects that affect other parts globally, e.g., constraints, assumptions, or reasoning parameters. The package is explained in Section 5.3.4.

`datatypes` Provides a set of (complex) data types, like unique identifiers, time stamps, or quantities, to enable a platform independent formalization, since the provided data type concepts can be implemented in many different ways. The containment of the unique identifier causes a package import by all other packages. The contained package `quantity` provides (helper) data types to describe a quantity, scale and domain (↗KB p. 267). The packages are explained in Section 5.3.2.

`management` Keeps records about responsibilities, and provides keywords to tag objects.

`itinfrastructure` Describes a reasoning project's IT infrastructure. Its components and their relationships are covered by a separate package `components` to foster reusability (cf. above). IT infrastructure components are identified by a sub class of the generic `UniqueId`, which causes a merge association to the `datatypes` package. The packages are explained in Section 5.4.5.

`attributes` Covers a reasoning project's IT infrastructure attributes. Classifiers describing attribute concepts are located in the top-level package, the different attribute concept instance approaches in the dedicated sub package `instances`. The packages are explained in Section 6.1.3.

`measuring` Describes all elements related to measuring, like a measuring instrument, or configuration parameters. A measuring result can be simple or derived measurements, which calls for a dedicated `measurements` package. The packages are explained in Section 6.1.3.

`workload` Describes workload, load generation, and configuration parameters. The package is explained in Section 6.1.6.

5.3.2 Data type package

The package groups all data types of the provenance information model that depend on the applied implementation (*EG-5.7:1*), or the reasoning project objectives (*EG-5.7:2*), or that expose technical or semantic encoding issues (*EG-5.7:3*). In particular, the package pursues two goals:

- **Support classifier reuse** Achieving a broadly applicable process model (cf. Section 4.1) also calls for a platform independent (↗KB p. 260) provenance information model that is agnostic to implementation details, especially specific data types. Placing these data types in a single package encapsulates implementation approaches for their reuse by all other classifiers in the provenance information model.
- **Provide single extension point** Some data types implicitly formulate challenging issues, e.g., documenting selection reasons for a measuring instrument in a computer-processable way. Their broadly applicable and generic modeling is a dedicated research discipline and not within the thesis scope, as carried out in future work (cf. Section 8.2.5). In order to provide a single extension point to these research efforts, the

`datatype` package collects relevant data types to ease incorporation of suitable modeling approaches. Accordingly, some classifier attributes in the package are simply modeled as `String`, e.g., a `Formula` rule or a `Selection` reason, since the machine-readable encoding of these information is in the scope of the aforementioned research efforts and not cardinally required for provenance anyway.

EG-5.7

(1) A globally unique identifier can be implemented as *Universally Unique Identifier* (`UUID`) in Java, or as *Globally Unique Identifier* (`Guid`) in .Net; a time stamp can be realized as UNIX time stamp or as MySQL `DATETIME`. Both examples illustrate that the same data type concept can be implemented in manifold ways. *(2)* The impact of the reasoning project objectives on the data type realization is also shown for the time stamp. There is a wide range of time granularity in related research: Contreras et al. [104] derive power estimates at *sub-second* time intervals [138], the IBM tool *Amester* can sample the power consumption at intervals *from 1 millisecond on up* [71], and the measuring instrument *Energenie EGM-PWM-LAN* (cf. Section 7.2) supports a time resolution of only *1 second*. This diversity underpins that the concrete time stamp realization depends on the specific goals, in that case the pursued time granularity. *(3)* The encoding of a mathematical formula in a generic, computer-readable way is a semantic encoding issue, as carried out in Section 8.2.5.

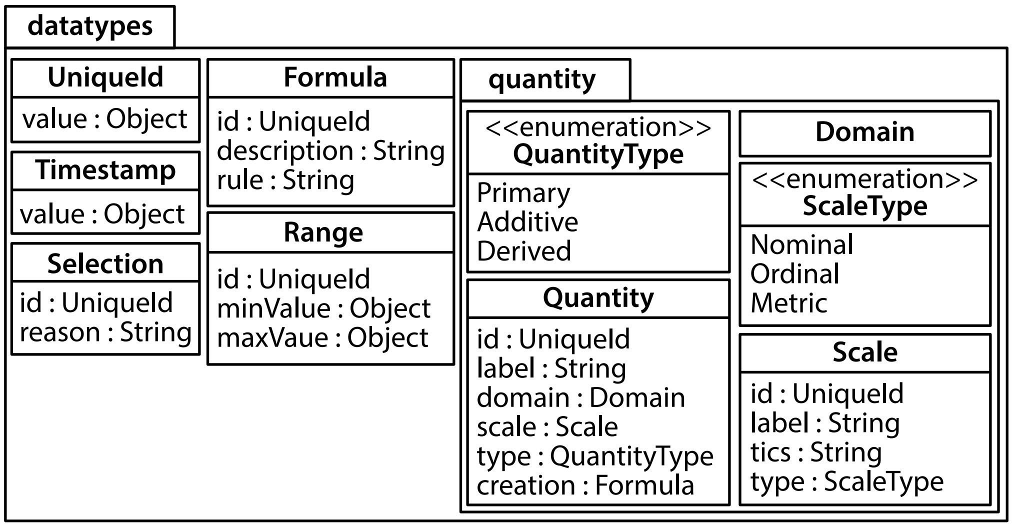


Figure 5.4: The provenance information model's `datatypes` package, grouping classifiers that represent (complex) data types.

Figure 5.4 depicts the `datatype` package and contained classifiers:

`UniqueId` An identifier being globally unique and unambiguous within a reasoning project. The identifier value is modeled in the generic data type `Object` to emphasize the need of an individual platform specific implementation. For identification purposes, each class in the provenance information model has an attribute `id` of type `UniqueId`.

`Timestamp` A point in time, whose generic data type `Object` emphasizes the need for an individual platform specific implementation (*EG-5.7:1*). Besides the platform specific aspects, the implementation also depends on the time granularity required by the respective reasoning project and the integrated models (*EG-5.7:2*). Achieving simplicity (cf. NFR-7, Section 3.4) requires a time granularity definition appropriate to the reasoning project, and particularly not as accurate as possible.

`Formula` A mathematical or statistical formula, e.g., describing the calculation of a derived measurement or an integrated model. The formula's objectives and use as well as its calculation rule are stored in the `description` and `rule` fields, respectively. The latter one is of the `String` data type due to the above outlined reasons.

`Selection` A made selection, e.g., in a decision template (cf. Section 5.2.2). The selection's justification is modeled by the class attribute `reason`, having the data type `String` due to the above outlined reasons.

`Range` A (numerical) range, such as an attribute concept range (cf. Section 6.1.3), limited by a lower and upper value.

`Quantity` A quantity (↗KB p. 263) whose capability is summarized in the `label` field. The quantity's building blocks *domain*, *scale* and *type* are modeled by the fields `domain`, `scale`, and `type`, respectively. The latter can realize three values, provided by the `QuantityType` enumeration. The variety of potential ways to achieve additive and derived quantities requires an explicit description of the applied processing approach [80]. Accordingly, the `creation` field stores the applied addition or derivation formula.

`Domain` A (mathematical) domain, like rational numbers, real numbers, or a set of literals. There is a dedicated class to address the platform independent objective described above.

`Scale` A (mathematical) scale (↗KB p. 267). The `label` field summarizes its intent, the `ScaleType` enumeration provides its type.

5.3.3 Management package

The package provides a set of classifiers to maintain management related information, pursuing two objectives:

- **Enable provenance of responsibility** The involvement of several roles and stakeholders in a reasoning project (cf. Section 5.1) underpins the (potential) need to do "provenance not only of data but of intellectual property" [188, p. 1822], e.g., to identify the originator of a measurement [188], model selection, or an additive quantity.

- **Enable instance discovery** A reasoning project requires a high number of classifiers (cf. Section 5.3.1) and produces an even higher number of classifier instances, which calls for support of their localization, particularly of their instances, e.g., a specific integrated model, a measurement value, or an IT infrastructure attribute instance.

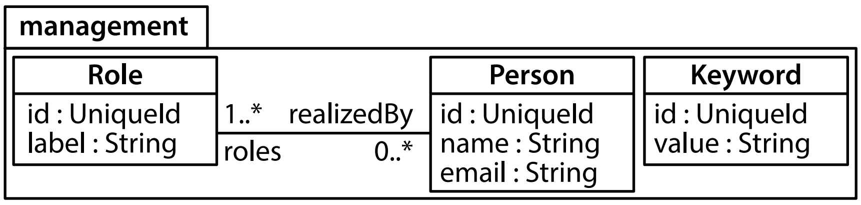


Figure 5.5: The provenance information model's `management` package, grouping classifiers for provenance of responsibility and object discovery.

Figure 5.5 depicts the `management` package and contained classifiers:

`Role` A role that is involved in a reasoning project and responsible for one or multiple tasks according to the notion presented in Section 5.1. The potential variety of reasoning project situations renders an extensive role model unproductive. Instead, a role is only described by a `label` field and acts as dedicated extension point for individual sub classing, e.g., to assign authorization information or describe a role hierarchy. A role can be realized by zero or an arbitrary set of persons.

`Person` A (physical) person that realizes one or multiple roles (cf. above). Given the same reasons as for the `Role` class, a person is tightly modeled, consisting of a `name` and `email` field to foster provenance of responsibility and contacting the person. Otherwise, the class acts as dedicated extension point for individual sub classing.

`Keyword` A simple keyword that can be assigned to several elements of the provenance information model, like measuring elements or a reasoning project. In other words, an arbitrary set of `Keyword` objects can be assigned to classifiers to facilitate searches for suitable elements, e.g., to support the arrangement of management reports [1].

5.3.4 Reasoning project package

The package provides classifiers for describing a reasoning project and related elements having a global impact, namely the reasoning parameters, constraints, and assumptions. The package description takes on a special position, since it is split in two parts: the section at hand *overviews* the package classifiers to address their central role that calls for a description along with the also central `datatype` (cf. Section 5.3.2) and `management` (cf. Section 5.3.3) package. In contrast, Chapter 6 extensively *details* the concrete use of the classifiers as referenced subsequently. Figure 5.6 depicts the `reasoningproject` package and contained classifiers:

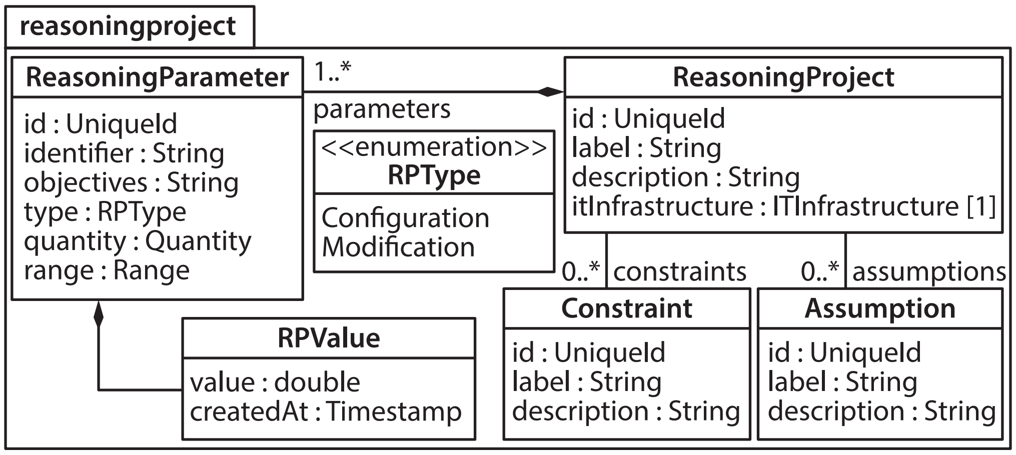


Figure 5.6: The provenance information model's `reasoningproject` package, grouping classifiers that represent execution-related (meta) information of a reasoning project.

`ReasoningProject` The reasoning project, acting as umbrella and connection point to other elements. This results in a rather compact class, built of a `label` and a general `description` of the reasoning project.

`ReasoningParameter` A reasoning function parameter according to the notion presented in Section 4.1. The `RPType` enumeration represents the parameter set decomposition in modification and configuration parameters. The `ReasoningParameter` is connected to a reasoning project by a composition (↗KB p. 271), because it is highly specific and only suitable as long as the reasoning project exists. Section 6.1.4 details the class and its use in action T-A3:A4.

`RPValue` A single value of a reasoning parameter at a particular point in time. The value is extracted from the `ReasoningParameter` class and assigned to a time stamp to enable consideration of chronological development, as detailed in Section 6.1.2 and stated in decision template T-D3.

`Assumption` An assumption made in the context of a reasoning project. Section 6.1.7 details the class and its use in action T-A3:A7.

`Constraint` A constraint that has to be respected in any situation of a reasoning project, e.g., prohibit a special model type or negate license fees. Section 6.1.8 details the class and its use in action T-A3:A8.

5.4 IT infrastructure notion

→ *Fig. 5.1 (p. 98)*

The section describes the developed notion of IT infrastructures that underlies the presented process model. For illustrative purposes, Figure 5.7 overviews in a simplified exemplary HPC cluster system (cf. Section 2.2.2) the notion's four building blocks:

Graph-based representation ① Models IT infrastructure components as nodes and logical communication dependencies as edges, as carried out in Section 5.4.1.

Interfacing with third-party models ② Supports the data import and exchange about the considered IT infrastructure, as detailed in Section 5.4.2.

Black box approach ③ Initially considers every IT infrastructure component as black box, as described in Section 5.4.3.

Description terminology ④ Employs the terms *capability*, *property*, and *attribute* to describe an IT infrastructure and its components, as explained in Section 5.4.4.

Section 5.4.5 introduces the provenance information model classifiers for IT infrastructure modeling.

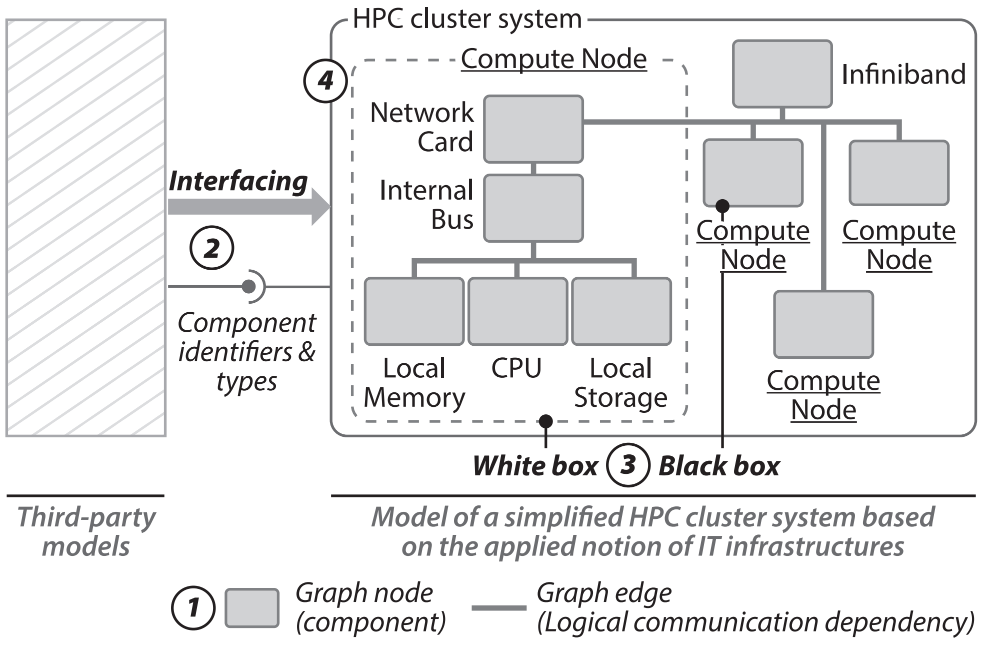


Figure 5.7: Summarizing overview illustrating the process model underlying notion of IT infrastructures.

5.4.1 Graph-based representation

The process model underlying notion of IT infrastructures bases upon graph concepts to facilitate the employment of graph algorithms and methods, e.g., assigning information to graph nodes, using dependency detection algorithms, or defining sub graphs for delimiting a specific IT infrastructure component sub set. Compared to other work, like Aguilera et al., who model "a distributed system as a graph of communicating nodes [...] in which case the edges are the network" [14, p. 75], the mapping of IT infrastructure aspects onto graph elements is different (*EG-5.8*):

Node Abstracts hardware entities and models type-independent *all components* of an IT infrastructure.

Edge Models *logical* communication dependencies between components, instead of physical communication hardware.

Whenever two or more components communicate or exchange data, there are a communication component (node) and logical communication dependencies (edges). Since the lion's share of communication is bi-directional and for the sake of simplicity, edges are non-directed.

EG-5.8 Figure 5.7 illustrates the graph-based notion for the exemplary HPC cluster. Both, communication components, like the internal bus and Infiniband, and worker components, like the local memory and the CPU, are represented by nodes, depicted as rounded boxes. The nodes are connected via edges, expressing a logical communication dependency and not a physical communication. This is notably articulate for the internal bus, which is represented by a node and not by edges.

The introduced graph representation, and especially transferring communication components from edges to nodes, allows reasoning to handle all IT infrastructure components equally (cf. Section 4.1), which is urgently required due to two reasons (cf. NFR-1 on page 84):

1. Nearly all IT infrastructure attributes are strongly influenced by communication aspects, e.g., performance is not only composed of computing cores and I/O performance, but also of communication interconnect [91] (cf. Section 2.4).
2. IT infrastructures render functionality by a complex qualitative and quantitative component interplay, which avoids identifying the specific contribution of a single component to its functionality [268].

5.4.2 Interfacing with third-party models

Use Case UC-3.2 in Section 3.3.2 outlines that available databases often contain suitable information about the considered IT infrastructure, e.g., in a *Configuration Management Database* (CMDB). To use existing information and save efforts, foster actuality, and avoid duplicates and integrity flaws, the process model underlying notion of IT infrastructures provides two provenance information model elements (cf. Section 5.4.5) as interfaces to a potentially existing database or tool landscape (*EG-5.9*):

- **Unique identifier** Identifies an IT infrastructure component. Storing similar or even the same values as the third-party model facilitates information exchange and keeping the model in sync with alterations in the considered IT infrastructure, since components can easily be related to corresponding objects.

- **Component type** Describes the type of an IT infrastructure component. The usable value range is extracted from the corresponding model and applied to describe components.

The arrow in Figure 5.7 represents the component mapping and particularly the data import in the provenance information model from the dashed box, being arbitrary third-party models. Mapping IT infrastructure components to *Common Information Model* (CIM) objects [123], for instance, would use the CIM object identifier `InstanceID`, inherited from the global super class `ManagedElement.InstanceID`; the CIM class inheritance hierarchy, like *ManagedElement* > ... > *NetworkAdapter* or *ManagedElement* > ... > *DiskDrive*, would define the infrastructure component type set.

EG-5.9

5.4.3 Black box approach

Section 4.3 introduces the black box approach as one of the process model's general implementation approaches, the section at hand further details it in the context of the process model's IT infrastructure notion.

The black box approach initially considers every IT infrastructure component as *black box*, hiding the component's internal matters. Whenever knowledge about the black box internals is required, it is refined to a *white box*, resulting in a sub graph (cf. Section 5.4.1) that consists of several newly modeled components (*EG-5.10*). The black box approach has two advantages for IT infrastructure modeling:

- **Support multiple granularity levels** According to its objectives, a reasoning project might be interested in differing IT infrastructure aspects, e.g., considering a compute node in detail to examine its power consumption, while neglecting storage nodes. Globally increasing model granularity to the desired level would quickly lead to an unmanageable complex model due to the scale of IT infrastructures. In contrast, multiple granularity levels within the same model are mandatory to facilitate a good trade-off between accuracy, complexity, and time to solution [337]. The black box approach fulfills this demand by punctually refining only elements that are of interest for a reasoning project and leaving all other components on a high abstraction level.

- **Foster fast IT infrastructure modeling** The extend of IT infrastructures mostly cause modeling being a laborious and costly task. The black box approach reduces required cost and time in two ways:

  - It allows modeling to start with a coarse-grained model consisting of a (small) set of black boxes that are refined exclusively as required.

- Using the IT infrastructure component types (cf. Section 5.4.2), the black box approach facilitates the (automatic) propagation of a refinement within the entire graph: black box nodes are identified by their type and replaced with the refining sub graph, respectively.

Algorithm 1 describes the aforementioned propagation in pseudo code: In a preparation step, the refining sub graph is stored in *refinedBlackBox* for further use (line 1). Afterwards, an iteration over all black boxes having the same component type as the refined black box (line 2) replaces located black boxes (*currentBB*) by the sub graph in two consecutive steps. First, the black box refinement *refinedBlackBox* is copied and placed into the graph by re-connecting the edges that were previously connected to the replaced black box (line 3). Second, the black box identifier value of all vertices in the refining sub graph is set to facilitate recognition of initially modeled black boxes (line 4).

Algorithm 1: Propagation of black box refinement

```
1 refinedBlackBox = describes refinement of blackbox
2 while currentBB = getUnrefinedBlackboxOfType(blackbox.type) do
3     placeRefinementIntoTree();
4     setBlackboxIdentifierTo(currentBB.identifier);
5 end
```

EG-5.10

The HPC cluster system in Figure 5.7 consists of four compute nodes, of whom the far left one was refined to a white box delimited by the dotted rectangle. It consists of a *Local Memory* component, a *CPU* component, a *Local Storage* component, an *Internal Bus* component, and a *Network Card* component that collectively assembly a new sub graph. The other three compute nodes remain black boxes.

Motivation and point in time for refining a black box into a white box depend on the reasoning project objectives and interests, as detailed in Section 6.1.2. In particular, the reasoning objectives imply differing requirements on the information available about the considered IT infrastructure, e.g., which attributes should be reasoned about for which components at which granularity level. Consequently, the IT infrastructure notion defines the general black box approach, but does purposely not provide a set of black box distinctiveness or globally valid refinement rules.

5.4.4 Description terminology

Three terms, sketchy handled in Section 2.4, describe an IT infrastructure:

- **Capability** A qualitatively well-defined (low level) functionality that is exposed to the user or application, like data transfer, data storage, or computation. It is workload execution agnostic, as the functionality remains the same, independent of the currently executed applications or tasks. The term is of secondary interest, as is only acts as helper information, e.g., a scheduler might use it for mapping workload onto IT infrastructure components [5] (cf. Section 2.3.3). High-level capabilities, like sophisticated *reliable* file transfer, are considered as *services* that are built on top of an IT infrastructure.

- **Property** A quantitative description considering infrastructure component capabilities as white boxes at any time, e.g., describing the theoretical maximum clock speed of a component offering computation capabilities. Property values can be gathered from vendor specifications, benchmarking, trace analysis, or any other source for (empirical) data. It is workload execution agnostic, because the general maximum value remains the same, independent of the currently executed applications or tasks. Exemplary property definitions can be found in the *Job Submission Description Language* (JSDL) [23, Sec. 6.4] as it specifies requirements of computational jobs. Just like capabilities, the term is of secondary interest, as it also acts only as helper information, e.g., for workload mapping [5] (cf. Section 2.3.3).

- **Attribute** A quantitative description of an IT infrastructure that considers its components as black boxes during workload execution at a particular time step, e.g., "power consumption in Watt per time step" or "time to completion of result calculation". The term is of primary interest, as it is in the main focus of the presented process model. Attribute concepts are further detailed in the dedicated Section 5.6.

5.4.5 Provenance

Figure 5.8 depicts the provenance information model's `itinfrastructure` package to describe an IT infrastructure according to the above detailed notion, using the following classifiers:

`ITInfrastructure` A representative of an IT infrastructure and connection point for other provenance information model classifiers that need to

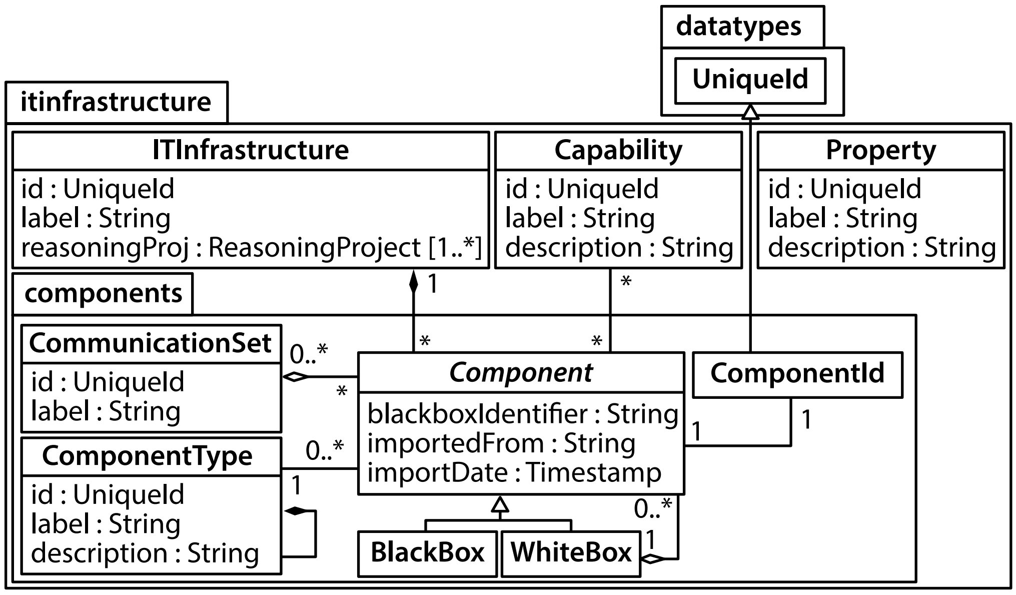


Figure 5.8: The provenance information model's `itinfrastructure` package, grouping classifiers for IT infrastructure modeling.

reference the considered IT infrastructure. Though, the class contains only a `label` field and a set `reasoningProj` of employing reasoning projects (cf. package `reasoningproject`) to facilitate the IT infrastructure model's reuse by multiple reasoning projects, as motivated in Section 5.3.1. Noteworthy, the `reasoningProj` set is a tangible example for the use of class fields instead of associations to describe classifier correlations (cf. Section 5.3.1). The components of the IT infrastructure are covered by `Component` objects below.

`Component` An IT infrastructure component according to the notion Section 5.4.1 introduces. The class is marked abstract due to two reasons:

1. An IT infrastructure component is exclusively either a black box or a white box (cf. Section 5.4.3), and the abstract nature of the `Component` class enforces choosing one of the two options.
2. Modeling sub graphs (cf. Section 5.4.1) recommends the recursive data structure *composite pattern* [159], whose contained abstract class is realized by the `Component` class.

Hence, the `Component` class only collects commonalities of black boxes and white boxes, namely a `blackboxIdentifier` as well as an `importedFrom` and `importDate` field to trace refinements and imports from third-party models, respectively. The provenance information model underlying object orientation and especially the sub classing are of special importance to component modeling, because the `Component` class can be arbitrarily extended, e.g., to store additional fields like operating system or other casuistic characteristics.

`ComponentId` A unique identifier of an IT infrastructure component. The class extends the global `UniqueId` not to expand the field list, but purely to enable and ease interfacing with third-party models (cf. Section 5.4.2), by providing a dedicated class. The data type `Object` of the inherited `value` field flexibly stores third-party identifiers, like the `instanceID` of a CIM database (cf. *EG-5.9*).

`ComponentType` Set of component types as introduced in Section 5.4.2. A composition association (↗KB p. 271) models the component type hierarchy, because child component types must be implicitly deleted if the parent component type disappears.

`BlackBox` A black box according to the notion introduced in Section 5.4.3. It provides no additional fields, since the class' only intend is to underpin the black box nature. In the context of the aforementioned composite pattern, the `BlackBox` class is a leaf.

`WhiteBox` A white box according to the notion introduced in Section 5.4.3. It provides no additional fields, since the class' only intend is to underpin the white box nature. In the context of the aforementioned composite pattern, the `WhiteBox` class is a composite.

`CommunicationSet` Collection of all components that are communicating in some way, or in other words, a representation of logical communication dependencies between components whose purpose is summarized in the `label` field. Referring to Example EG-5.8, a `CommunicationSet` would contain the internal bus and the three connected components within the compute node white box.

`Capability` A component capability according to the notion introduced in Section 5.4.4. The class provides only the fields `label` and `description` to explain the capability in general, because its secondary role and the variety of information required for a particular situation (cf. Section 5.4.4) render a further refinement unsuitable. In contrast, it acts as extension point for creating sub classes.

`Property` A property according to the notion introduced in Section 5.4.4. The class provides only the fields `label` and `description` to explain the capability in general, because its secondary role and the variety of information required for a particular situation (cf. Section 5.4.4) render a further refinement unsuitable. In contrast, it acts as extension point for creating sub classes.

5.5 Measuring

→ Fig. 5.1 (p. 98)

A measuring is the "reasonable, exclusively descriptive, assessment-free, and rule-based mapping of a finite set of characteristics at a discrete point in time t on a symbol set" (↗KB p. 256). This definition roughly guides measuring and implicitly defines some (high-level) quality criteria regarding measuring execution and gained results. However, three issues require a concretion of the alluded generic implicit criteria, and particularly a template formalization to achieve useful results, as the subsequent itemization motivates using ⓝ for later reference:

- **Variety of methods** Although most measurements can be traced back to counting [312, 382, 16] (cf. [352]) there are several differing approaches and techniques to conduct a measuring ① (*EG-5.11*).

EG-5.11

> Measuring software complexity can be achieved by counting 1) the lines of code, 2) the hours required to understand the code, or 3) the amount of intuitive code segments.

- **Uncertainty** Measuring "is afflicted with [...] uncertainty, caused by imperfection of tools, observers and environmental factors" [377, p. 115] ②.
- **Reasoning project demands** The specific circumstances of a reasoning project tend to put further demands on the extend and obligation of measuring rules, particularly the
  1. involvement of several stakeholders with potential goal conflicts ③,
  2. inordinate difficulty of some measuring setups caused by the complexity of considered IT infrastructure elements ④, and the
  3. potentially challenging time- and cost-constraints ⑤.

This requires a strict rule set for mapping facts to measurements [80]. Table 5.1 itemizes in alphabetical order four objectives for the rule set that can be derived from the issues itemized above. Besides, Table 5.1 highlights the templates that realize the objectives and collectively formalize a measuring.

Objective	*Addressed issue*				
	①	②	③	④	⑤
Exclusively descriptive	✓			✓	
Enforce an exclusively descriptive character without any assessment, as measurements *observe* facts [80] ④, aiming at gathering objective (raw) data [312, 352, 382] ①. → T-C2 (in T-A2:A3) validates requirements of a *descriptive* measuring.					
Reasonable	✓				✓
Align the measuring activity and related decisions to the respective use in order to address the variety of measuring approaches ①, to reduce (dispensable) costs ⑤, and to cover the diversity of real world element characteristics (↗KB p. 256). → T-S2 (in T-A2:A1) explicitly specifies the measuring objectives.					
Reproducible		✓		✓	
Ensure reproducibility ② and repeatability ④ (cf. Section 5.3) of data gathering by documenting the conditions under "which the repeatability of measurement results is achieved" [35, p. 6]. → T-F2 (in T-A2:A4) documents the repeatability conditions.					
Stable			✓		
Ensure equal measurements independently from the executing entity or person [55]. Besides, avoid the *feedback mechanism* ③, the potential (subconsciously) influence of the measuring by the executing person, as measurements might affect it in a negative way [56]. → T-A2 dictates the actions of a measuring.					

Table 5.1: Objective overview of templates that formalize measuring.

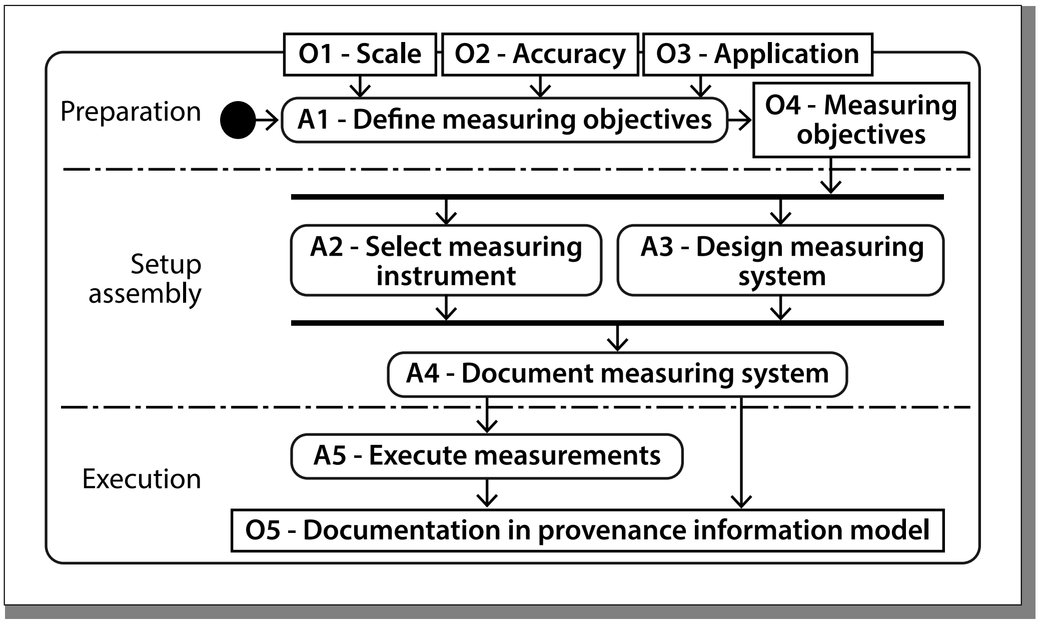
Section 5.5.1 describes the measuring activity T-A2, and Section 5.5.2 provides the provenance information model parts relevant to measuring.

5.5.1 Measuring activity

Activity template T-A2 - Measuring formalizes the execution of a measuring as a "set of specific operations used in the performance of particular measurements [...]" [35, p. 6], compliant to the objectives overviewed in Table 5.1. Collectively, an execution instance of T-A2, gained data, and achieved results are named a *measuring suite*. Two actions use a measuring suite in a reasoning project (cf. Section 6):

1. Action T-A3:A13 uses it for model proxy creation.
2. Action T-A3:A16 uses it for reasoning function evaluation.

Noteworthy, the measuring activity purposely omits any interpretation or further use of data. Instead, it encapsulates the measurement based data gathering and provides the gained raw data for further processing to the two calling actions T-A3:A13 and T-A3:A16, respectively. Template element 5.1 depicts the action flow of activity template T-A2, the following sections provide its action details. Template element 5.1 illustrates the three parameters *scale* (T-A2:O1), *accuracy* (T-A2:O2), and *application* (T-A2:O3) that are passed to the measuring activity by the two applying actions alluded above. Besides, Template element 5.1 highlights the template's three phases:


O1 - Scale
O2 - Accuracy
O3 - Application
Preparation
A1 - Define measuring objectives
O4 - Measuring objectives
Setup assembly
A2 - Select measuring instrument
A3 - Design measuring system
A4 - Document measuring system
Execution
A5 - Execute measurements
O5 - Documentation in provenance information model

Template element 5.1: Action flow of T-A2 - Measuring.

- **Preparation** Extracts the measuring objectives from the given parameters T-A2:O1, T-A2:O2, and T-A2:O3. Action T-A2:A1 transforms the activity's parameters into the measuring objectives T-A2:O4.
- **Setup assembly** Assembles the measuring setup compliant to the measuring objectives. Action T-A2:A2 chooses appropriate measuring instrument(s), action T-A2:A3 designs and evaluates the measuring setup, and action T-A2:A4 documents the measuring setup for reproducibility.
- **Execution** Executes the measuring compliant to the measuring objectives. Action T-A2:A5 uses the assembled measuring setup, and documents results in T-A2:O5 employing classifiers of the provenance information model's `measuring` package.

T-A2:A1 – Define measuring objectives

The three externally given parameters T-A2:O1, T-A2:O2, and T-A2:O3 dictate the measuring objectives. Yet, there is an urgent need for additional harmonization and especially formalization, as several roles, each applying a different understanding, might be involved. Common measuring practice further enforces this need by stating that "before a measuring is executed, both the measurements and the measuring setup must be thoroughly specified" [377, p. 114] (*EG-5.12*).

The urgent need of explicitly stating measuring objectives originates in natural science. The realm of physics substantiates the non-triviality of a specification: measuring the length of a metal bar in micrometer requires the indication of temperature and air pressure, which can be omitted for measuring in millimeter [377]. *EG-5.12*

Action T-A2:A1 uses sentence template T-S2 - Define measuring objectives to guide the entire measuring activity, and to synchronize (potentially distributed) measuring results. The filled out instance of T-S2 is stored in T-A2:O4. Template element 5.2 depicts the sentence template's grammar part, covering the following variables:

⟨**MeasurementScale**⟩ Explicitly defines the targeted scale (↗KB p. 267) of the measurement, extracted from parameter T-A2:O1. Still, the fundamental role of clearly defined measuring objectives for a measuring activity's success calls for a dedicated variable. Besides, a mismatch between the scale requested by the calling actions and the achieved scale would produce unsuitable measurements.

Gain ⟨MeasurementScale⟩ with an accuracy of ⟨Accuracy⟩ on ⟨Components⟩ using ⟨Setup⟩.

⟨MeasurementScale⟩ ⊨ ⟨GScale⟩	⇽	T-A2:O1
⟨Accuracy⟩ ⊨ ⟨GPercent⟩	⇽	T-A2:O2
⟨Components⟩ ⊨ ⟨GIdentifierSet⟩	⇽	T-A2:O3
⟨Setup⟩ ⊨ ⟨GIdentifier⟩	⇽	T-F2

Template element 5.2: Grammar part of T-S2 - Define measuring objectives.

⟨Accuracy⟩ Sets the minimum required accuracy, e.g., ±1% (↗KB p. 258), extracted from parameter T-A2:O2. In other words, the variable specifies the maximum acceptable difference between the "real" value and the measured value, called uncertainty, each measurement is afflicted with (cf. Section 5.5). Importantly, measurements coming without a specified accuracy are meaningless [377].

⟨Components⟩ Entitles the IT infrastructure components, measurements should describe. The component set is implicitly given by T-A2:O3, because the calling actions dictate, which components to consider.

⟨Setup⟩ References the applied measuring setup, as reproducible and stable results require a clear and explicit measuring setup.

The following sections provide the production part of sentence template T-S2. Noteworthy, only variable T-S2:⟨Setup⟩ is further processed, the three other variables guide this processing.

T-A2:A2 – Select measuring instrument

Mapping a finite set of characteristics on a symbol set requires a *measuring instrument*. For this, DIN 1319 [35] defines in an abstract blueprint a *sensor*, an *amplifier/modifier*, and a *filter/correction* (↗KB p. 256), each describing a role that can be implemented or realized in many ways (*EG-5.13*).

Action T-A2:A2 selects a suitable implementation of the aforementioned roles compliant to the measuring objectives defined in T-S2. Therefore, the action uses decision template T-D2 – Select measuring instrument that formalizes the selection process and consists of three elements:

Implication table Lists measuring instrument criteria that might affect other actions, as depicted in Template element 5.3.

Decision tool Itemizes criteria for choosing an appropriate measuring instrument. For each criteria, it provides a short explanation, defines the aspired value direction (depicted as an arrow), and optional limits (depicted as a cross), respectively, as depicted in Template element 5.4.

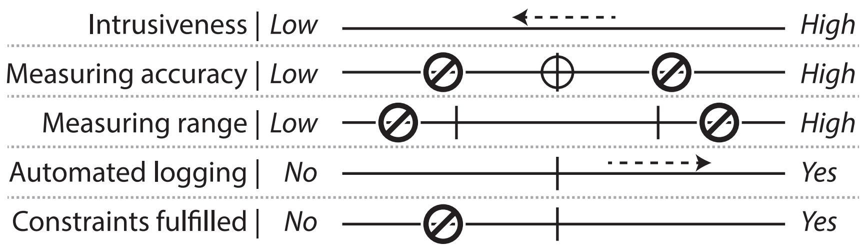
Documentation Documents the made decisions regarding the measuring instrument. Is contained in the `measuring` package of the provenance information model and described Section 5.5.2.

The voltage "sensor" in a Multimeter is a physical item. In contrast, the power consumption "sensors" Intel introduced with the Sandy Bridge chip are *Running Average Power Limit* (RAPL) counters, i.e., a small set of registers that count events [73, 175].

EG-5.13

Opt.	Explanation and implications
I1	**Intrusiveness** – Level of instrumentation required by the measuring instrument (↗KB p. 258). ↦ (High) intrusiveness potentially alters the measurement values. This might negatively impact the application of statistical methods for model proxy function creation in action T-A3:A13, because the actually sampled population might be different to the target population [120].
I2	**Accuracy** – Uncertainty of the measurements gained by the measuring instrument (set in T-S2:⟨Accuracy⟩). ↦ The poorer the accuracy (or the higher the uncertainty), the lower the gained measurements' quality. This might negatively impact statistical methods for model proxy function creation in action T-A3:A13.
I3	**(In)direct measurements** – A measuring instrument can gain targeted measurements either directly or provide only helper values that are processed to the targeted measurements, e.g., calculate *watt* out of *volt* and *ampere*. ↦ Employing helper values requires additional processing that might decrease accuracy. In any case, it causes additional effort in actions T-A3:A13 and T-A3:A16.
I4	**Automated logging** – Data mining and (statistical) post processing require logging and persistent storage of the collected measurements. ↦ Manual logging is error-prone, what might negatively impact the model proxy's accuracy in action T-A3:A13.

Template element 5.3: Implication table of T-D2 - Select measuring instrument.



Intrusiveness Describes the level or severity of instrumentation (↗KB p. 258), which should be preferably low [431].

Measuring accuracy Numeralizes the difference between the "real" and the measured value. The measuring instrument's accuracy should match the value given in T-S2:⟨Accuracy⟩, as a lower value is not suitable, and a higher value tends to cause dispensable cost, because the achieved accuracy increase is not exploited by the reasoning project. Instruments without an explicit accuracy specification shouldn't be used at all.

Measuring range States whether the instrument's measuring range covers the pursued value(s), e.g., Volt between 10 and 100V.

Automated logging States whether the measuring instrument supports automatic logging, e.g., storing measurements over time on a SD card. This capability is not mandatory, but very beneficial in terms of error-avoidance, effectiveness, and automation.

Constraints fulfilled States whether the measuring instrument fulfills criteria and constraints specified by the respective reasoning suite T-A3:O3, like license fee limits.

Template element 5.4: Decision tool of template T-D2 - Select measuring instrument.

T-A2:A3 – Design measuring system

Action T-A2:A3 designs the measuring system in terms of involved components, calibrating facilities, and any other equipment. In particular, the action plans where to place the measuring instruments, how to connect them to controlling and storing devices, and where to place potentially required further control instances. This design action is geared to the policies provided by DIN 1319 (↗KB p. 258). The variety of situations and the resulting diversity of possible measuring setups render a generic design presetting unproductive. In contrast, action T-A2:A3 uses checklist template T-C2 - Examine measuring setup design to evaluate a measuring setup's shape. Template element 5.5 depicts checklist template T-C2 that itemizes a rule set each measuring setup should comply to.

- □ **Reduce intrusiveness** – The basic concepts of intrusiveness (↗KB p. 258) are not only applicable to a single measuring instrument, but also to a whole measuring setup, e.g., addressing the logging-caused load that might influence the measuring. Although it is rarely possible to assemble a completely non-intrusive setup, there are several reduction approaches, e.g., use existing or already incorporated sensing capabilities, like operating system performance metrics or hardware performance counters [132].
- □ **Omit influencing factors** – The (undetected) influence to measuring mostly results in misleading measurements. Just as complete un-intrusiveness, omitting all influencing factors is rarely possible. One approach is to isolate considered elements, e.g., turn of LCDs when measuring power consumption of a built-in CPU to omit all power consumption that is not related to the CPU [104].
- □ **Comply to standards** – One of the most important measuring standards is DIN 1319, detailing the components of a measuring setup and requirements (↗KB p. 258).
- □ **Fulfill measuring objectives** – Before documenting the measuring setup and employing it for measuring execution, a final check should ensure that the measuring setup is compliant to the measuring objectives defined in sentence template T-S2.

Template element 5.5: Checklist of T-C2 - Examine measuring setup design.

T-A2:A4 – Document measuring system

Action T-A2:A4 documents the measuring activity, covering the measurement procedure, equipment, and influencing quantities [35]. The action uses form template T-F2 - Document measuring setup that instructs measuring setup documentation as depicted in Template element 5.6.

F1 – Unique ID

Filling objectives and rules – Uniquely identify the measuring setup for referencing, especially in T-S2:⟨Setup⟩.

F2 – Measuring information model

Valid data – UML object diagram
Justification – Documentation extends the provenance information model, which is also provided as UML object diagram (cf. Section 5.3).
Filling objectives and rules – Document static information about the measuring setup, particularly employed measuring instruments, selection causes, and measuring setup parameters.

F3 – Physical design

Valid data – UML component diagram
Justification – An UML component diagram describes the physical structure and realizing components of a system [344], while focusing on technical components and aspects that are required during run time, instead on internals of executed methods and processes [344] (↗KB p. 273). This orientation recommends UML component diagrams for measuring setup documentation.
Filling objectives and rules – Document the measuring setup in terms of self-contained functional components, their physical composition, and interplay. Use the instances of the object diagram in field T-F2:F2. In either case, the component diagram should contain the elements stated by DIN 1319 (↗KB p. 258) and provide implementing manifestations, accordingly.

Template element 5.6: Form of T-F2 - Document measuring setup.

T-A2:A5 – Execute measurements

Action T-A2:A5 represents measurement execution, covering the measuring system's employment, and the evaluation and documentation of gained results [35]. Documentation uses the provenance information model described in the following Section 5.5.2. The action does not provide any concrete templates, because the variety of potential measuring instruments, measuring systems, and required evaluation details render a concrete template unsuitable. Instead, the action acts as an umbrella for measurement execution to enable its explicit positioning in activity template T-A2.

5.5.2 Provenance

Figure 5.9 depicts the provenance information model's `measuring` package that provides classifiers for documenting a measuring activity. The contained sub packages group the subsequent classifier description:

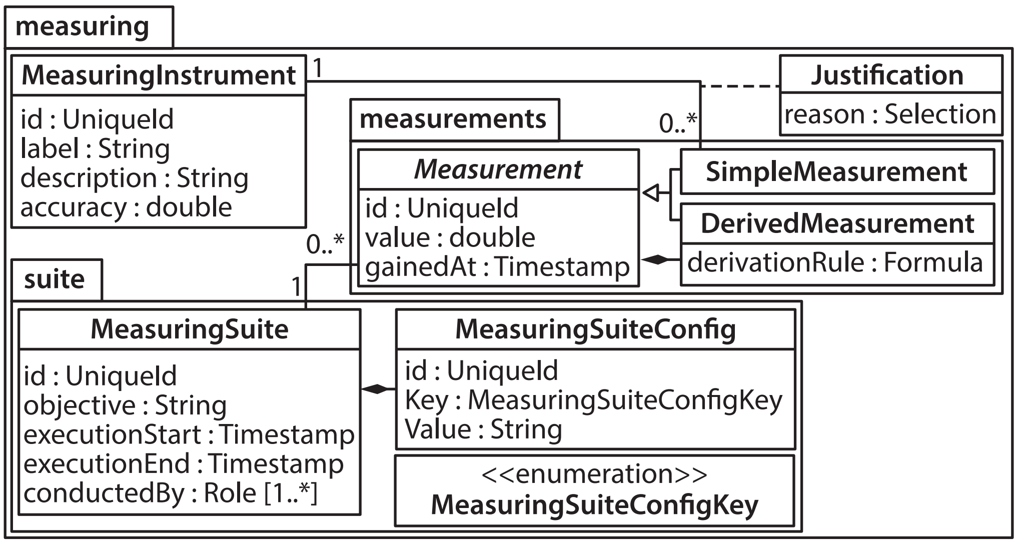


Figure 5.9: The provenance information model's `measuring` package, grouping classifiers that describe measuring execution and achieved results.

The package's classifiers are detailed by the subsequent itemization.

`suite` Classifiers that document information about the measuring activity.

`MeasuringSuite` A measuring suite representative (cf. Section 5.5.1) that acts as reference point for other provenance information model classifiers. The measuring suite's goals and intents are summarized in the `objective` field, the measuring suite's execution time in the fields `executionStart` and `executionEnd`, and the set of roles responsible for the measuring suite in the `conductedBy` list. The class does not store variables of sentence template T-S2, as they are covered by other classifiers below.

`MeasuringSuiteConfig` A parameter specified for the measuring suite, e.g., a binary flag stating whether measurements were compiled automatically. Storing every parameter complies with the call for reproducibility (cf. Section 5.5), especially for derived measurements represented by the `DerivedMeasurement` class, because it ensures that all n measurements of a series are "obtained under repeatability conditions" [35, p. 8]. The `key` identifies the parameter, using values provided by the `MeasuringSuiteConfigKey` enumeration to avoid inconsistencies due to differing keys for the same parameter.

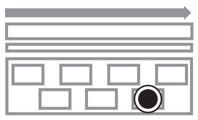
`measurements` Classifiers that document achieved measurements.

`Measurement` A measurement (↗KB p. 256) action T-A2:A5 gained in the context of the associated `MeasuringSuite`. The class is marked abstract, because a measurement can exclusively be either a simple measurement or a derived measurement. Thus, the class collects common attributes of both, namely the `value` and the point in time `GainedAt` to document measuring over time [35].

`SimpleMeasurement` A simple measurement, directly gained by a measuring instrument. To support reproducibility in general and to provide "information regarding the accuracy of the measurement" [35, p. 13] in particular, the class is associated to the `MeasuringInstrument`, the selection rationale is stored in the association class `Justification`.

`DerivedMeasurement` A derived measurement that is either directly assembled by other measurement values, or indirectly determined by means of known physical or mathematical relationships [35]. The former approach additionally models the associated values as composition between the `DerivedMeasurement` and `Measurement` classes. For both approaches, the `derivationRule` is stored as `Formula` to support traceability. Employing the composite pattern – `SimpleMeasurement` is a leaf element and `DerivedMeasurement` is a composite element – allows arbitrary derivation hierarchies.

`MeasuringInstrument` A measuring instrument selected in action T-A2:A2. The measuring instrument's capability is summarized in the `label` field, detailed information about the measuring instrument, like a manual or objectives, is stored in the `description` field. The `accuracy` field addresses the urgent need of providing measurement accuracy.



→ *Fig. 5.1 (p. 98)*

5.6 IT infrastructure attribute notion

Section 5.4.4 introduces IT infrastructure attributes as a quantitative description of an IT infrastructure that considers its components as black boxes during workload execution at a particular point in time. This section enhances the general understanding for its employment in the reasoning methodology described in Chapter 6. For the sake of simplicity, IT infrastructure attributes are from now on simply called *attributes* if collisions are precluded. Attribute decomposition is detailed in Section 5.6.1, the binding of attributes to IT infrastructure components is discussed in Section 5.6.2.

5.6.1 Decomposition in concept and instances

An attribute is decomposed in a *generic concept* and at least one *instance* of this concept.

In terms of software programming, the **generic concept** can be seen as the attribute *interface*. It defines the concept's name (*interface name*), scale (*interface return type*), and an optional parameter set (*interface parameters*, *EG-5.14*). Section 6.1.3 details the attribute concept definition in action T-A3:A3, and relevant provenance information model parts.

EG-5.14

> An exemplary attribute concept is "power consumption (*name*) in Watt per time step (*scale*) for a given load factor (*parameters*)" [383].

The generic concept is realized by at least one **instance**. Applying again the perspective of software programming, a concept instance can be regarded as the *(interface) implementation* that concretely describes how an attribute value (*return value*) is computed, e.g., using common mathematical equations, existing models, prepared regressions, or any other solution that might be suitable for the current scenario. Section 6.2.3 explains the selection and integration of existing attribute instances in action T-A3:A12, Section 6.2.4 describes the implementation of attribute instances in the form of model proxy functions in action T-A3:A13.

IT infrastructure attribute decomposition has three benefits:

- **Facilitate top-down refinement** The iterative function refinement applied in the reasoning methodology (cf. Section 4.2) is likely to require the analysis of the same attribute on several granularity levels for a variety of IT infrastructure component types. Depending on the reasoning objectives, considered component types, and the varying quality of data, each potential attribute instance has its specific (dis)advantages [6] (*EG-5.15*). Besides, employing the justification for the unsuitability of "one-size-fits-all" modeling (cf. Section 4.1) clarifies that it is rarely possible to use the same attribute implementation for all refinements. In contrast, attribute decomposition enables the specification of a global understanding (*concept*) in the reasoning objectives, and the employment of a particular implementation (*instance*) that is most suitable for the current refinement.

  When reasoning about probability distribution based availability [6] (*concept*), each distribution has specific advantages depending on the objective, the existing empirical observation, and the considered component type. For instance, Pareto (*instance*) is often advocated for lifetime estimation, whereas Weibull (*instance*) is more suitable for various resource availability data [422, 298]. *EG-5.15*

- **Support delegation** Covering several IT infrastructure components or a whole IT infrastructure according to NFR-1 implicitly requires the involvement of multiple areas of expertise and roles (cf. Section 5.1). Attribute decomposition allows the delegation of tasks to appropriate roles: the management role R5, responsible for the reasoning project, can define the attribute concept, the attribute domain experts R3 can (independently) provide the attribute instances according to the demands formalized in the attribute concept. This approach directly fosters the employment of expert knowledge for each refinement, since the attribute instance for a concrete reasoning function refinement can be executed by the most skilled role or person.

- **Support measuring** Measuring theory requires the definition of a measurand being independent from the applied measuring activity [80] (*EG-5.16*). Attribute decomposition fulfills this need, because the measurand is defined once in the attribute concept and gathered in several ways using appropriate measuring activities for attribute instance implementation.

EG-5.16 Measuring the distance between two points (*concept*) can be achieved by using a metering rule (*instance*), or by applying triangulation (*instance*), which both result in the same distance concept [65]. Thus, the distance *concept* can be applied on molecules but also on the Milky Way, each using a different *instance* to gain the concrete values.

5.6.2 Attribute binding types

Depending on the reasoning project objectives and the attribute's intent, its concept and instances are defined for the same or for different granularity levels and component types regarding the IT infrastructure (*EG-5.17*) [47], as subsequently explained:

- **Same granularity level** The attribute concept and its instance(s) must be defined on the same granularity level. This is particularly required for attributes from a consumer perspective, as they use the whole IT infrastructure or a delimited subset that provide their functionality by a component interplay.

- **Multiple granularity levels** The attribute concept and its instance(s) can be defined on several granularity levels and for multiple component types. This is particularly the case for attributes form a provider perspective, as each component can get a dedicated attribute instance.

EG-5.17 An attribute requiring the same granularity level is *Time to Completion* (TTC, cf. Section 2.4.2), because the TTC of a scientific application is only meaningful for the executing IT infrastructure. Also for component sub sets, this is valid, e.g., although TTC of the *Sustainable Memory Bandwidth in High Performance Computers* (STREAM) benchmark focuses on memory, its value strongly depends on the interplay with other components like the CPU. An attribute supporting multiple granularity levels is power consumption (cf. Section 2.4.1), because the concept is defined once – "power consumption in Watt per time step" – and its instances bound to the IT infrastructure on multiple granularity levels, e.g., for an Intel PXA255 processor [104], or an entire data center [138].

5.7 Application proxy workload selection

→ *Fig. 5.1 (p. 98)*

“Production” applications (cf. Section 2.3.2) directly expose the behavior of the day-by-day IT infrastructure use. This recommends them as workload to use when executing reasoning and particularly for load generation. Nevertheless, it is not always possible to run (production) applications on each machine in question [407]. This situation calls for a workload that substitutes or proxies the application, especially for the creation of a model proxy function in action T-A3:A13, and for model input parameter gain in action T-A3:A18, since both actions rely on load values for IT infrastructure components. The plurality of potential benchmark candidates, produced by the four layer based concept of benchmark implementation (cf. Section 2.3.2), and the countless set of scientific applications render the obvious choice of a proxy workload a futile endeavor. Instead, a thorough selection is required. Even though this situation seems to call for a decision template, two reasons recommend a checklist template:

- **Comparability** The most important capability of a proxy workload, i.e., representing the substituted application appropriately, cannot be compared, because the definition of *appropriate* is highly situation specific. In addition, there is no generic “better”, like for measuring instrument intrusiveness (cf. Template element 5.4). Instead, the substitution’s fashion and particularly its compliance to the reasoning project objectives must be checked. Besides, the more accurate the workload mimics the substituted application, the higher the risk that it cannot be executed for the same reasons that prohibit the origin application’s execution.
- **Result mapping** Application substitution requires the mapping of results gained with the proxy workload on results (theoretically) gained with the production application. Even though workload classification approaches address this challenge [360], the mapping still causes unavoidable bias. Parametric benchmarks do not require the mapping and cover a broad range of application behaviors [360], but their big parameter set, like of the file system bandwidth testing code *IOR* [180], makes their use very complex.

Checklist template T-C3 - Examine application proxy selection in Template element 5.7 itemizes the features of a proxy workload that must be checked.

- ☐ **Instruction mix** – Each instruction (type) (cf. Section 2.3.2) exhibits a specific behavior and represents a different type of workload, like matrix multiplications or string comparisons, that potentially produce differing load values. An appropriate representation requires the workload to employ the same instruction mix or to expose the same load behavior as the substituted application.
- ☐ **Locality** – Code element placement in cache and instruction registers as well as the respective sizes might significantly influence the workload execution behavior. For instance, the nine loops of the Whetstone benchmark (cf. Section 2.3.2) are very small what causes an extremely high code locality and cache hit rate, even for small instruction caches [407]. Also simple code modifications, like source code reordering, potentially alter the execution behavior [407]. Consequently, the proxy workload should manifest the same locality as the substituted application.
- ☐ **Complexity** – Contemporary compilers provide manifold options, especially for optimization. Avoiding the replacement of code segments and in turn exhibiting a different workload behavior requires the proxy workload to manifest the same code complexity from a compiler's point of view as the substituted application.
- ☐ **IT infrastructure components** – The possibility to bind attributes on multiple granularity levels and component types (cf. Section 5.6.2) requires a proxy workload to stress (at least) the same elements as the substituted application.

Template element 5.7: Checklist of T-C3 - Examine application proxy selection.

5.8 Summary

The chapter at hand details and formalizes concepts, notions, and activities that collectively assemble the presented process model's "artifacts and procedures" part (cf. Figure 5.1), as the following itemization recapitulates:

- **Roles and actors** Abstract stakeholders in Section 5.1 to clarify responsibilities and to enable task assignments in a reasoning project.

- **Template meta model** Provides concepts and language elements in Section 5.2 for describing and formalizing process model elements.

- **Provenance information model** Provides classifiers in Section 5.3 for storing partial results and for tracing reasoning project execution in order to foster and ensure reproducibility.
- **IT infrastructure notion** Describes in Section 5.4 the process model underlying understanding of IT infrastructures.
- **Measuring** Formalizes in Section 5.5 the gathering of measurements, built of measuring objectives definition and measuring setup design.
- **IT infrastructure attribute notion** Describes in Section 5.6 the process model underlying understanding of IT infrastructure attributes.
- **Application proxy workload selection** Formalizes in Section 5.7 workload replacement selection if the original one cannot be executed.

The subsequent Chapter 6 describes the process model's reasoning methodology that heavily employs the elements provided in this chapter.

Chapter 6

Design cycle – Process model reasoning methodology

The chapter details the process model's reasoning methodology that employs the artifacts and procedures introduced in Chapter 5. Figure 6.1 highlights in its upper part the chapter's focus and the methodology's conceptual decomposition in three phases that *prepare*, *compile*, and *use* the individual reasoning function (cf. Section 4.1). The three phases reflect the three Use Case sub systems identified in Section 3.3.2, and realize the process model underlying design concepts (cf. Section 4.2), as subsequently illustrated:

- **Phase A – Reasoning suite definition** Identifies and specifies reasoning objectives and related information to facilitate the reasoning objectives driven refinement of the individual reasoning function (cf. Section 4.3). The phase assembles two results:
    - A so-called *reasoning suite* that formalizes gathered information to steer the reasoning project, and to dictate guidelines for (partial) result validation.
    - A *reasoning function skeleton* that is processed by *Phase B*.
- **Phase B – Reasoning function compilation** Directed by the reasoning suite, the phase iteratively refines the given skeleton to a fully functional reasoning function (cf. Section 4.2).
- **Phase C – Reasoning execution** Employs the given reasoning function to execute the reasoning, using differing reasoning tools, like optimization or What-if analysis.

Activity template T-A3 - Reasoning methodology formalizes the entire reasoning methodology and the introduced three phases. According to the research goal of providing a widely applicable process model (cf. Section 4.1), activity template T-A3 describes not only primary reasoning actions, but also secondary special cases, to cover a variety of specific situations, as well. As an unavoidable ancillary effect of this intent, activity template T-A3 exposes a high extend.

The chapter addresses this situation by dedicating a section to each of the three introduced phases: Sections 6.1, 6.2, and 6.3 detail *Phase A*, *Phase B*, and *Phase C*, respectively. Each section, in turn, provides the according action flow and action details of activity template T-A3. To ease referencing, section titles reflect action labels, e.g., "A6 – Select workload". Template element C.1 in Appendix C on page 314 depicts the fully detailed action flow, containing all elements of activity template T-A3.

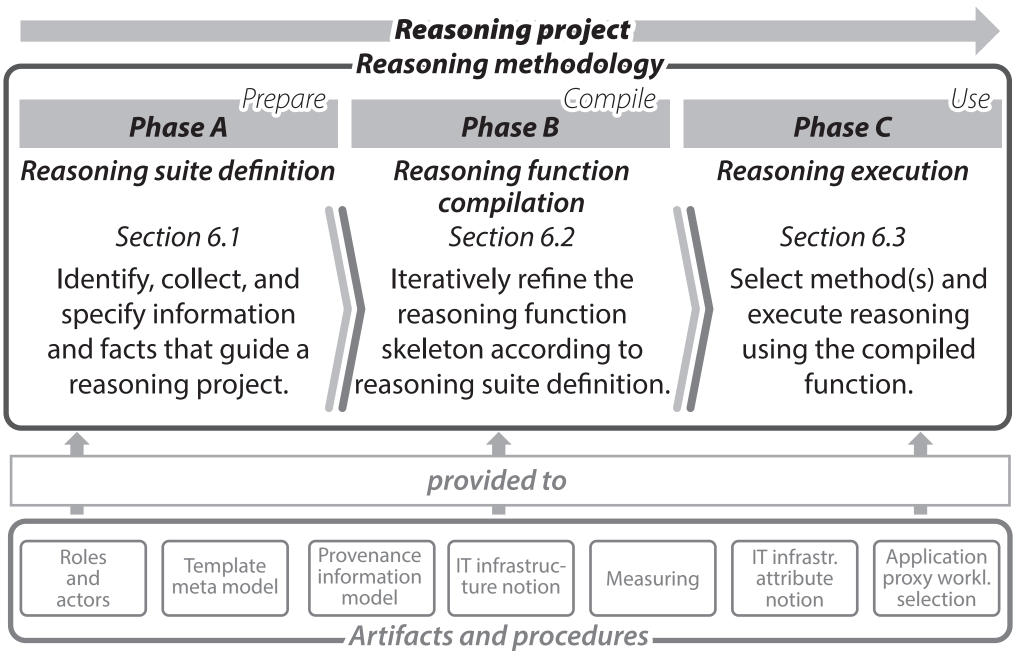
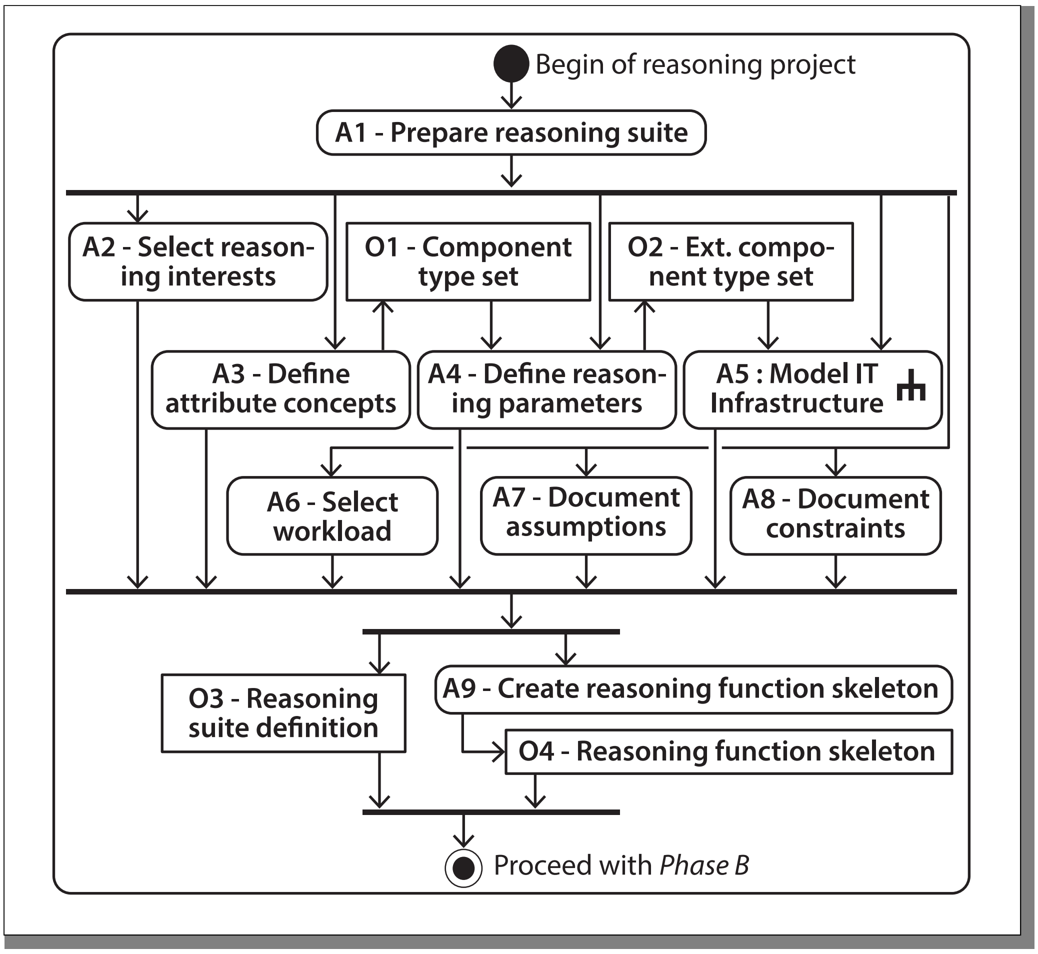

Reasoning project
Reasoning methodology
Prepare
Phase A
Reasoning suite definition
Section 6.1
Identify, collect, and specify information and facts that guide a reasoning project.
Compile
Phase B
Reasoning function compilation
Section 6.2
Iteratively refine the reasoning function skeleton according to reasoning suite definition.
Use
Phase C
Reasoning execution
Section 6.3
Select method(s) and execute reasoning using the compiled function.
provided to
Roles and actors
Template meta model
Provenance information model
IT infrastructure notion
Measuring
IT infrastr. attribute notion
Application proxy workl. selection
Artifacts and procedures

Figure 6.1: Overview of the developed process model emphasizing the focus of Chapter 6 on the contained reasoning methodology.

6.1 Phase A – Reasoning suite definition

→ Fig. 6.1 (p. 144)

In covering the introductory alluded high extend of a reasoning intent, activity template T-A3 consists of more than 20 elements (cf. Template element C.1 on page 314). Several of them are processed by differing roles and stakeholders (cf. Section 5.1) on multiple granularity levels and deal with different IT infrastructure components. This (potentially) causes much efforts, concurrent and decoupled execution, and partial results. *Phase A* of activity template T-A3 aims at addressing this situation by aligning actions, by synchronizing efforts, by merging compiled (partial) results, and by fostering reproducibility. Therefore, the phase executes nine actions that collectively identify and formalize information and parameters that guide a reasoning project and particularly single actions. The phase results in a *reasoning suite* (T-A3:O3), and a *reasoning function skeleton* (T-A3:O4).


Begin of reasoning project
A1 - Prepare reasoning suite
A2 - Select reasoning interests
O1 - Component type set
O2 - Ext. component type set
A3 - Define attribute concepts
A4 - Define reasoning parameters
A5 : Model IT Infrastructure
A6 - Select workload
A7 - Document assumptions
A8 - Document constraints
O3 - Reasoning suite definition
A9 - Create reasoning function skeleton
O4 - Reasoning function skeleton
Proceed with *Phase B*

Template element 6.1: Action flow of T-A3 - Reasoning methodology - Phase A.

Template element 6.1 depicts the action flow of *Phase A*. Some actions are executed concurrently, some sequentially, to address action result reciprocity, and reasoning project versatility. Besides, the process model's implementation concepts (cf. Section 4.1) order actions, pursuing a cost- and effort-efficient, goal-oriented, and focused reasoning as well as an objective-driven refinement. This results in four steps, defining

1. reasoning project interests and goals in action T-A3:A2,
2. the reasoning function co domain in action T-A3:A3 according to 1),
3. the reasoning function domain in action T-A3:A4 according to 1) and 2),
4. supporting elements in the remaining actions according to 1), 2), and 3).

The remainder of this section explains the action details of *Phase A*: action T-A3:A1 defines sentence template T-S3 that represents the introductory alluded reasoning suite; afterwards, actions T-A3:A2 to T-A3:A8 process concurrently variables of the sentence template; action T-A3:A9 finally deduces the reasoning function skeleton.

6.1.1 A1 – Prepare reasoning suite

→ T.E. 6.1 (p. 145)

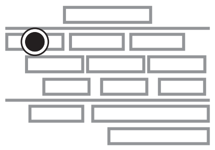
Sentence template T-S3 - Define reasoning suite formalizes and represents the reasoning suite that is assembled in *Phase A*. Template element 6.2 depicts the template's grammar part. Noteworthy, most variables are identifiers, as the corresponding information cannot be described in one word. Instead, the identifiers refer to the documentation parts of the employed templates.

Reason about ⟨Interest⟩ of ⟨AttributeConcepts⟩ in ⟨ITInfrastructure⟩, executing ⟨Workload⟩, applying ⟨Parameters⟩, assuming ⟨Assumptions⟩ [, providing ⟨Constraints⟩].

⟨Interest⟩ ⊨	[trend of] value | sample | delta	⇠ T-A3:A2
⟨AttributeConcepts⟩ ⊨	⟨GIdentifierSet⟩	⇠ T-A3:A3
⟨ITInfrastructure⟩ ⊨	⟨GIdentifier⟩	⇠ T-A3:A5
⟨Workload⟩ ⊨	⟨GIdentifierSet⟩	⇠ T-A3:A6
⟨Parameters⟩ ⊨	⟨GIdentifierSet⟩	⇠ T-A3:A4
⟨Assumptions⟩ ⊨	⟨GIdentifierSet⟩	⇠ T-A3:A7
⟨Constraints⟩ ⊨	⟨GIdentifierSet⟩	⇠ T-A3:A8

Template element 6.2: Grammar part of T-S3 - Define reasoning suite.

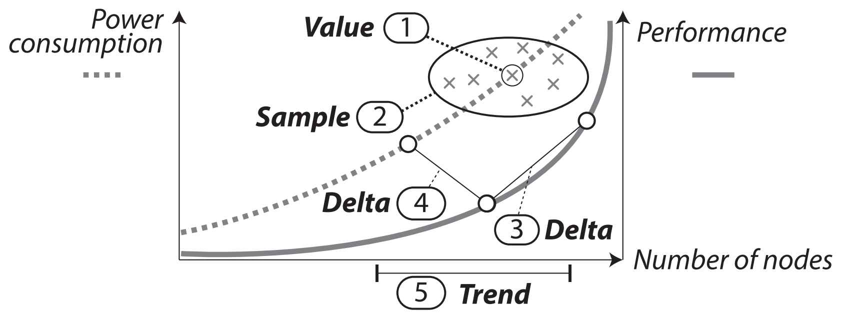
6.1.2 A2 – Select reasoning interests



→ *T.E. 6.1 (p. 145)*

Variable T-S3:⟨Interest⟩ specifies the general interest of a reasoning intent. In other words, it deals with the general meaning of an entry in the reasoning function result vector (cf. Equation 4.1), abstracting from attributes, IT infrastructure components, and workload. Variable T-S3:⟨Interest⟩ can take one of the five following options (*EG-6.1:1*):

- **1 – Value** A single value of an attribute concept for a particular modification parameter value.
- **2 – Sample** A population or sample of values (↗KB p. 268) in the context of statistical analysis.
- **3 – Delta** A delta within an attribute concept.
- **4 – Delta** A delta between two attribute concepts.
- **5 – Trend** A trend analysis, i.e., the development of a set of parameters or a period of time for the other four options.



EG-6.1

(1) The figure explains the options using a simplifying example: the fictitious behavior of an HPC cluster's power consumption (dashed) and performance (solid) is plotted against the cluster's number of nodes (cf. Section 2.2.2). *2* describes the power consumption distribution, *3* describes the performance delta related to the number of nodes, respectively. *(2)* Computing the performance delta for two amounts of nodes requires a metric scale for the performance attribute (↗KB p. 267), because a nominal or ordinal scale wouldn't allow this computation.

Each option puts individual (strong) requirements on the quantities and scales (*EG-6.1:2*) that action T-A3:A3 and T-A3:A4 set for the reasoning function's (co) domain. The reasons for this impact are twofold:

1. Each option requires a specific scale to achieve the targeted meaning. For instance, computing a delta mandatorily prerequisites a metric scale, as other scales do not allow the calculation of distances (↗KB p. 267).
2. Options that put values in correlation require all involved elements to use the same or at least comparable quantities, because a mapping of differently scaled values is mostly difficult or even impossible [80].

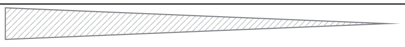
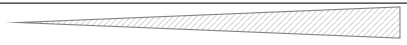
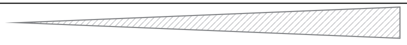
For the same reasons, option selection directly affects the set of (mathematical and statistical) operations reasoning can use in *Phase C*.

Thus, action T-A3:A2 uses decision template T-D3 – Select reasoning interests to select an option, and places the result in T-S3:⟨Interest⟩. T-D3 consists of three parts:

- **Implication table** Highlights implications on quantity and scale of the reasoning function (co) domain. Template element 6.3 depicts the implication table.
- **Decision tool** Deals with scale types (↗KB p. 267). In particular, it compares their usage characteristics, like ease of creation, and considers the (non) applicability of statistical tools (↗KB p. 268), indicated by ✓ and ×. Template element 6.4 depicts the decision tool.
- **Documentation** Documents the selected option and decision reasons. Noteworthy, documentation is not accomplished by action T-A3:A2. Instead, action T-A3:A3 and T-A3:A4 document the decision in the context of scale and quantity selection as discussed above.

Opt.	Explanation and implications
I1	**Value** – Reasoning considers a value without any processing. ↦ Since no (mathematical) operations are performed on attribute concept values, action T-A3:A3 can select attribute concept quantities and scales freely.
I2	**Sample** – Reasoning considers a population. ↦ Associating domain and co domain values using correlation methods, like Regression (↗KB p. 268), requires setting a cardinal (co) domain in action T-A3:A3 and T-A3:A4. ↦ (Mathematical) optimization (↗KB p. 262) in action T-A3:A19 requires metric domain values.
I3	**Delta** – Reasoning considers the delta between attribute values for differing reasoning parameter values. ↦ Requires action T-A3:A3 of selecting the same or at least comparable quantities and metric scales [375] (↗KB p. 267) for all involved attribute concepts, because value mapping from differing scales tends to be difficult or even impossible [80]. In any case, it might require a preparatory processing, e.g., "translating" the binary availability of yes/no to 0 and 100 on a percentage metric scale. ↦ Requires action T-A3:A19 to conduct at least two reasoning runs, one for each delta operand, respectively.
I4	**Trend** – Reasoning performs a trend analysis (↗KB p. 268) about a reasoning modification or configuration parameter. ↦ Applying arithmetic operations on the reasoning parameter(s) requires a metric scale in action T-A3:A4 [375]. ↦ A trend analysis of development over time might require action T-A3:A19 to execute for each time step a particular optimization or What-if analysis run, and to store a lot more provenance values. Although trend analysis computes a bigger data base that might provide more insights, it also causes higher cost. ↦ Requires action T-A3:A19 to conduct several reasoning runs, one for each trend step, respectively.

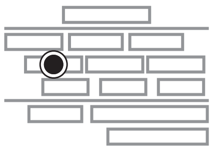
Template element 6.3: Implication table of T-D3 - Select reasoning interests.

	Scale type		
	Nominal	Ordinal	Metric
Equality	✓	✓	✓
Frequency (abs., rel., perc.)	✓	✓	✓
Modus	✓	✓	✓
Accumulated frequency (abs., rel.)	×1)	✓	✓
Percentile	×1)	✓	✓
Histogram	×2)	×2)	✓
Interpretation of distance	×3)	×3)	✓
Means	×4)	×4)	✓
Ease of creation5)			
Expressiveness6)			
Employment7)			

1. Requires ordering, which nominal scales do not provide.
2. Requires distances to calculate the width of Histogram bars, which nominal and ordinal scales do not provide.
3. Requires distances, which nominal and ordinal scales do not provide.
4. Most mean types require arithmetic operations and distances, which nominal and ordinal scales do not provide.
5. A scale's capability is correlated to its difficulty of creation, since each capability poses additional demand(s) on the scale [375], e.g., a nominal scale requires only a (simple) assignment of objects to a qualitative category, a metric scale requires also an (enhanced) distance function.
6. A scale's expressiveness is correlated to its (implicitly) provided information, e.g., an ordinal scale describes a hierarchy between categories and hence, is more expressive than a nominal scale.
7. The metric scale is likely the be the most frequently used one in a reasoning project. This is of special importance in action T-A3:A12 that selects an existing model for integration: if a model integration candidate's scale doesn't match the scales of the already integrated models, it is extremely difficult or impossible to map a nominal or ordinal scale on a metric scale.

Template element 6.4: Decision tool of template T-D3 - Select reasoning interests.

6.1.3 A3 – Define attribute concepts



→ T.E. 6.1 (p. 145)

Attribute decomposition allows the distributed implementation of attribute concept instances by differing stakeholders to employ expert knowledge where appropriate (cf. Section 5.6). This decoupled implementation urgently prerequisites that all involved stakeholders apply (exactly) the same concept notion, and that applied (implementation) methods result in the same values [80]. If prerequisites are not fulfilled, (semantic) value mismatches are likely to produce misleading or even wrong reasoning results (*EG-6.2*).

An obvious mismatch is the differing description of availability in *yes/no* (binary nominal scale) and *probability* (metric scale). Mismatches more difficult to identify describe energy efficiency in *Cycles/Watt* and in *FLOP/Watt*, or performance in *FLOP/s* and in *Byte/s*: both descriptions produce a natural number, but expose fundamentally differing meanings.

EG-6.2

Thus, action T-A3:A3 formalizes an attribute concept in sentence template T-S4 - Define attribute concept to dictate attribute concept information, to foster a common understanding, and to guide the (distributed) attribute instance implementation in action T-A3:A12 and T-A3:A13. Template element 6.5 depicts the grammar part of T-S4, the subsequent itemization details its production part. The result of action T-A3:A3 becomes part of the reasoning suite by adding T-S4:⟨Identifier⟩ to T-S3:⟨AttributeConcepts⟩.

The attribute concept ⟨Identifier⟩ describes ⟨Objectives⟩ in ⟨Quantity⟩ of ⟨ComponentTypes⟩ provided that ⟨Constraints⟩.

⟨Identifier⟩ ⊨ ⟨GIdentifier⟩		⇠ T-A3:A3
⟨Objectives⟩ ⊨ [⟨UtilFunc⟩][⟨Range⟩]String		
⟨UtilFunc⟩ ⊨ A utility function		⇠ T-D4
⟨Range⟩ ⊨ ⟨GRange⟩		⇠ T-D4
⟨Quantity⟩ ⊨ ⟨GQuantity⟩at⟨Accuracy⟩		⇠ T-D5
⟨Accuracy⟩ ⊨ ⟨GPercent⟩		⇠ T-D5
⟨ComponentTypes⟩ ⊨ ⟨GIdentifierSet⟩		⇠ T-D6
⟨Constraints⟩ ⊨ ⟨GIdentifierSet⟩		⇠ T-F3

Template element 6.5: Grammar part of T-S4 - Define attribute concept.

⟨**Identifier**⟩ Identifies the attribute concept within the reasoning project, and is added to the identifier set in T-S3:⟨AttributeConcepts⟩.

⟨**Objectives**⟩ Describes the attribute concept goal in a non-formal, textual way. It enhances the mostly pure numeric ⟨Quantity⟩ (cf. below) to reduce mismatch risk, e.g., two instances apply the same quantity but differing meanings (EG-6.2), a situation the ⟨Quantity⟩ does not cover. The optional ⟨UtilityFunction⟩ and ⟨Range⟩ elements specify an utility function and a variable range for the attribute concept, respectively. Decision template T-D4 introduced below guides value setting.

⟨**Quantity**⟩ Describes the attribute concept goal in a formal, mostly numeric way by specifying a regular quantity (cf. Section 5.2.2), and the required accuracy. The latter must be set individually for a particular reasoning project, as accuracy appropriateness depends on the reasoning and attribute concept objectives (*EG-6.3*). Both must comply to the constraints specified in action T-A3:A2. Decision template T-D5 introduced below guides value setting.

EG-6.3

Bhatia et al. [45], talk about "low" error rates of <30%, Carrington et al. [84] conceive an error range from 1-15% as good "for simple models", and Davis et al. [112] label a model having an average error of 4.8% as "very good".

⟨**ComponentTypes**⟩ Lists component types the attribute concept can be bound to (cf. Section 5.6.2). The list is not fully modeled, but acts as input T-A3:O1 for action T-A3:A4 (cf. Template element 6.1). Decision template T-D6 introduced below guides value setting.

⟨**Constraints**⟩ Describes (environment) factors that might influence attribute concept values and hence, must be covered (*EG-6.4*). The potential extend of a constraint is covered by form template T-F3 introduced below. Its field T-F3:F1 is added to the identifier set in T-S4:⟨Constraints⟩.

EG-6.4

An exemplary constraint for the power consumption attribute concept could require considering the PowerNap [277] technology, and excluding *Dynamic Frequency Voltage Scaling* (DFVS) [371]. This affects the selection of model integration candidates and requires a deactivation of DFVS for measuring.

Select attribute concept objectives

Decision template T-D4 – Select attribute concept objectives processes variable T-S4:⟨Objectives⟩, particularly the utility function and the range. Template element 6.6 depicts its implication table, the documentation part is taken by the respective provenance information model packages described below.

Opt.	Explanation and implications
I1	**Utility function** – Defines the aspired and targeted value (range) of the attribute concept (*EG-6.5*). ↦ Is mandatory for selecting and using optimization algorithms in action T-A3:A17 and T-A3:A19, respectively. ↦ Utility function definition tends to be a challenging task, as it implicitly defines a lot of situation specific semantics [9]. Hence, it should only be defined if it is about to be used in action T-A3:A19 to save efforts.
I2	**Range** – Defines the relevant attribute concept value range. ↦ The concrete range affects action T-A3:A13 and T-A3:A16: both might gain measurements using activity template T-A2. Thus, the measuring instrument selected in action T-A2:A2 must compulsory support the defined range, since otherwise an appropriate measuring is not possible. ↦ Usually, the value range's extend is not only related to the information gain, but also to the induced costs for reasoning execution in action T-A3:A19, because the bigger the range, the more runs for an optimization or What-if analysis might be required.

Template element 6.6: Implication table of T-D4 - Select attribute concept objectives.

For the performance attribute concept *Time To Completion* (TTC) a utility function could formalize that "lower is better, as users [...] prefer short response times" [239, p. 202]

EG-6.5

Select attribute concept quantity

Decision template T-D5 – Select attribute concept quantity processes variable T-S4:⟨Quantity⟩. Template element 6.7 depicts its implication table, the documentation part is taken by the respective provenance information model packages described below.

Opt.	Explanation and implications
I1	**Quantity and scale** – Describes the attribute concept's goal in a formal, mostly numeric way. A quantity can be *primary*, *additive*, and *derived* (↗KB p. 263), a scale structures or splits a (co) domain in categories or blocks, respectively (↗KB p. 267). ↦ The more common quantity and scale are, the more likely it is to find an appropriate model integration candidate in action T-A3:A12. ↦ If no suitable model integration candidate exists, model proxy creation in action T-A3:A13 might conduct measuring, which should be feasible for the selected quantity and scale.
I2	**Accuracy** – Defines the maximum acceptable uncertainty of all attribute concept instances. ↦ The accuracy level affects action T-A3:A13 and T-A3:A16: both might gain measurements using activity template T-A2. Thus, the measuring instrument selected in action T-A2:A2 must compulsory support the defined accuracy level, since otherwise an appropriate measuring is not possible. Besides, the accuracy level and the difficulty of measuring are directly related (cf. Section 5.5). ↦ Normally, the accuracy level influences the effectiveness of an optimization intent [47] in action T-A3:A19.

Template element 6.7: Implication table of T-D5 - Select attribute concept quantity.

Select IT infrastructure component types

Decision template T-D6 – Select IT infrastructure component types processes variable T-S4:⟨ComponentTypes⟩. Template element 6.8 depicts its implication table, its documentation part is taken by the respective provenance information model packages described below. The resulting set of component types is stored in T-A3:O1.

Opt.	Explanation and implications
I1	**Source of component type set** – Describes the sources of the type set. ↦ If there is a database storing information regarding the considered IT infrastructure, action T-A4:A2 should extract and import component types to reduce (modeling) costs.
I2	**Component type set** – Describes the IT infrastructure component types the attribute concept is to be considered for. ↦ The component type set should be preferably small, because all contained IT infrastructure component types, that are potentially further extended in action T-A3:A4, must be modeled in action T-A3:A5. Hence, the bigger the set, the higher might modeling costs be. ↦ The component type set should be preferably small, because potentially for every set entry, action T-A3:A12 must select a model integration candidate, or action T-A3:A13 must create a model proxy function. Hence, the bigger the set, the higher might (modeling) costs be. ↦ The extend of selected IT infrastructure component types might constrain reasoning options in action T-A3:A19, since action T-A3:A4 can define modification parameters only for provided component types. Thus, the smaller the component type set, the less modification parameters might be available. ↦ The iterative reasoning function refinement in *Phase B* and particularly the definition of iteration objectives in action T-A3:A10 are guided by the selected component type hierarchy.

Template element 6.8: Implication table of T-D6 - Select IT infrastructure component types.

Document attribute concept constraints

Form template T-F3 - Document attribute concept constraints documents a single constraint that is of importance for an attribute concept, e.g., (not) to consider a certain functionality or configuration parameter (EG-6.4 on page 152). Field T-F3:F1 is added to the identifier set in T-S4:⟨Constraints⟩. Template element 6.9 depicts the form fields of T-F3.

F1 – Unique ID

Filling objectives and rules – Uniquely identify the form template instance for referencing, especially in T-S4:⟨Constraints⟩.

F2 – Constraint

Filling objectives and rules – Explicitly state a particular configuration, prerequisite, or aspect that influences the attribute concept, e.g., (de) activating or (not) omitting a specific technology. The description should by preferably clear and detailed to address the diversity of involved stakeholders and to foster applying the same setting within a reasoning project. Constraint formulation must avoid conflicts with global reasoning project constraints specified in action T-A3:A8.

Template element 6.9: Form of T-F3 - Document attribute concept constraints.

Attribute concept provenance

Provenance of attribute concepts and particularly of sentence template T-S4 is realized by the provenance information model package `attributes`. Figure 6.2 depicts its structure that reflects IT infrastructure attribute decomposition (cf. Section 5.6), and the twofold attribute instance implementation by selecting an existing model (action T-A3:A12) or creating a model proxy (action T-A3:A13). Classifiers are described below.

`AttributeConcept` An attribute concept according to the notion introduced in Section 5.6. The class reflects sentence template T-S4 as follows: the `objectives`, `utilFunc`, and `range` fields use classifiers of the `datatype` package (cf. Section 5.3.2) to represent variable T-S4:⟨Objectives⟩; the fields `quantity` and `minAccuracy` represent variable T-S4:⟨Quantity⟩; the multiplicity of field `constraints` reflects the optional character of variable T-S4:⟨Constraints⟩; the class `AttributeInstance` covers variable T-S4:⟨ComponentTypes⟩ as described below.

`AttributeInstance` An attribute instance according the notion introduced in Section 5.6. A composition association (↗KB p. 271) to the realized `AttributeConcept` underpins the attribute decomposition (cf. Section 5.6.1), as an attribute instance is meaningless without the attribute concept it implements. The `label` field summarizes the attribute instance, the `description` field outlines the applied realization approach. The class is marked abstract to emphasize that an attribute instance is either an existing model (selected in action T-A3:A12), or a model proxy function (created in action T-A3:A13). The two sub classes `ExistingModel` and `ModelProxyFunction` described below address this distinction. For both, the attribute value calculating formula is stored in the `implementation` field, being of `Formula` type provided by the `datatype` package (cf. Section 5.3.2). The `accuracy` field documents the achieved accuracy of the attribute instance, which has to be equal or above the `minAccuracy` value of the associated `AttributeConcept` object. The field `boundTo` covers variable T-S4:⟨ComponentTypes⟩ and references all IT infrastructure components and types the attribute instance is bound to (cf. Section 5.6.2).

`ExistingModel` An existing model selected in action T-A3:A12. The `reasons` field documents selection reasons, being of the type `Selection` from the `datatype` package (cf. Section 5.3.2).

`ModelProxyFunction` A function substituting a model in case no existing model is suitable as attribute concept instance. The class is marked abstract to emphasize that action T-A3:A13 can create a model proxy based on measuring or based on aggregation.

`MeasuredProxy` An attribute instance derived from measurements. This means that the formula, which is stored in the super class field `implementation`, is based on measurements that were gained by the associated measuring suite (cf. Section 5.5), stored in the `source` field.

`AggregatedProxy` An attribute instance based on aggregation. This means that the formula, which is stored in the super class field `implementation`, describes the aggregation of those attribute instance values the composition association references.

`AttributeInstanceValue` A single attribute instance value, created at a particular point in time. It is extracted from the `AttributeInstance` to enable the consideration of multiple values for one instance, e.g., if T-S3:⟨Interest⟩ is set to *sample*. Storing the point in time enables provenance of job cancellation and considering development over time as required by NFR-5 and NFR-6, respectively.

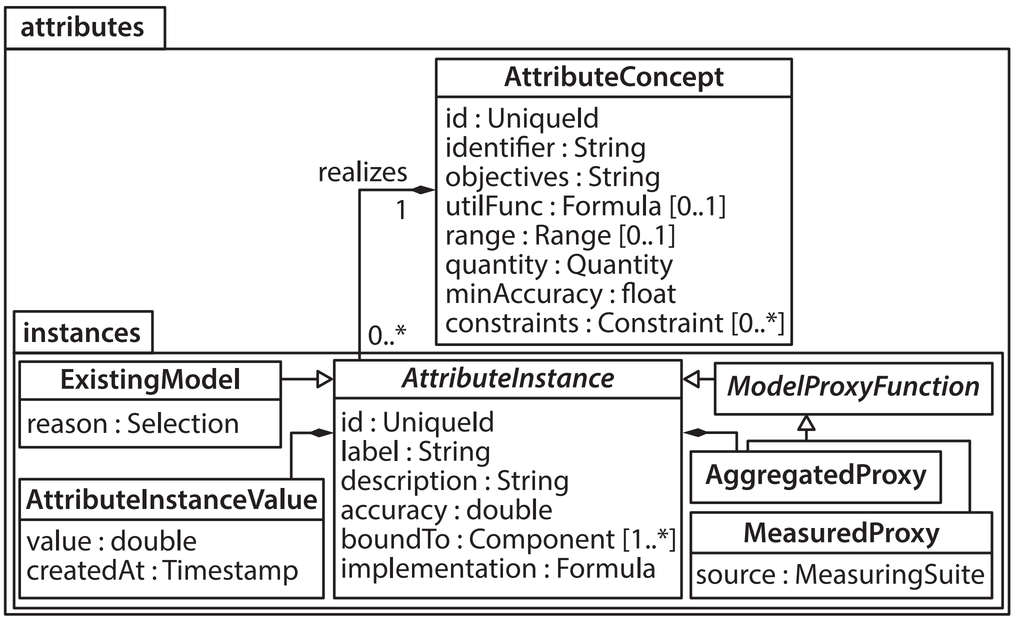
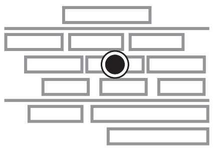


Figure 6.2: The provenance information model's `attributes` package, grouping classifiers that describe quantitative IT infrastructure attributes.



→ *T.E. 6.1 (p. 145)*

6.1.4 A4 – Define reasoning parameters

Guided by decisions made in action T-A3:A2, action T-A3:A4

1. formalizes required reasoning parameter information, and
2. potentially extends the set of considered IT infrastructure component types T-A3:O1 (cf. above) to T-A3:O2, e.g., if a modification concerns a component type that is not yet contained.

Sentence template T-S5 - Define reasoning parameter formalizes the alluded reasoning parameter information. Template element 6.10 depicts its grammar part, its production part is detailed below. Documentation of sentence template T-S5 employs classifiers of the provenance information model package `reasoningproject`, which is explained in Section 5.3.4.

Reasoning parameter ⟨Identifier⟩ describes ⟨Objectives⟩ given in ⟨Quantity⟩.

⟨Identifier⟩	⊨	⟨GIdentifier⟩	↢ T-A3:A4
⟨Objectives⟩	⊨	⟨ParameterType⟩String	↢ T-A3:A4
⟨ParameterType⟩	⊨	Configuration|Modification	↢ T-D7
⟨Quantity⟩	⊨	⟨GQuantity⟩[within⟨Range⟩]	↢ T-D8
⟨Range⟩	⊨	⟨GRange⟩	↢ T-D8

Template element 6.10: Grammar part of T-S5 - Define reasoning parameter.

⟨Identifier⟩ Identifies the reasoning parameter within the reasoning project, and is added to the identifier set in T-S3:⟨Parameters⟩.

⟨Objectives⟩ Describes the reasoning parameter's goal in a non-formal, textual way. It aims at enhancing the mostly pure numeric ⟨Quantity⟩ variable (cf. below) to reduce mismatch risk, e.g., two instances apply the same quantity but apply differing meanings (EG-6.2) what is not covered by the ⟨Quantity⟩ variable. Besides, it labels the reasoning parameter as modification or configuration parameter (cf. Section 4.1) in variable ⟨ParameterType⟩. Decision template T-D7 introduced below guides value setting.

⟨Quantity⟩ Describes the reasoning parameter's goal in a formal, mostly numeric way. The variable specifies a regular quantity (↗KB p. 263), and optionally a value range. The latter must be set individually for a particular reasoning project, as range appropriateness depends on the reasoning and attribute concept objectives. Decision template T-D8 introduced below guides value setting.

Select reasoning objectives

Decision template T-D7 – Select reasoning parameter objectives covers a reasoning parameter's objectives and processes variable T-S5:⟨ParameterType⟩.

Template element 6.11 depicts the template's implication table, documentation employs classifiers of the provenance information model package `reasoningproject`, which is explained in Section 5.3.4.

Opt.	Explanation and implications
I1	**Parameter type** – Defines the reasoning parameter being a modification or configuration parameter (cf. Section 4.1). ↦ Action T-A3:A10 respects only modification parameters to specify the goals of an iteration. Thus, each additional modification parameter tends to require an additional iteration in the reasoning function compilation process in *Phase B*. ↦ Although action T-A3:A19 can theoretically apply optimization and What-if analysis approaches on both, modification and configuration parameters, the latter's fixed nature (cf. Section 4.1) renders this intent futile. Hence, if one of the alluded reasoning tools should be used, the reasoning parameter must be a modification parameter.

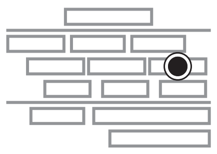
Template element 6.11: Implication table of T-D7 - Select reasoning parameter objectives.

Select reasoning parameter quantity

Decision template T-D8 – Select reasoning parameter quantity processes variable T-S5:⟨Quantity⟩. Template element 6.12 depicts the template's implication table, documentation employs classifiers of the provenance information model package `reasoningproject`, which is explained in Section 5.3.4.

Opt.	Explanation and implications
I1	**Quantity and scale** – Describes the reasoning parameter's goal in a formal, mostly numeric way (↗KB p. 263). ↦ The more common quantity and scale are, the more likely it is that action T-A3:A12 finds an appropriate model integration candidate, which is able to process the reasoning parameter. ↦ If no suitable model integration candidate exists, model proxy creation in action T-A3:A13 might conduct measuring, which should be feasible for the selected quantity and scale.
I2	**Range** – Defines valid or relevant minimum and maximum values of the reasoning parameter. ↦ Setting no range renders plausibility checks in action T-A3:A14 very difficult, as the requirements are not clear, and especially the extremal values cannot be analyzed for odd model behavior. For instance, if an integrated model predicts cluster performance for up to 100 nodes, but reasoning requires covering 200 nodes, the model might not be capable of and particularly might compute misleading results. ↦ A big range might increase reasoning execution costs in action T-A3:A19, because more values must be examined.

Template element 6.12: Implication table of T-D8 - Select reasoning parameter quantity.



6.1.5 A5 – Model IT infrastructure

→ *T.E. 6.1 (p. 145)*

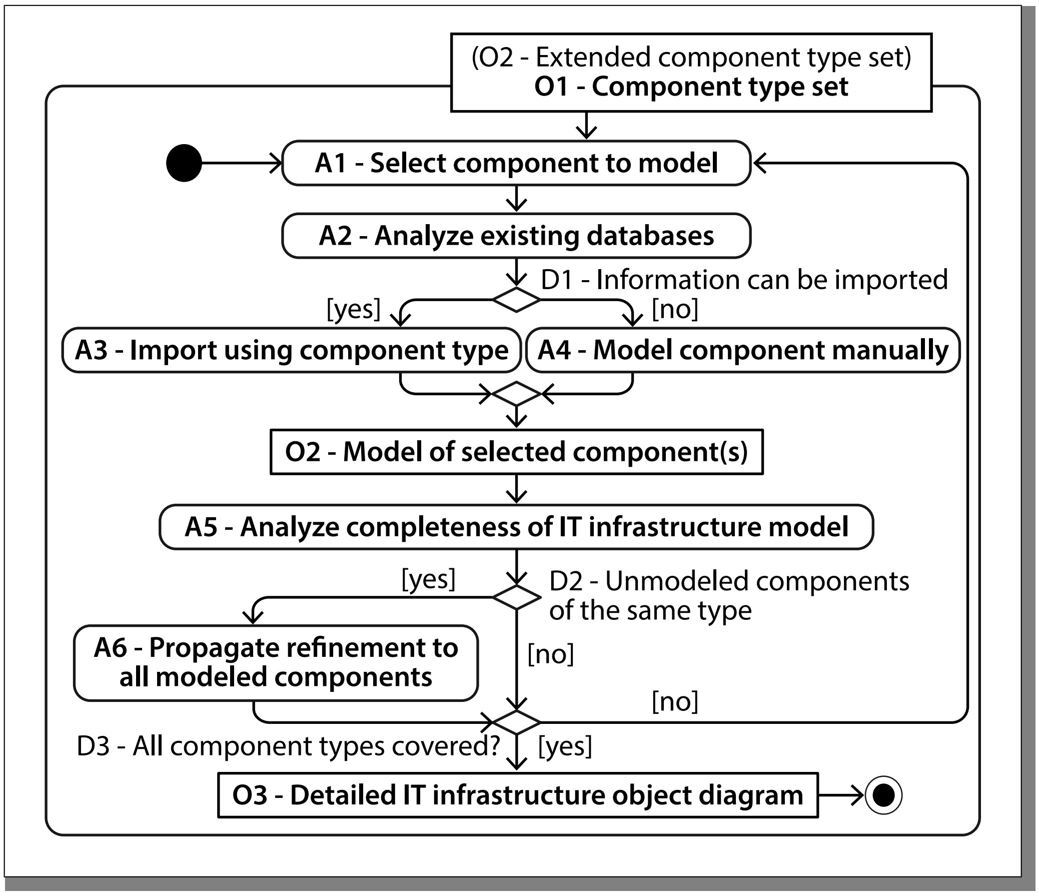
Action T-A3:A5 assembles a model of the considered IT infrastructure based on the notion explained in Section 5.4. Guided by the reasoning objectives driven approach (cf. Section 4.3) and the required simplicity (cf. NFR-7), it

- details only IT infrastructure elements and aspects that are important for the reasoning project as specified in the IT infrastructure component type list T-A3:O2, created by action T-A3:A3 and T-A3:A4,
- does explicitly not aim at describing the IT infrastructure as detailed and extensively as possible,
- does not decide about or set the model granularity level on its own authority, but uses the provided specifications.

In doing so, action T-A3:A5 performs two tasks, namely it *creates an UML object diagram*, and it *places this a UML object diagram in form template* to ease its referencing.

Create UML object diagram

Activity template T-A4 - Model IT infrastructure formalizes the IT infrastructure modeling activity. It results in an UML object diagram T-A4:O3 (↗KB p. 271) that uses classifiers of the provenance information model package `itinfrastructure` (cf. Section 5.4.5). Template element 6.13 depicts the activity template's action flow and illustrates its main approach, i.e., iterating the given IT infrastructure component type list and expand the model accordingly. The template's action details are described below:


(O2 - Extended component type set)
O1 - Component type set
A1 - Select component to model
A2 - Analyze existing databases
D1 - Information can be imported
[yes]
[no]
A3 - Import using component type
A4 - Model component manually
O2 - Model of selected component(s)
A5 - Analyze completeness of IT infrastructure model
[yes]
D2 - Unmodeled components of the same type
A6 - Propagate refinement to all modeled components
[no]
[no]
D3 - All component types covered?
[yes]
O3 - Detailed IT infrastructure object diagram

Template element 6.13: Action flow of T-A4 - Model IT infrastructure.

T-A4:A1 Selects the next component type or slice of the IT infrastructure to model in the current iteration. Selection is applied on the remaining set of un-modeled component types that have not been addressed by previous iterations. If it is the first iteration, certainly all IT infrastructure component types provided in T-A3:O2 can be selected. The selection is strongly driven by the reasoning project's overall goals and the black box approach (cf. Section 5.4.3).

T-A4:A2 Screens existing data sources for information that could be used for interfacing with third-party models (cf. Section 5.4.2). T-A4:D1 processes the binary *yes/no* result about availability of (partial) data.

T-A4:A3 In case T-A4:D1 validates to *yes*, the action imports information about the considered IT infrastructure components into the provenance information model, using the component type as linking element (cf. Section 5.4.2). The theoretical infinite set of potential data sources, ranging from wide-spread GLUE or CIM databases (cf. Section 2.2.3) to proprietary tools, renders import implementation a highly casuistic task and hence, no generic import algorithm is provided.

T-A4:A4 In case T-A4:D1 validates to *no*, the currently iterated component(s) is (are) modeled manually.

T-A4:O2 Stores results of actions T-A4:A3 and T-A4:A4 for further processing.

T-A4:A5 Screens the modeled IT infrastructure for completeness in general and for components having the same type as the currently iterated component(s), in particular. The screening result is processed by T-A4:D2 and T-A4:D3.

T-A4:A6 In case T-A4:D2 validates to *yes*, there are IT infrastructure components left that are not represented in the so far created model but are of the same type as the currently iterated component(s). The action executes Algorithm 1 (cf. Section 5.4.3) for propagating the modeling results of the current iteration to the IT infrastructure object diagram.

T-A4:D3 Triggers an additional iteration if component types remain un modeled.

T-A4:O3 The activity finishes with a provenance information model augmented by objects describing the considered IT infrastructure.

Place object diagram in form template

Form template T-F4 - Document IT infrastructure model, depicted in Template element 6.14, acts as container for the UML object diagram activity template T-A4 creates, to enable its referencing in variable T-S3:⟨ITInfrastructure⟩.

F1 – Unique ID

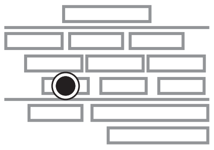
Filling objectives and rules – Uniquely identify the IT infrastructure model for referencing, especially in T-S3:⟨ITInfrastructure⟩.

F2 – IT infrastructure model

Valid data – UML object diagram
Justification – The form template acts only as container for an IT infrastructure *UML object diagram* to enable referencing the diagram.
Filling objectives and rules – The UML object diagram created by activity template T-A4. The UML object diagram models the considered IT infrastructure according to the component type set defined in T-A3:O2.

Template element 6.14: Form of T-F4 - Document IT infrastructure model.

6.1.6 A6 – Select workload



→ *T.E. 6.1 (p. 145)*

Workload and its generated load (cf. Section 2.3.3) tend to have a strong impact on IT infrastructure attributes (cf. Section 2.4) and hence, on a reasoning project. Besides, most model integration candidates selected in action T-A3:A12 call in some way for (work)load input parameters [41]. Different workload classes and a variety of configurations (cf. Section 2.3) prohibit using *the* workload, but require a thorough selection.

Thus, action T-A3:A6 employs decision template T-D9 – Select workload to select suitable workload. T-D9 addresses the fact that a certain workload's suitability strongly depends on the individual intent of a reasoning project [34], and that each class and configuration has several implications on and benefits for a reasoning project's statements, as subsequently described. Decision template T-D9 consists of three elements:

Implication table Considers the workload classes *Application* and *Benchmark* (cf. Section 2.3.1), as depicted in Template element 6.15.

Decision tool Reflects the variety and individual suitability of workload options by targeting at a *set* of potential options and not on a single one, like most of the other decision tools in this thesis. It achieves this objective by iteratively applying reduction rules on option sets in a decreasing abstraction level. For instance, it starts with the general classes *Application* and *Benchmark*, removes *Application*, and proceeds with the options of the remaining *Benchmark* class. Three aspects recommend using set theory for workload selection:

1. Small set sizes make it clear and intuitive to use;
2. Adjusting option sets and/or reduction rules allows customization;
3. The alluded variety calls for flexibility, which is provided by set theory, but not by other approaches, like decision trees.

The decision tool provides for several situations the reduction rule(s) and the sets to apply the reduction rule(s) on, as depicted in Template element 6.16 and 6.17.

Documentation Uses the provenance information model package `workload` and performs two tasks, namely it *creates an UML object diagram*, and it *places this a UML object diagram in form template* to ease its referencing.

(1) Reasoning could use the *FLOP/s* values computed by the *High Performance LINPACK* (HPL) benchmark (cf. Appendix B). *(2)* An *instruction* is not the same on a RISC and CISC machine (cf. Section 2.2.1) and hence, “instructions per second” cannot be compared. *EG-6.6*

Opt.	Explanation and implications
I1	**Application** – Covers production software executed on the IT infrastructure (cf. Section 2.3.1). ↦ Action T-A3:A18 might require an *abstract workload model* of the application for load computation (cf. Section 2.3.3). ↦ Model proxy function compilation in action T-A3:A13 or physical load derivation in action T-A3:A18 might require the application's execution. ↦ The non-standardized nature of most applications prohibits a comparison with other IT infrastructures in *Phase C*.
I2	**Benchmark** – Real-world computing task emulator (cf. Section 2.3.2). ↦ Action T-A3:A18 might require an *abstract workload model* of the benchmark for load computation (cf. Section 2.3.3). ↦ Model proxy function compilation in action T-A3:A13 or physical load derivation in action T-A3:A18 might require the benchmark's execution. ↦ Enables the use of benchmark results for reasoning in action T-A3:A19 (*EG-6.6:1*). ↦ Comparison of benchmark results with other IT infrastructures, e.g., in the Top500 list, requires benchmark execution in action T-A3:A13 and T-A3:A18 to use the same/predefined compiler and execution configurations, and ensuring of comparability (*EG-6.6:2*). ↦ The urgent need for thorough documentation of used parameters and configurations (cf. Section 2.3.2), might cause additional effort in action T-A3:A13 and T-A3:A18.

Template element 6.15: Implication table of T-D9 - Select workload.

The reduction rule set $\odot$ is applied on the available workload options, resulting in the final set of workload options to chose. Reduction rule justifications base on workload notion introduced in Section 2.3.2.

$$\begin{aligned} Classes &= \{Application, Benchmark\} \\ Benchmarks &= \{system, partial, combined, kernel, synthetic\} \\ ConcreteBenchmarks &= \text{Defined according to benchmark classes} \end{aligned}$$

Compare results to other IT infrastructures

1. $\odot \cup \{Classes \setminus Application\}$
2. $\odot \cup \{ConcreteBenchmarks \setminus \text{Not wide-spread benchmarks}\}$

Result comparison requires "standardized" workload, which is only achieved by benchmarks ($\rightarrow$ 1). They must also be widespread, since isolated used benchmarks provide no data base ($\rightarrow$ 2). If a concrete workload is postulated, like the HPL benchmark for the Top500 list [128], selection process chooses the dictated workload and stops.

Focus on specific domain

1. $\odot \cup \{Benchmarks \setminus synthetic\}$

For reasoning projects focusing on a specific domain, e.g., HPC cluster optimization for CFD applications, synthetic benchmarks are removed, as they do not describe an application domain sufficiently ($\rightarrow$ 1).

Quality criteria

1. $\odot \cup \{ConcreteBenchmarks \setminus \text{Not well-documented benchmarks}\}$

Especially for benchmarks there is the urgent need to document configuration and compiler parameters, so all benchmarks not providing these information are removed ($\rightarrow$ 1).

- **Load characteristics** – Load generation recommends synthetic benchmarks, as they consist of a variety of instructions, which mostly results in a balanced and general component load.
  $\odot \cup \{Benchmarks \setminus kernel\}$
- **IT infrastructure components** – Employed benchmarks should stress (at least) those IT infrastructure components that are relevant to the reasoning project as defined in T-A3:O2. Hence, either system, partial, or combined benchmarks are suitable.

Template element 6.16: Decision tool of template T-D9 - Select workload for workload class selection.

- **Heterogeneous IT infrastructures** – As Section 2.2 explains, there is a multitude of IT infrastructures. Hence, all concrete benchmarks that are not executable on the IT infrastructure at hand must be removed.
- **Attribute vector values** – In case the output of a benchmark should be used in the attribute vector for reasoning, it must be ensured that the benchmark's statements, meanings, and scale match the pursued objective (*EG-6.7:1*).
- **Proxy application** – Whenever a benchmark should proxy an application, e.g., because the application binary is not available, the benchmark should describe the application as accurate as possible, preferably on each creation layer (cf. Section 2.3.2, *EG-6.7:2*). In any case, checklist template T-C3 (cf. Template element 5.7) must be fulfilled.

Template element 6.17: Decision tool of template T-D9 - Select workload for concrete benchmark selection.

EG-6.7

(1) The *Whetstone* benchmark [106] results in *mega Whetstone instructions per second* (MWIPS), the *LINPACK* benchmark [125] results in *Floating Point Operations per Second* (FLOP/s). Both can be used to compare the performance of two systems, respectively. In contrast, a MWIPS number and a FLOP/s number cannot be compared. When using the Whetstone benchmark, IT infrastructure characteristics must be respected, because instructions are different for CISC and RISC machines, and hence, "comparing the instructions of a CISC machine and a RISC machine is similar to comparing Latin and Greek" [122]. The mandatory match also applies for the workload characteristics: *(2)* due to on-chip caches and optimizing compilers, small benchmarks tend to lose their predictive value for big, memory consuming applications, since the entire benchmark fits in the local cache [407].

Create UML object diagram

The documentation part of decision template T-D9 uses the provenance information model package `workload`. Figure 6.3 depicts its structure, its classifiers are detailed below.

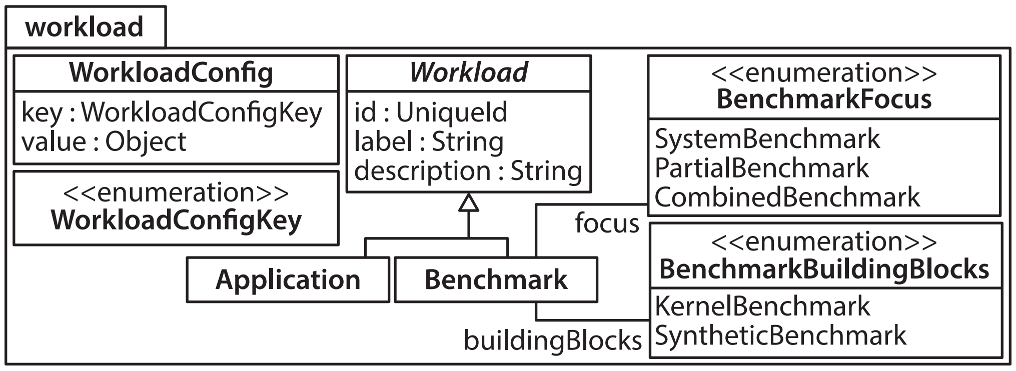


Figure 6.3: Classifiers of the provenance information model that represent and describe used workload and configuration parameters, respectively.

`Workload` A workload according to the notion explained in Section 2.3. The `label` and `description` fields store its name and summarize characteristics relevant to the reasoning project, respectively. The class is marked abstract to stress that workload is either an `Application` or a `Benchmark`. The `version` field reflects the (potential) great importance of the used parallel solver or tool version [304].

`Application` An application according to the notion explained in Section 2.3.1. The huge amount of potential applications and their characteristics (cf. Section 3.2.1) bans a detailed modeling. Instead, sub classing can be used to augment the class by individual information about the application.

`Benchmark` A benchmark according to the notion explained in Section 2.3.2. The benchmark characteristics *focus* and *building blocks* are represented by the fields `focus` and `buildingBlocks`. Both use values provided by the enumerations `BenchmarkFocus` and `BenchmarkBuildingBlocks`, respectively.

`WorkloadConfig` A single configuration parameter used for workload creation, compilation, and execution. The workload configuration is modeled as key value store to achieve high flexibility. The key is a

`WorkloadConfigKey` to foster type safety, the value is an `Object` to underpin the variety of stored values. `WorkloadConfig` objects are not associated to a `Workload`, but to a `ReasoningSuite` (cf. Section 5.3.4) or a `MeasuringSuite` (cf. Section 5.5.2) to support differing configurations for the same workload according to the specific situation.

Place UML object diagram on form template

Form template T-F5 - Document workload model acts as container for the workload UML object diagram to enable its referencing in variable T-S3:⟨Workload⟩. Template element 6.18 depicts the form template.

F1 – Unique ID

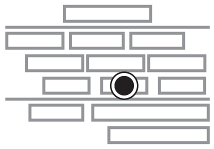
Filling objectives and rules – Uniquely identify the workload for referencing, especially in T-S3:⟨Workload⟩.

F2 – Workload model

Valid data – UML object diagram
Justification – The form template acts only as container for a workload UML object diagram to enable the diagram's referencing.
Filling objectives and rules – The UML object diagram that models the workload selected by decision template T-D9.

Template element 6.18: Form of T-F5 - Document workload model.

6.1.7 A7 – Document assumptions



→ T.E. 6.1 (p. 145)

An assumption is understood as a statement that is accepted without proof and regarded as fundamental to a subject (*EG-6.8:1*). Although as little as possible assumptions should be made (Occam's razor) [84], assumptions are an essential part of modeling and are made by existing model integration candidates that might be selected in action T-A3:A12 (*EG-6.8:2*).

EG-6.8

> *(1)* Assumptions are fundamental in several ways, e.g., microeconomics (usually) assumes perfect competitive markets. Also the realm of computer science extensively uses assumptions, e.g., about job dependencies in workflows [437], or about preciseness of resource information for Grid schedulers [416] while modeling performance of parallel computing on non-dedicated heterogeneous networks of workstations [425]. *(2)* Pfeiffer et al. base their modeling on the assumption that "application run time can be approximated as a linear combination of inverse speeds and latencies mostly obtained from microkernels" [319, p. 1].

Assumption making and in particular the quality of made assumptions heavily depend on experience, available information, and individual efforts. Furthermore, the (fast) development in computer science might antiquate a written and implicitly limited set of assumptions very quickly. This renders the definition of generically applicable rules or processes how to make a (good) assumption difficult and questionable. Nevertheless, ensuring reproducibility and a profound scientific mode of operation require thorough documentation of made assumptions [80, 178], especially because they represent aspects that are normally taken granted and are seldom made explicit. A clear documentation also helps and eases (nearly) all activities of a reasoning project, because difficult or expensive actions can be omitted since they are rendered unnecessary by a made assumption.

Though, action T-A3:A7 dictates the documentation of a particular assumption using form template T-F6 - Document reasoning project assumption. The assumption's identifier in T-F6:F1 is added to the identifier set in T-S3:⟨Assumptions⟩, the employed provenance information model classifiers are part of the `reasoningproject` package and explained in Section 5.3.4. Template element 6.19 depicts form template T-F6.

F1 – Unique ID

Filling objectives and rules – Uniquely identify the assumption for referencing, especially in T-S3:⟨Assumptions⟩.

F2 – Assumption summary

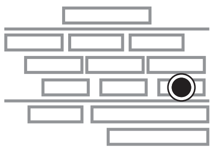
Filling objectives and rules – (Shortly) summarize the assumption, e.g., "considered modifications can be executed", to ease its use. Depending on the assumption's extend, field T-F6:F2 is optional.

F3 – Assumption

Filling objectives and rules – Describe the made assumption as extensive and clear as possible, and explain it in a way that is directly testable [178].

Template element 6.19: Form of T-F6 - Document reasoning project assumption.

6.1.8 A8 – Document constraints



→ T.E. 6.1 (p. 145)

Constraints state rules that span an entire reasoning project, e.g., limiting the set of model integration candidates to a specific type, or negating license fees. In analogy to requirements engineering, constraints can be seen as non-functional requirements on the reasoning project. This important role calls for a thorough constraint documentation, because they might (heavily) influence the entire reasoning project and in particular the reasoning function compilation process in *Phase B*.

Though, action T-A3:A8 dictates the documentation of a particular constraint using form template T-F7 - Document reasoning project constraint. The constraint's identifier in T-F7:F1 is added to the identifier set in T-S3:⟨Constraints⟩, the employed provenance information model classifiers are part of the `reasoningproject` package and explained in Section 5.3.4. Template element 6.20 depicts form template T-F7.

F1 – Unique ID

Filling objectives and rules – Uniquely identify the constraint for referencing, especially in T-S3:⟨Constraints⟩.

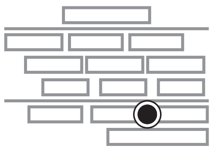
F2 – Constraint summary

Filling objectives and rules – (Shortly) summarize the constraint, e.g., "integrate only open-source models", to ease its use. Depending on the constraint's extend, field T-F7:F2 is optional.

F3 – Constraint

Filling objectives and rules – Describe the constraint as extensive and clear as possible.

Template element 6.20: Form of T-F7 - Document reasoning project constraint.



→ T.E. 6.1 (p. 145)

6.1.9 A9 – Create reasoning function skeleton

Action T-A3:A9 extracts information from sentence templates T-S4 and T-S5 to create the reasoning function skeleton depicted in Equation 6.1.

$$f(\underbrace{mod_1, ..., mod_n}_{\substack{\text{Set 1}\\ \text{Modification}\\ \text{parameters}\\ \text{Template T-S5}}}, \underbrace{conf_1, ..., conf_m}_{\substack{\text{Set 2}\\ \text{Configuration}\\ \text{parameters}\\ \text{Template T-S5}}}) = \underbrace{\begin{pmatrix} attr_1 \\ ... \\ attr_z \end{pmatrix}}_{\substack{\text{Vector}\\ \text{Attribute}\\ \text{concepts}\\ \text{Template T-S4}}} \tag{6.1}$$

Set 1 Contains the *modification* reasoning parameters defined in sentence template T-S5 (cf. Template element 6.10). Variable quantity, scale, and domain are provided by the template, as well.

Set 2 Contains the *configuration* reasoning parameters defined in sentence template T-S5 (cf. Template element 6.10). Variable quantity, scale, and domain are provided by the template, as well.

Vector Contains the IT infrastructure attribute concepts defined in sentence template T-S4 (cf. Template element 6.5).

6.2 Phase B – Reasoning function compilation

→ *Fig. 6.1 (p. 144)*

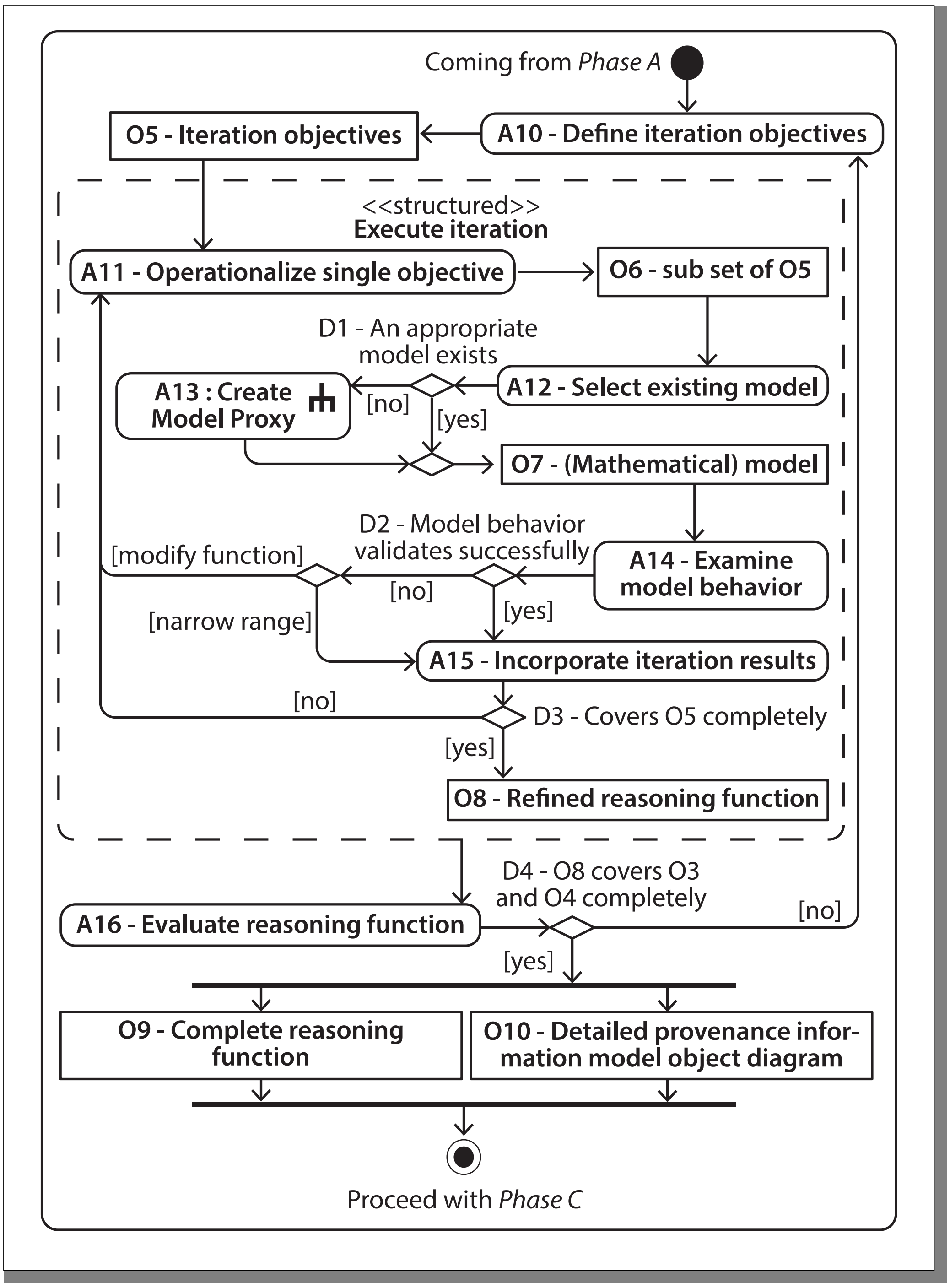
One of the research's main ideas is the iterative creation of an individual reasoning function, beginning with a coarse-grained function skeleton that is refined until the reasoning function covers all aspects required by the reasoning suite T-A3:O3 (cf. Section 4.1).

Phase B of activity template T-A3 formalizes this iterative assembly, called *compilation*, of an *individual* reasoning function. Strictly guided by the reasoning suite created in *Phase A*, *Phase B* iteratively transforms the given reasoning function skeleton (T-A3:O4) into a full reasoning function (T-A3:O9). Besides, *Phase B* augments the provenance information model with documentation about the reasoning function compilation process (T-A3:O10). Template element 6.21 depicts the action flow of *Phase B* and emphasizes the iterative refinement.

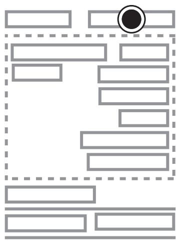
A single iteration consists of three phases:

1. Action T-A3:A10 selects one or multiple aspects of the reasoning suite that are not yet covered by the reasoning function. Besides, the action investigates aspect interdependency and implications.
2. Actions T-A3:A11 to T-A3:A15 execute the iteration, expand the reasoning function, and document partial results.
3. Action T-A3:A16 evaluates the reasoning function and triggers an additional iteration in case the reasoning function does not completely comply to the reasoning suite.

The remainder of this section provides the action details of *Phase B*.


Coming from Phase A
O5 - Iteration objectives
A10 - Define iteration objectives
<<structured>>
Execute iteration
A11 - Operationalize single objective
O6 - sub set of O5
D1 - An appropriate model exists
A13 : Create Model Proxy
[no]
[yes]
A12 - Select existing model
O7 - (Mathematical) model
D2 - Model behavior validates successfully
[modify function]
[no]
[yes]
A14 - Examine model behavior
[narrow range]
A15 - Incorporate iteration results
[no]
D3 - Covers O5 completely
[yes]
O8 - Refined reasoning function
D4 - O8 covers O3 and O4 completely
A16 - Evaluate reasoning function
[no]
[yes]
O9 - Complete reasoning function
O10 - Detailed provenance information model object diagram
Proceed with Phase C

Template element 6.21: Action flow of T-A3 - Reasoning methodology - Phase B.



→ *T.E. 6.21 (p. 175)*

6.2.1 A10 – Define iteration objectives

Action T-A3:A10 prepares an iteration's execution by stating objectives and rules that guide the accomplishment of action T-A3:A11 to T-A3:A15. In doing so, it extracts a subset of the reasoning suite, formalizes these information in sentence template T-S6 - Define iteration objectives, and places it in T-A3:O5. Template element 6.22 depicts the sentence template's grammar part, the list below provides the sentence template's production part.

Iteration ⟨Objectives⟩ of ⟨Elements⟩ in the reasoning function.

$$
\begin{aligned}
\langle\text{Objectives}\rangle &\vDash \text{adds}| & \Leftarrow \text{T-D10}\\
&\quad \text{improves accuracy (by}|\text{to)}\langle\text{GPercent}\rangle\\
\langle\text{Elements}\rangle &\vDash \langle\text{Parameter}\rangle|\langle\text{AttributeConcept}\rangle|\\
&\quad \langle\text{Elements}\rangle\\
\langle\text{Parameter}\rangle &\vDash \langle\text{Modification}\rangle|\langle\text{Configuration}\rangle\\
\langle\text{Modification}\rangle &\vDash \langle\text{GIdentifierSet}\rangle\\
\langle\text{Configuration}\rangle &\vDash \langle\text{GIdentifierSet}\rangle\\
\langle\text{AttributeConcept}\rangle &\vDash \langle\text{GIdentifierSet}\rangle
\end{aligned}
$$

Template element 6.22: Grammar part of T-S6 - Define iteration objectives.

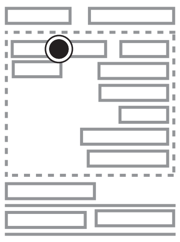
⟨Objectives⟩ Describes the iteration's objectives, being either adding an element to the reasoning function, or improving the accuracy of an already existing one to or by the specified value. Theoretically, an arbitrary set of objectives can be selected from the reasoning suite for a particular iteration. In contrast, objective *ordering* is required to strictly respect the component type hierarchy of the IT infrastructure model in order not to contradict the applied top-down refinement and evaluation (cf. Section 4.3, *EG-6.9*). Decision template T-D10 – Select iteration objectives tackles this problem by detailing potential implications. Template element 6.23 depicts the template's implication table, its documentation employs classifiers of the provenance information model package `reasoningproject` explained in Section 5.3.4.

⟨Elements⟩ Contains identifiers of processed attribute concepts, or modification and configuration parameters, given by T-S3:⟨AttributeConcepts⟩ and T-S3:⟨Parameters⟩, respectively.

Opt.	Explanation and implications
I1	**Extend function (co) domain** – Add parameter(s) or attribute vector entries within the iteration. ↦ Processing all elements that are likely to be covered by 1) the same model integration candidate selected in action T-A3:A12, or 2) the same measuring system used in action T-A3:A13, increases efficiency (cf. NFR-8), because for all added elements the same artifacts can be used. ↦ Processing all elements, action T-A3:A16 can evaluate using the same measuring system, reduces cost, as there is no need to execute measuring activity T-A2 several times. Instead, the assembled measuring setup can be reused. ↦ The reasoning function's partial refinement, the top-down driven incorporation of refinements in action T-A3:A15, and the immediate evaluation of those refinements in action T-A3:A16 require compliance to the component type hierarchy defined in T-A3:O2. Otherwise, the partial replacement of reasoning function elements, being one of the main design concepts (cf. Section 4.2), is not possible (EG-6.9).

Template element 6.23: Implication table of T-D10 - Select iteration objectives.

Example EG-4.3 iteratively refines the reasoning function skeleton $f(n) = 9.4 \times n + 4$ that deals with HPC cluster power consumption. A reasoning suite could require the final reasoning function to compute the cluster's power consumption depending on the CPU clock speed, and it could dictate the component types *node* > *CPU*. If the iteration objectives do not respect this hierarchy, iterative replacement doesn't work: a valid iteration objective could be "Refine 9.4 by a model that describes the power consumption of one node at load *l*". It is valid, since the fixed value and the replacing model are on the same granularity level, i.e., the node. The next iteration, in turn, would refine the used model by incorporating CPU clock speed. In contrast, an invalid iteration objective would be "Refine the coarse-grained reasoning function by a CPU model". It is invalid, as it skips the cluster, and values computed by the refinement cannot reasonably be compared with the replaced value 9.4. *EG-6.9*



→ *T.E. 6.21 (p. 175)*

6.2.2 A11 – Operationalize single objective

In contrast to action T-A3:A10 that rather strategically selects iteration objectives and examines implications, action T-A3:A11 substantiates a single objective contained in T-A3:O5. The action operationalizes the objective by detailing the refinement intent and specifying execution guidelines, using form template T-F8 - Document iteration task. It is depicted in Template element 6.24, and can be delegated to the role being responsible or executing the iteration (cf. Section 5.1). A form template instance is stored in T-A3:O6.

F1 – Unique ID

Filling objectives and rules – Uniquely identify the operationalization for referencing, especially in action T-A3:A12 and T-A3:A13.

F2 – Objective

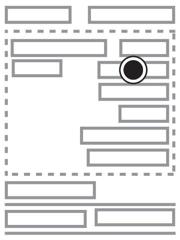
Valid data – A value of variable T-S6:⟨Objectives⟩.
Filling objectives and rules – Describe the objective to implement in the current iteration and how to refine the reasoning function.

F3 – Information regarding conduction

Filling objectives and rules – Optional field. Describe hints or any other aspect that might support task execution according to the specifications and demands formulated in the reasoning suite (T-A3:O3).

Template element 6.24: Form of T-F8 - Document iteration task.

6.2.3 A12 – Select existing model



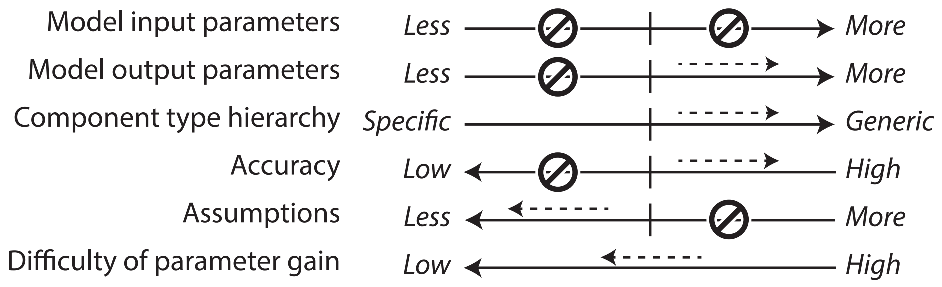
→ *T.E. 6.21 (p. 175)*

One of the process model's fundamental design concepts is the integration of existing artifacts and particularly of existing models (cf. Section 4.2). Thus, action T-A3:A12 uses decision template T-D11 – Select model integration candidate to select a model integration candidate compliant to the reasoning suite (T-A3:O3), and to the objectives of the current iteration (T-A3:O6). Decision template T-D11 consists of three parts:

- **Implication table** Addresses the model integration candidate's input and output parameters. Because most related decisions and actions have already been made and executed, the implication table is comparatively small, as depicted in Template element 6.25.
- **Decision tool** Considers several aspects of a model integration candidate. At its top, Template element 6.25 overviews the aspects graphically: each aspect is illustrated by an ordinal scale, a vertical line identifies the requirement stated in the reasoning suite. The arrow and prohibition sign indicate the aspired direction and invalid areas, respectively.
- **Documentation** Uses the `attributes` package of the provenance information model as explained in Section 6.1.3.

Opt.	**Explanation and implications**
I1	**Model input** – The model's consumed input. ↦ A model supporting less input parameters than required by the iteration objectives (T-A3:O6) tends to trigger additional iterations in T-A3:D2. Thus, it might cause additional costs. ↦ A model consuming more input parameters than required by the iteration objectives (T-A3:O6) might cause additional cost when generating input values in action T-A3:A18, and when executing the reasoning in action T-A3:A19.
I2	**Model output** – The model's produced output. ↦ A model producing not all parameters required by the iteration objectives (T-A3:O6) tends to trigger an additional iteration in T-A3:D2. Thus, it might cause additional costs.

Template element 6.25: Implication table of T-D11 - Select model integration candidate.



Model input parameters Describes the match of the model's input parameters, and the reasoning suite (T-A3:O3) and iteration objectives (T-A3:O6). It should perfectly match, because less or more input parameters might cause an additional iteration or additional effort in value generation (cf. implication table in Template element 6.25), respectively.

Model output parameters Describes the match of the model's output parameters, and the reasoning suite (T-A3:O3) and iteration objectives (T-A3:O6). The model should provide at least the required output parameters, because less might cause an additional refinement. In contrast, more output parameters is without problems, because they are provided as by-product.

Component type(s) Describes the IT infrastructure component type set the model can be applied to. The arrow represents the position in the component type hierarchy according to the process model's notion of IT infrastructures (cf. Section 5.4). Usually, the focus of the candidate is correlated to its accuracy.

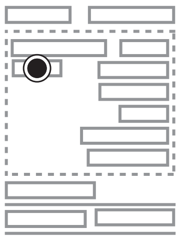
Accuracy Describes the model result accuracy. The value should be preferably high in general, and above the value specified in attribute concept definitions (cf. action T-A3:A3), in particular.

Assumptions Compares the amount of similar assumptions made by the model and specified in the reasoning suite in T-S3:⟨Assumptions⟩. A model making less assumptions can be selected, a model making more should be avoided.

Difficulty of parameter gain Some models process more input parameters than required by the iteration objectives (T-A3:O6). In this case, value assembly should be as easy as possible and avoid problematic situations, like "there is no direct way to measure the number of architected register file accesses (regfile)" [217].

Template element 6.26: Decision tool of template T-D11 - Select model integration candidate.

6.2.4 A13 – Create model proxy



→ *T.E. 6.21 (p. 175)*

Despite the plethora of existing models (cf. Section 7.4), requirements formulated by the reasoning objectives in general and by decision template T-D11 in particular might disqualify (available) model integration candidates, as they do not fulfill all criteria (sufficiently).

Thus, action T-A3:A13 creates a *model proxy function* that substitutes the missing model to enable a refinement anyhow. Compared to the (mostly) generic and broadly applicable substituted model, the model proxy function highly fits (over-fits) to the considered situation in order to accomplish a good costs vs. portability trade-off (*EG-6.10*). Nevertheless, since the model proxy function solely aims at replacing a model for a specific reasoning project, and not at developing a generic and portable solution, this over-fitting can be neglected.

EG-6.10

> A model intended at describing the "power consumption of ARM processors" normally aims at being applicable to all ARM processors. In contrast, a model proxy function would focus on a specific situation and purposely omit a wide applicability, e.g., consider only an ARMv11 processor and extraordinary configurations of the processor's surrounding, in order to reduce the function creation costs, caused by intense measuring or investigation, to a minimum.

The main approach of action T-A3:A13 is the derivation of a compact, mathematical formula, based on collected raw data [30]. Both, derivation methods and raw data sources, are manifold (*EG-6.11*).

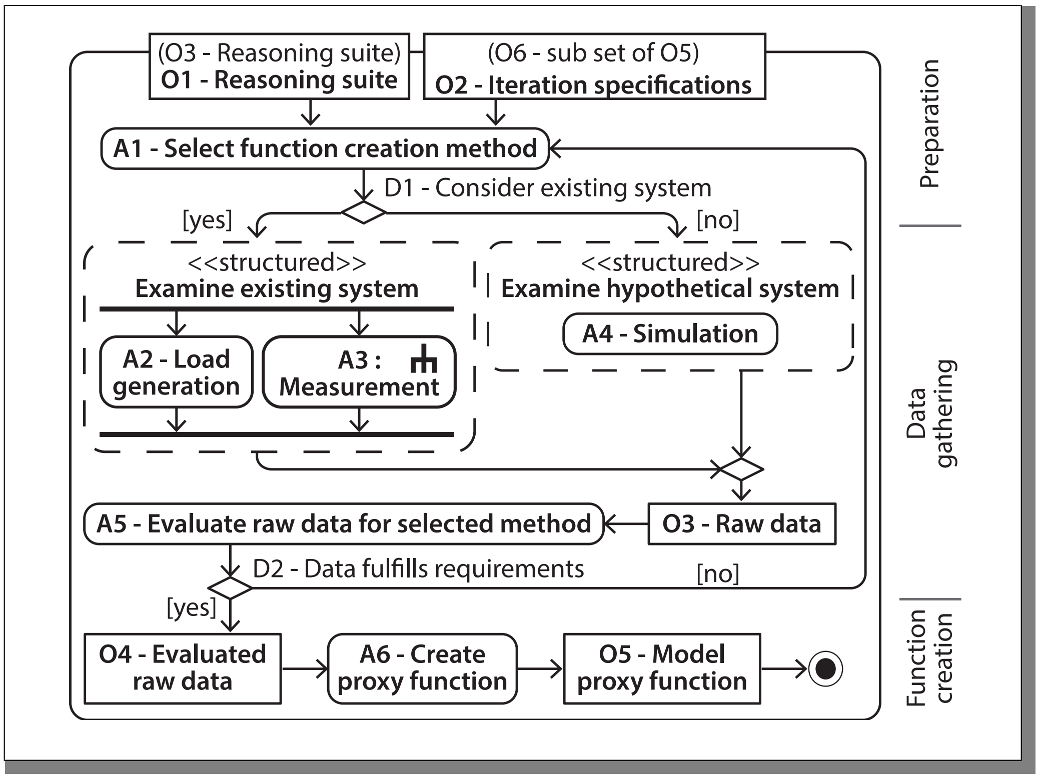
EG-6.11

> Exemplary derivation methods are curve fitting [13], multiple linear regression analysis [71], and aggregation of existing values. Also within a method class, there are several possible approaches. Regression, for instance, consists of linear, non linear, and multiple regression, each posing differing requirements on the experience and data. Exemplary data sources are measuring activities (cf. Section 5.5), hardware counters providing "application developers [...] information about the performance of critical parts of the application" [73, p. 189], and trace analysis.

Action T-A3:A13 tackles this variety by formalizing the creation process in activity template T-A5 - Create model proxy function. Template element 6.27 depicts the template's action flow and highlights its three phases:

- **Preparation** Selects the proxy function creation method and pre processes the reasoning suite (T-A3:O3) for decision point T-A5:D1.

- **Data gathering** Depending on the considered IT infrastructure, the phase executes the selected workload and measures relevant data on the IT infrastructure, or it conducts a simulation.
- **Function creation** Uses the gathered raw data to derive and assemble the final model proxy function. The main challenge for this phase is the underlying data, as it is of varying quality [137].



Template element 6.27: Action flow of T-A5 - Create model proxy function.

The subsequent itemization provides the template's action details:

T-A5:A1 Selects a model proxy function creation method (EG-6.11). Besides, it implicitly delimits the considered IT infrastructure, and stores the result in T-A5:D1. In addressing the variety of creation methods, action T-A5:A1 uses decision template T-D12 – Select model proxy function creation method for selection making. The decision template consists of two elements:

- **Implication table** Highlights implications of selecting a function creation method, and of considering a physical or hypothetical IT infrastructure, as depicted in template element 6.28.

- **Documentation** Uses classifiers of the provenance information model, especially of the `attributes` and `measuring` packages (cf. Section 6.1.3 and 5.5.2), depending on the applied method.

T-A5:A2 Executes the workload selected in action T-A3:A6 (contained in parameter T-A5:O1) to generate load for measuring activities in action T-A5:A3, which is executed in parallel. In case the selected workload cannot be used, an alternative workload is chosen using checklist template T-C3 (cf. Section 5.7).

T-A5:A3 Measures (relevant) raw data using activity template T-A2, while action T-A5:A2 executes the workload. Alternatively, the action processes log files [89]. In both cases, the result is stored in T-A5:O3.

T-A5:A4 Conducts a simulation on an IT infrastructure model, in case the considered IT infrastructure is not available at all, or a hypothetical IT infrastructure should be examined. In both cases, action T-A5:A2 and T-A5:A3 cannot be executed, as they require physical hardware (cf. decision template T-D12). The result is stored in T-A5:O3.

T-A5:A5 Checks the gained raw data in T-A5:O3 to ensure compliance to the reasoning suite (T-A5:O1), to the iteration objectives (T-A5:O2), and to the model proxy function creation method selected in action T-A5:A1. Although raw data gathering is formalized by a set of templates, the fundamental role of valid data calls for an additional explicit check [319], as a model proxy function derived from “wrong” data tends to produce misleading results, and a “major challenge in model building is collecting sufficient data” [319, p. 3]. Thus, action T-A5:A5 uses checklist template T-C4 - Examine gained raw data, which is depicted in Template element 6.29. It formalizes a set of aspects gathered raw data (T-A5:O3) should comply to, and stores validated data in T-A5:O4.

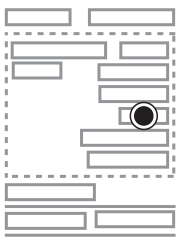
T-A5:A6 Creates the model proxy function based on the evaluated raw data in T-A5:O4, using the method selected in action T-A5:A1.

Opt.	Explanation and implications
I1	**Data gain** – Addresses the different data gaining methods, a proxy function creation method might require. ↦ (Direct) measurements in action T-A5:A3 are fast and mostly accurate, but require instrumentation (cf. Section 5.5), and are only applicable to (physically) existing systems [132]. ↦ Especially full-system simulation in action T-A5:A4 is extremely slow compared to measurements, and cannot be used with long applications and large data-sets [132]. ↦ Simulation requires an IT infrastructure model and an application model in action T-A5:A4, which both might be difficult or expensive to create.
I2	**Function derivation** – Covers the concrete execution of the selected function creation method. ↦ Some approaches pose strong requirements on the stakeholder's experience and skills in action T-A5:A6, e.g., polynomial curve fitting requires a polynomial curve, whose thorough creation tends to be a challenging task.
I3	**Function use and employment** – Addresses the use of the created proxy function. In particular, it deals with the "ceteris paribus" rule, stating that all configuration parameters should stay the same between function creation and use. ↦ Predictions based on data derived functions are only reasonable if it is guaranteed that the basic conditions are the same for both [375], the function creation in activity template T-A5, and for reasoning execution in action T-A3:A19. ↦ Although the created model proxy function is required to comply to the specifications in T-A5:O1 and T-A5:O2, it tends to influence the available reasoning capabilities in *Phase C*. Especially the range of interest, and potential extrapolation is relevant to action T-A3:A19.

Template element 6.28: Implication table of T-D12 - Select model proxy function creation method.

- ☐ **Quantity** – In case the current iteration (cf. action T-A3:A11) processes an attribute concept, it must be ensured that the gathered raw data applies or matches the attribute concept's quantity as specified in T-S4:⟨Quantity⟩, and that gathered raw data exposes at least the required accuracy as also specified in T-S4:⟨Quantity⟩.
- ☐ **Data processing** – Gathered raw data can be processed by the function derivation method selected in action T-A5:A1. Although this is specified in several places, e.g., in the IT infrastructure attribute concept definitions, this must be verified, since function creation in action T-A5:A6 will fail otherwise. For instance, applying regression requires cardinal values (↗KB p. 268).
- ☐ **Quality of the fit** – A prevalent indicator for data gathering problems are outliers [319]. A first impression for data quality can be the obtained by analyzing potential derivation results, e.g., considering the root mean squared error [319].

Template element 6.29: Checklist of T-C4 - Examine gained raw data.



6.2.5 A14 - Examine model behavior

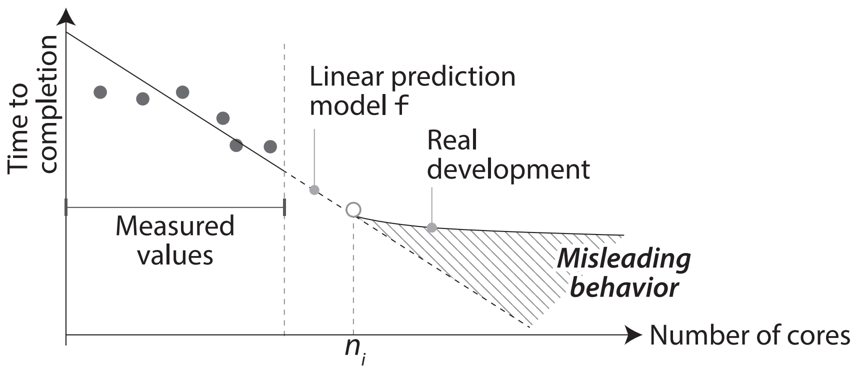
At the current state of the iteration's execution, a (mathematical) model is stored in T-A3:O7 (cf. Template element 6.21). This (mathematical) model was either evaluated for a set of criteria during its selection in action T-A3:A12, or it was derived from raw data by a formalized activity and time-tested methods in action T-A3:A13. Thus, the (mathematical) model is likely to be technically sound and thoroughly created. Nevertheless, the complexity of IT infrastructures and attribute influencing factors (cf. Section 2.4) require an additional explicit examination of the model behavior before its incorporation in the reasoning function in order to identify and react to potential model restrictions or misleading results (*EG-6.12:1*) [118].

→ T.E. 6.21 (p. 175)

Possible identification approaches range in several dimensions, e.g., the granularity level, the applied methodology, the focus, or the covered problem causes (*EG-6.12:2*). This variety and the simultaneous specialization of approaches prohibit a predefined activity compressing the required validation tasks in a limited set of actions. Thus, action T-A3:A14 addresses this situation by decoupling model investigation in two consecutive steps:

Examine model behavior using checklist template T-C5, and

React to potential problems using decision template T-D13.



EG-6.12

(1) The figure above illustrate a fictive performance model that describes *Time to Completion* (TTC, cf. Section 2.4.2) for an arbitrary application in correlation to the number of cores in an HPC cluster system (cf. Section 2.2.2). The measured values on the figure's left hand side indicate a linear correlation of number of cores and TTC reduction, which is derived to a linear prediction model f by regression. The identified linear correlation is valid for up to n_i cores. For more than n_i cores, the TTC doesn't decline linearly as modeled, but tends to a static value, resulting in a misleading model behavior. *(2)* A coarse-grained possibility to identify this behavior is boiling down the problem to Amdahl's Law [172], a fine-grained possibility constitutes the approach of Sharapov et al. [362] who present a process describing how workload exploits parallelism. A more analytical approach present Nussbaum et al. [299], splitting scalability in algorithmic and architectural aspects.

Examine model behavior

Checklist template T-C5 - Examine model behavior summarizes a set of aspects to examine before the model is incorporated in the reasoning function in action T-A3:A15. The checklist template, depicted in Template element 6.30, must mandatorily be applied on both, models selected in action T-A3:A12, and model proxy functions created in action T-A3:A13, due to two reasons:

- Although most models are thoroughly developed and technical sound, validation might be incomplete or not appropriate for the reasoning project. For instance, validation might use workload that differs in the exposed characteristics, compared to the workload selected in action T-A3:A6.
- Model proxy functions are created using plain mathematical/statistical methods without a thorough consideration of computer science related aspects, like application scaling behavior, which must be covered, as well.

- ☐ **Scaling behavior** – Analyze the behavior of the model for increasing and decreasing values of relevant characteristics. These characteristics can cover message size (*EG-6.13:1*), workload variation pace (*EG-6.13:2*), and the problem size (*EG-6.13:3*).
- ☐ **Lower/Upper limits** – Investigate extreme areas of the model, by analyzing the lower and upper limits of the recommended or used value ranges, e.g., using reference implementations [362].
- ☐ **Problem decomposition** – Split the workload in suitable smaller problems using analytic tools, e.g., split an application in its parallel and sequential part according to Amdahl's Law [172], and analyze the transitional points, if not already done by the model.
- ☐ **Influencing factors** Identify and separate influencing factors. An important one is system noise, defined as "operating system activity that negatively impacts the processing capability [317]" [112, p. 2].

Template element 6.30: Checklist of T-C5 - Examine model behavior.

(1) Hoisie et al. [192] found out that the message size of communicating elements of a parallel application strongly impacts the application performance. In particular, if "the message size is less than half the peak bandwidth of the network, it will be latency bound, if it is larger [...], the bandwidth will dominate" [192, p. 4]. They underpinned this relation empirically, stating that heavily communicating applications like SAGE (cf. Appendix B) run slower on the full Blue Gene/L machine, compared to Red Storm or Purple (cf. Section 7.4.5). *(2)* Ge et al. [162] analyzed that the PAST algorithm "works well for slowly varying workloads but incurs large performance and energy penalties for volatile workloads" [162, p. 21]. *(3)* The *Numerical Aerodynamic Simulation* (NAS) Parallel Benchmark (cf. Appendix B) provides several problem sizes to enable benchmark adaptions to the considered machine(s) [29, 162]. Some applications even provide different scaling modes, e.g., "applications were run in weak scaling mode, that is where the problem size per processor remains the same independent of the number of processors used" [112, p. 5]. In contrast, the problem size could grow proportionally with the number of processors [112]. *EG-6.13*

React to potential problems

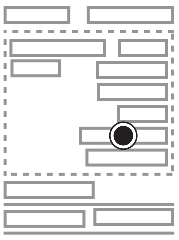
Decision template T-D13 – Select reaction to identified model behavior issues supports the selection of an appropriate reaction to an identified problematic model behavior. It consists of two elements:

- **Implication table** Lists the two reaction options to a potentially identified problematic model behavior, as depicted in Template element 6.31.
- **Documentation** Uses the provenance information model classifiers of related reactions: narrowing the range (T-D13:I1) uses the `attributes` and `reasoningproject` packages, adapting the model (T-D13:I2) affects the `instances` package for IT infrastructure attributes.

Opt.	Explanation and implications
I1	**Narrow range** – The reasoning parameter value range is narrowed to avoid issues in border areas. ↦ A bigger value range tends to cause higher reasoning costs in action T-A3:A19, e.g., due to more optimization runs. ↦ The narrowed range must comply with specifications in the reasoning suite (T-A3:O3), and particularly in sentence template T-S4 and T-S5 about the IT infrastructure attribute concept and reasoning parameter ranges, respectively. ↦ Narrowing the range does not require an additional iteration, as the (mathematical) model in T-A3:O7 remains as it is. ↦ A range adaption must be propagated to T-S4:⟨Objectives⟩ or T-S5:⟨Quantity⟩ of the corresponding attribute concept or reasoning parameter, respectively.
I2	**Modify model** – The examined (mathematical) model in T-A3:O7 is altered in some way, e.g., by incorporating mathematical barriers or noise factors in its formula. ↦ Fostering a clean and reproducible model modification requires an additional iteration of the currently processed iteration objectives, starting at action T-A3:A11.

Template element 6.31: Implication table of T-D13 - Select reaction to identified model behavior issues.

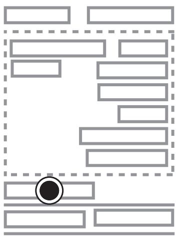
6.2.6 A15 – Incorporate iteration results


→ *T.E. 6.21 (p. 175)*

Action T-A3:A15 incorporates the evaluated (mathematical) model in T-A3:O7 into the reasoning function in two steps:

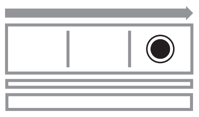
- **Refine reasoning function** Incorporates the iteration result in the already existing reasoning function (skeleton) by replacing a formula element with a more detailed or more accurate one (cf. Section 4.2).
- **Evaluate refinement** Evaluates the achieved reasoning function in two ways: First, analyze the fulfillment of the iteration's objectives, specified in T-A3:O6, by verifying that the refinement addresses all targeted goals. Second, compare the replaced value with values gained by the replacing refinement [242]. In case the values are not within an acceptable range and T-A3:D3 validates to *false*, another iteration is required. In case only a little correction is necessary, the refinement can be enhanced by correction or noise factors, to improve prediction accuracy.

6.2.7 A16 – Evaluate reasoning function


→ *T.E. 6.21 (p. 175)*

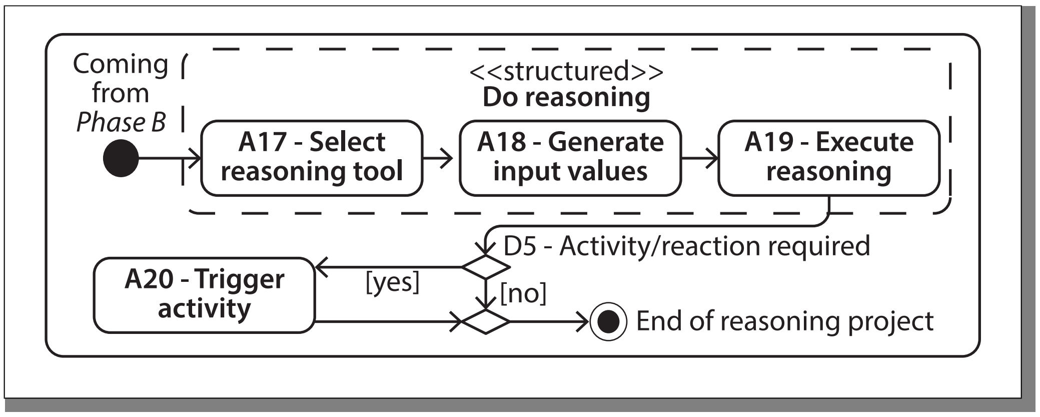
As last task in *Phase B* of activity template T-A3, action T-A3:A16 evaluates the complete reasoning function in T-A3:O8 on its compliance to the reasoning suite (T-A3:O3), and on the support of the IT infrastructure attribute concepts specified in T-S3:⟨AttributeConcepts⟩, and reasoning parameters specified in T-S3:⟨Parameters⟩. The evaluation result is processed by T-A3:D4 that triggers another refinement iteration, if required. Otherwise, the final reasoning function (T-A3:O9), and a detailed object diagram of the provenance information model (T-A3:O10) are passed over to *Phase C*.

6.3 Phase C – Reasoning execution

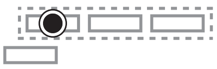

→ *Fig. 6.1 (p. 144)*

As explained in the previous section, *Phase B* of activity template T-A3 compiles an individual reasoning function and stores it in T-A3:O9.

Phase C of activity template T-A3 employs this reasoning function to execute the concrete reasoning, guided by the reasoning suite (T-A3:O3). Compared to *Phase A* and *Phase B*, *Phase C* is rather compact, since the bulk of actions and decisions are already completed and made within the other two phases, respectively. Template element 6.32 depicts the action flow of *Phase C* and emphasizes its aforementioned little extend. The remainder of this section provides the action details of *Phase C*.



Template element 6.32: Action flow of T-A3 - Reasoning methodology - Phase C.



6.3.1 A17 – Select reasoning tool

→ T.E. 6.32 (p. 190)

There is a plurality of potential reasoning tools and configurations to use with the compiled reasoning function. Two aspects reduce this set:

1. Decisions made in action T-A3:A2 might require the (non) consideration of the development over time, and the analysis of a delta between or within attribute(s). Thus, all reasoning tools that are not capable to provide this functionality are disregarded.
2. The Use Case analysis explicitly calls in its sub system *C – Reasoning execution* (cf. Section 3.3.2) for the support of What-if analysis, optimization mechanisms, and descriptive statistics. Consequently, those reasoning tools must be contained in the tool set.

Action T-A3:A17 focuses on the second aspect, and uses decision template T-D14 – Select reasoning tool to select one of the three reasoning tools. The decision template consists of three elements:

- **Implication table** Itemizes the three options *descriptive statistics*, *optimization*, and *What-if analysis* in alphabetical order according to the requirements specified in the Use Case sub system *C – Reasoning execution*. Template element 6.33 depicts the implication table.
- **Decision tool** Opposes the reasoning tools according to the three following characteristics, as depicted in Template element 6.34:

  Main focus Describes whether the reasoning tool tends to process the reasoning function's domain or co domain. Optimization, for instance, mainly processes the reasoning function domain and uses the co domain only as constraint source. In contrast, descriptive statistics mainly investigate the reasoning function's co domain and the calculated attribute vector.

Required domain flexibility Considers the required potential to realize reasoning input parameter values in order to achieve reasonable results with the particular reasoning tool. For instance, optimization requires modifications to the IT infrastructure according to optimization results, reflected in modification parameters (cf. Section 4.1). In contrast, descriptive statistics makes no assumptions about the reasoning function's domain.

Difficulty of use Arranges the three reasoning tools according to their employment and especially the difficulty of use. Optimization, for instance, requires an *objective function* (↗KB p. 262), whose normally difficult formulation renders optimization in general a challenging intent. In contrast, What-if analysis tries (more or less) different domain values and considers the resulting attribute vector, which is a comparatively easy task (↗KB p. 276). Obviously, the level of difficulty highly depends on the particular skills, experience, and demands, but a basic arrangement is still possible.

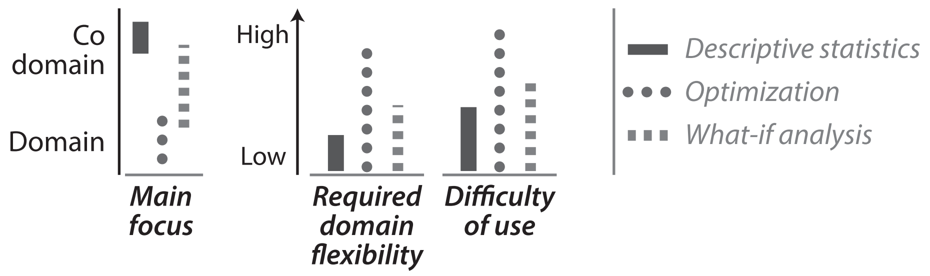
- **Documentation** Uses the class `ReasoningTool` in the provenance information model package `reasoningproject` (cf. Section 5.3.4).

(1) Exemplary descriptive statistics scenarios analyze the distribution of power consumption values in correlation to the number of nodes in a cluster, or the derivation of a sensitivity measure describing the correlation between an attribute and a modification. For instance, in case the mean has a low distribution, it could be induced that the modification has a small effect on the attribute, as shown by Carrington et al. [83] for the performance of the *Parallel Ocean Program* (POP). *(2)* An exemplary situation that contradicts the use of optimization is a procurement decision, since there is only a limited set of available hardware that might not be capable of providing the optimization result. Hence, although optimization might result in a fictive option A, only the options B and C might be available on the market at that point in time.

EG-6.14

Opt.	Explanation and implications
I1	**Descriptive statistics** – Use the tool set of descriptive statistics (↗KB p. 268). ↦ Facilitates the mathematical prove of dependencies in action T-A3:A19. ↦ Has nearly no influence on the reasoning function's domain and does not put any requirements on the reasoning input parameters. Hence, there is probably no (technical) implementation required in action T-A3:A20.
I2	**Optimization** – Use the tool set of optimization (↗KB p. 262). ↦ Requires the possibility to alter the domain values in reality in action T-A3:A20, since otherwise optimization results cannot be implemented or used (*EG-6.14:2*). ↦ Requires the creation of an objective function in action T-A3:A19, which tends to be a challenging task (↗KB p. 262). ↦ Requires the definition of constraints in action T-A3:A19. ↦ Fosters automated reasoning, since all elements can be encoded in a computer readable way, e.g., the feasible region and the objective function.
I3	**What-if analysis** – Use the tool set of What-if analysis (↗KB p. 276). ↦ Enables the comparison of a (small) set of reasoning function input vectors in action T-A3:A19.

Template element 6.33: Implication table of T-D14 - Select reasoning tool.



Descriptive statistics Mainly operates on the computed attribute vectors – the reasoning function's co domain – and does not pose any requirements on the reasoning function's domain. Besides, already basic functionality, like a sample standard deviation (↗KB p. 268), can compile useful results, what renders the employment of descriptive statistics comparatively easy (*EG-6.14:1*).

Optimization Selects a sub set of a candidate set – the reasoning function's domain – according to an objective function and a set of constraints [168] – the reasoning function's co domain (↗KB p. 262). An optimization result describes the "best" set of modification and configuration parameters (cf. Section 4.1). This result, in turn, should be realizable, e.g., by accomplishing a modification to achieve the computed parameter (*EG-6.14:2*), since otherwise the optimization result is useless, what poses (strong) demands on the reasoning function's domain. The usually generic way of formalizing optimization problems bans the provision of good or consistent algorithms. Instead, there is a set of optimization problem classes that help to find and formulate an appropriate algorithm (↗KB p. 262).

What-if analysis Focuses on both, the reasoning parameters and attribute concepts, as it tries differing input vector combinations and analyzes the results. The theoretical infinite set of analysis possibilities strongly requires a thorough selection of input vectors what tends to be a difficult task. In contrast to optimization, What-if analysis does not pose any requirements on the domain and is well suited for procurement consideration, as different offers can be compared.

Template element 6.34: Decision tool of template T-D14 - Select reasoning tool.

6.3.2 A18 – Generate input values

→ *T.E. 6.32 (p. 190)*

Phase A of activity template T-A3 defines and provides most input values of the reasoning function, either explicitly as static (numeric) values, or implicitly as value ranges that should be analyzed, depending on action T-A3:A2, T-A3:A3, and T-A3:A4. Nevertheless, additional input values might be required, e.g., due to constraint definitions in action T-A3:A8. Besides, despite the selection of appropriate workload in action T-A3:A6, load values might still be missing, e.g., because workload execution was no necessary for measurements. Action T-A3:A18 covers both situations: for missing input values, it provides a dedicated scope for their generation; for missing load values, it dictates required generation tasks.

The great extend of the load value generation problem domain (cf. Section 2.3.3), the mostly close coupling of generation approaches to a specific application or hardware, and the individually required granularity level render an activity template or sentence template unsuitable (*EG-6.15*). Instead, action T-A3:A18 accomplishes this task in two steps, namely it *selects a load value generation approach* using decision template T-D15, and it *evaluates generated results* using checklist template T-C6.

EG-6.15

Murphy [288] is an example of a high granularity level for differing load values depending on the executed code. He empirically evidences that integer codes perform 15% fewer memory references and perform twice as many branches as their floating point counterparts. Implicitly, this also illustrates the fundamental influence of the considered workload, as scientific code tends to be built of larger building blocks due to complex formula calculations[338]. Another approach applying a high granularity level can be found in [5] that provides an approach for calculating the expected load down to a single core, the memory input and output bandwidth, and the system bus throughput. An example highlighting the important role of the applied granularity level provide Li et al., stating that load characteristics include “system utilization, job arrival rate and interarrival time, job cancellation rate, job size (degree of parallelism), job runtime, memory usage, and user/group behavior” [253, p. 177].

Select load value generation approach

Decision template T-D15 – Select load value gathering method formalizes the selection of a load value generation approach, and consists of two elements:

- **Implication table** Compares trace analysis, simulation, and convolution (cf. Section 2.3.3), as depicted in Template element 6.35.

- **Documentation** Uses the `workload` package of the provenance information model detailed in Section 6.1.6.

Opt.	Explanation and implications
I1	**Trace analysis** – Execute workload while capturing information in trace files (*EG-6.16:1*) using profiling libraries or applying measuring activity template T-A2. ↦ Requires instrumentation of hardware and software [17] (*EG-6.16:2*) in action T-A3:A19. Although most vendors provide pre-installed libraries, and despite the existence of (semi) automated instrumentation (*EG-6.16:3*), this might cause additional installation and configuration costs. ↦ Requires the execution of the workload to enable capturing (important) information in trace files [45]. Workload is used as selected in action T-A3:A6. If workload execution is not possible, a proxy workload is selected using decision template T-C3 (cf. Section 5.7). ↦ Depends on the quality of the data. ↦ Requires the thorough documentation of applied configuration and deployment parameters (cf. Section 2.3.2).
I2	**Simulation** – Imitates a system as it progresses through time (cf. Section 7.4.4). ↦ Especially full-system simulations tend to last very long. ↦ Requires mandatorily a *concept model*, whose creation tends to be a challenging task. ↦ Benefits from the plurality of existing simulation tools.
I3	**Convolution** – Computationally maps an application signature onto a machine profile (cf. Section 7.4.3). ↦ Requires a *Machine Profile* and an *Application Signature*, e.g., using MetaSim [84]. ↦ Benefits from the plurality of existing tools (cf. Section 7.4).

Template element 6.35: Implication table of T-D15 - Select load value gathering method.

EG-6.16

> *(1)* Ge et al. [162] use history-based workload prediction: during execution, they "collect a set of performance events and summarize them [...], and predict [...] the future workload using the history values [...]" [162, p. 22], e.g., by employing the PAST prediction algorithm [413]. *(2)* A wide-spread tool for instrumentation and information capturing is AutoTune providing a set of annotations [283]. *(3)* Alkindi et al. [17] enhance the performance prediction process in PACE [297] by utilizing dynamic/automatic instrumentation by inserting special sensors in an application source code.

Evaluate generated results

Checklist template T-C6 - Examine load value generation guides load value generation and result evaluation: it itemizes aspects and information that must be respected or gathered while creating load values as input variables. Noteworthy, checklist template T-C6 is independent of the used load value generation approach, that might be selected by decision template T-D15, or that might be a proprietary one. Checklist template T-C6 is depicted in Template element 6.36, and exposes the following characteristics:

- Items are ordered according to Barker et al. [31], who start load examination with the workload's main data structure, followed by its distribution among and communication between parallel processes. Hence, after general characteristics, checklist template T-C6 considers workload decomposition and communication, before building blocks and correlations are examined. This notion also complies with Jha et al. [212], who determine what is distributed, how the pieces interact, and what the flow of control and information between the pieces is.
- Items aim at providing a generic list, instead of stating concrete values, in order to support a wide applicability (cf. Section 4.3), e.g., communication patterns are valid for OpenMP, MPI, and Grid-wide communication.

6.3.3 A19 – Execute reasoning

→ *T.E. 6.32 (p. 190)*

Action T-A3:A19 uses the reasoning tool selected in action T-A3:A17 to execute the concrete reasoning. Execution is explicitly placed in a dedicated action and not implicitly contained in the reasoning tool selection to enable and foster task delegation, since (potentially) different roles could be assigned for reasoning tool selection and reasoning execution, respectively. In contrast, action T-A3:A19 does not provide formalization or templates, because the

- □ **General characteristics** – Cover general (meta) information about the job, e.g., its size, number of requested processors or its runtime [253].
- □ **Decomposition** – Describe a workload's decomposition in building blocks and how the building blocks utilize parallelism. This is correlated to the scaling behavior of the workload, because it implies how the problem size per processor behaves.
- □ **Communication** – Cover the exchanged data and communication patterns of (atomic) workload building blocks. For instance, particle-in-cell codes normally require more data movement per computation unit than dense matrix calculations or Monte Carlo simulations [31]. For parallel workloads, the exchanged data's amount and type significantly affect coherency traffic [93].
- □ **Memory use** – Consider the memory use and access patterns [412] of the entire workload as well as single building blocks. A concrete example is the correlation of the job size and the memory usage per processor [253].
- □ **Execution environment** – Cover the environment of the executed workload, e.g., is foreign workload executed at the same time.
- □ **Correlations** – Describe correlations between the above itemized aspects like memory usage vs. job size or actual run time vs. job size [253], which can be easily checked using Pearson's R correlations or Spearman's rank correlations [253].
- □ **Scaling** – Consider scaling behavior as carried out in action T-A3:A14.

Template element 6.36: Checklist of T-C6 - Examine load value generation.

use of the selected reasoning tool(s) is situation specific, and relevant rules are provided for selection in action T-A3:A17.

6.3.4 A20 – Trigger activity

→ *T.E. 6.32 (p. 190)*

Certain reasoning results might call for a particular activity, as described in Use Case UC-6 (cf. Section 3.3.2). Action T-A3:A20 covers the handling and execution of this activity. Since the concrete constitution of the activity highly depends on the situation, the objectives of the management, and the available resources, action T-A3:A20 does not provide any details, but purely acts as umbrella to enable a clear arrangement of activity execution in the context of the process model, and a reasoning project. The manifestation of this action is discussed in Section 8.2.6 as part of Future Work.

6.4 Summary

The chapter at hand details the process model's reasoning methodology. Reflecting the identified three Use Case sub systems (cf. Section 3.3.2), and realizing the process model underlying design concepts (cf. Section 4.2), the methodology consists of the following three phases:

- **Phase A – Reasoning suite definition** Identifies and specifies reasoning objectives and related information to facilitate the reasoning objectives driven refinement of the individual reasoning function (cf. Section 4.3). The phase assembles two results:
  - A *reasoning suite* that formalizes essential information to steer the reasoning project, and to dictate (partial) result validation guidelines.
  - A *reasoning function skeleton* that is processed by *Phase B*.
- **Phase B – Reasoning function compilation** Guided by the reasoning suite, the phase iteratively refines the given reasoning function skeleton to a fully functional reasoning function (cf. Section 4.2).
- **Phase C – Reasoning** Uses the compiled reasoning function to execute the reasoning.

Chapter 7

Rigor cycle – Validation of the process model

This chapter validates the presented process model for the integrated reasoning about quantitative IT infrastructure attributes. Section 7.1 explains validation objectives and describes the applied validation methodology, before Sections 7.2, 7.3, and 7.4 perform the validation applying multiple validation approaches. Collectively, the three sections cover the *Rigor Cycle* and the *Relevance Cycle* of the applied *Design Science* paradigm framework (cf. Section 1.4).

7.1 Validation objectives and methodology

According to the general demand of validating scientific results in "some logical, objective, and algorithmic way" [231, p. 1089], the *Design Science* paradigm framework underlying this thesis (cf. Section 1.4) mandatorily postulates the "application of rigorous methods in the [...] evaluation of the design artifact [...] to rigorously demonstrate [its] utility, quality, and efficacy" [187, Table 1, p. 83]. For this intent, the framework provides a taxonomy of evaluation methods (↗KB p. 253), based on Zelkowitz et al. [434, 435] who surveyed hundreds of IT research papers [235]. The following itemization overviews the methods applied to validate the presented process model (cf. Figure 7.1). It particularly explains method intentions, introduces implementations, and provides a rationale for their selection, respectively.

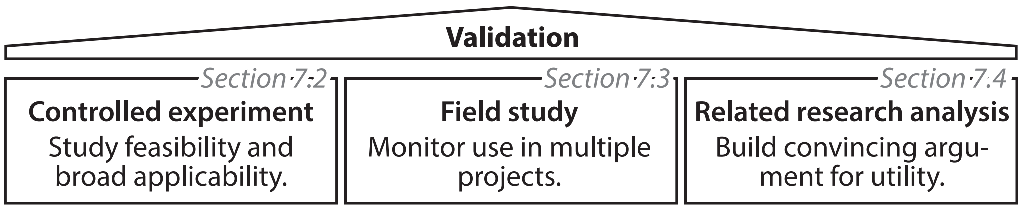

Validation
Section 7.2
Controlled experiment
Study feasibility and broad applicability.
Section 7.3
Field study
Monitor use in multiple projects.
Section 7.4
Related research analysis
Build convincing argument for utility.

Figure 7.1: Methods applied to validate the presented process model.

- **Controlled experiment** The *experimental* method "studies [the process model] in [a] controlled environment for qualities" [187, p. 86], especially for feasibility, and for broad applicability.

  Implementation Conducts a reasoning project about the power consumption and performance of an experimental *Raspberry Pi* (RPi) cluster to compare theory and reality [56] in Section 7.2.

  Rationale The validation intent requires (full) control of and exclusive access to the examined IT infrastructure. The former eliminates bias and side effects caused by (uncontrolled) influencing factors, like the simultaneous execution of foreign applications, the latter enables installation of potentially required measuring instruments. Both is seldom possible and never guaranteed for production systems like the SuperMUC (cf. Section 3.2.1). In contrast, the used RPi cluster was built at the chair for experimental purposes and is under full control.

- **Field study** The *observational* method "monitor[s] use of [the process model] in multiple projects" [187, p. 86] to validate the process model's fulfillment of "requirements [...it was] meant to solve" [187, p. 85].

  Implementation Contrasts the process model's capabilities with the (non) functional requirements listed in the *Requirements Specification* (RS) (cf. Section 3.5) in Section 7.3.

  Rationale Requirements analysis in Chapter 3 thoroughly examines the research *Environment* and particularly identifies requirements in the scenarios *SuperMUC* and *DRIHM*. These requirements are used to validate the process model's problem solving character, and to apply the process model (theoretically) in its application domain.

- **Related research analysis** The *descriptive* method "use[s] information from [related research] to build a convincing argument for the [process model's] utility" [187, p. 86], and to underpin non-commutability and advance of the process model.

Implementation Compares the process model with related research, split in 1) work dealing with reasoning in the context of quantitative IT infrastructure attributes, and 2) work potentially providing partial contributions, like IT infrastructure modeling, in Section 7.4.

Rationale The high relevance of IT infrastructure attributes cause a plurality of specialized (and mature) models. In order to provide "convincing arguments" for the process model's utility, its functionality and characteristic are compared to existing approaches.

7.2 Controlled experiment

→ *Fig. 7.1 (p. 200)*

The section constitutes the first validation part (cf. Section 7.1), being an *experimental* validation method that analyzes IT artifacts in a "controlled environment for qualities" [187, Tab. 2]. It conducts a reasoning project using the presented process model. The reasoning project must be understood as a prototypical instance, a "partial, deliberate incomplete implementation [...] that serves as demonstrator of selected [...] characteristics in practical use, for experimental purpose, and for collecting practical experience" [152]. The prototype targets two characteristics of the presented process model:

- **Feasibility** Illustrate that reasoning about quantitative IT infrastructure attributes using the presented process model is feasible and productive, particularly that the IT artifacts (cf. Chapter 5) and the methodology (cf. Section 6) can be instantiated. Hence, the controlled experiment conducts a complete prototypical reasoning project. Since it aims at showing the process model's feasibility and not at answering a reasoning question under "real world" conditions, the reasoning project represents a *possible* way, but not inevitably the *best* way. For instance, it shows that model selection is feasible, but it purposely does not aim at selecting the most appropriate or most accurate model, as this wouldn't make any difference for the general feasibility.

- **Applicability** Underpin the process model's applicability not only to the analyzed scenarios in Chapter 3, but also to completely different IT infrastructures. Hence, the controlled experiment conducts a prototypical reasoning project on a small cluster of Raspberry Pis, a credit-card sized computer run by an ARMv11 microprocessor. For this cluster that has been assembled at the author's chair, the prevalent attributes *power consumption* and *performance* are considered (cf. Section 2.4).

Features and qualities of the process model that are not of primary interest for the prototype's intent and particularly for the two characteristics are omitted. In particular, this applies to accuracy, as it confirms neither the feasibility nor applicability of the process model. Instead, it depends on the concrete making and execution of several decision and actions, like model selection or measuring, as comprehensively carried out in Chapter 5 and 6. Generally speaking, the disregard of single features and qualities has no effect on the prototypical instantiation of the process model, since they do not reflect conceptual weak points, but implementation details.

The reasoning methodology's three phases (cf. activity template T-A3) structure the description of the controlled experiment: Section 7.2.1, 7.2.2, and 7.2.3 describe *Phase A*, *Phase B*, and *Phase C* of the reasoning methodology, respectively. For the sake of compactness, description omits introductory paragraphs or explanations. Instead, it confines to the pure process model's execution and provides focused notes where suitable.

7.2.1 Phase A – Reasoning suite definition

The phase identifies, collects, and specifies relevant information that guides and steers a reasoning project. It results in a *reasoning suite*, containing the specified information, and in a *reasoning function skeleton*, which is processed by the following *Phase B*.

T-A3:A1 – Prepare reasoning suite

The reasoning project considers as IT infrastructure a RPi Beowulf cluster (cf. Section 7.1). Two reasons explain the selection:

1. Its assembly at the author's chair guarantees an exclusive experimental setup, and being under full control, as controlled experiments prerequisite.
2. The investigation of Raspberry Pi clusters by other researchers, like the "Iridis-Pi" system of Cox et al. [105], substantiates the validation of the process model's applicability on relevant IT infrastructures.

The reasoning project deals with power consumption and performance as a provider and consumer perspective attribute (cf. Section 2.1.1), respectively. The reasoning project aims at answering the fictive question "Would additional Raspberry Pi nodes increase the cluster's performance and would the performance increase outperform the expected power consumption increase?". The intent is formalized in an instance of sentence template T-S3.

Sentence template T-S3 instance
Reason about ⟨*trend of the delta*⟩ of ⟨*Power Consumption, Performance*⟩ in ⟨*RaspberryPi cluster*⟩, executing ⟨*STREAM benchmark, HPL benchmark*⟩, applying ⟨*Nodecount*⟩, assuming ⟨*ClusterExpandability*⟩, providing ⟨*NoLicences*⟩.

T-A3:A2 – Select reasoning interests

Using the selection tool of decision template T-D3, the reasoning project's interest is defined as *trend of the delta.*

T-A3:A3 – Define attribute concepts

Sentence template T-S4 instance
The attribute concept ⟨*Power Consumption*⟩ describes ⟨*the overall power consumed for a complete workload run*⟩ in ⟨*Watt at* ±2%⟩ of ⟨*RPi cluster*⟩ provided that ⟨*not used nodes are not powered*⟩.

Sentence template T-S4 instance
The attribute concept ⟨*Performance*⟩ describes ⟨*a particular workload's time to completion (TTC)*⟩ in ⟨*seconds at* ±2%⟩ of ⟨*RPi cluster*⟩ provided that ⟨*the workload is executed in parallel*⟩.

Setting T-S3:⟨Interest⟩ to *trend of the delta* in action T-A3:A2 requires the use of metric scales for both attribute concepts according to the implication table of decision template T-D3. Both attribute concept definitions produce the entry *RPi cluster* for the component type set T-A3:O1.

T-A3:A4 – Define reasoning parameters

Sentence template T-S5 instance
Reasoning parameter ⟨*Nodecount*⟩ describes ⟨*the number of powered RPi nodes in the cluster*⟩ given in ⟨*natural numbers between 1 to 20*⟩.

The reasoning parameter adds the entry *RPi* to the component type set T-A3:O1, resulting in T-A3:O2.

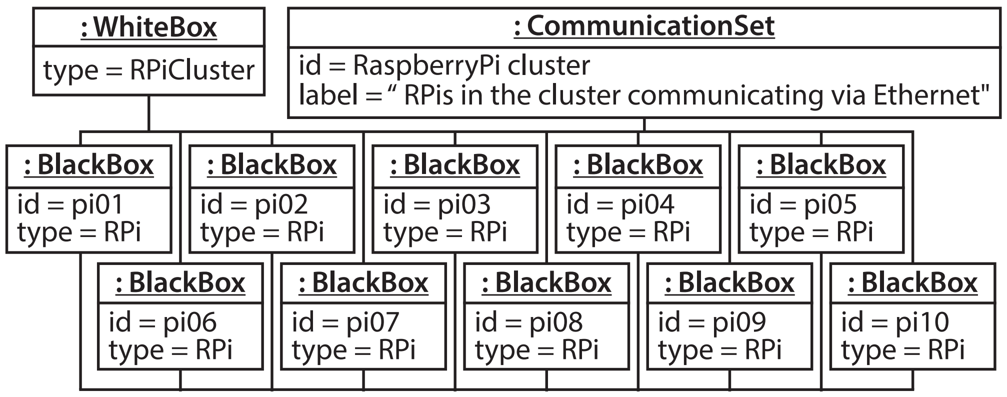
T-A3:A5 – Model IT infrastructure

Section 7.1 introduces the reasoning project's IT infrastructure as an experimental RPi cluster. It consists of 10 RPi Type B nodes, each running an ARMv11 microprocessor and 512 MB memory.[1] The nodes are arranged and assembled as Beowulf cluster (cf. Section 2.2.2). Activity template T-A4 creates the IT infrastructure model documented in the form template T-F4 instance below according to the provided component type setT-A3:O2.

F1 – Unique ID

RPi cluster

F2 – IT infrastructure model



Form template T-F4 instance.

Obviously, the IT infrastructure model could be theoretically further refined, e.g., adding a RPi's microprocessor and memory. In contrast, the IT infrastructure model is strictly limited to the component type set T-A3:O2, and RPis remain black boxes, according to reasoning objectives driven refinement (cf. Sections 4.3 and 6.1.5).

T-A3:A6 – Select workload

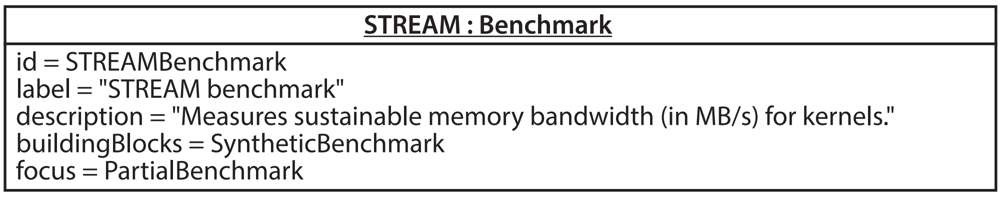
The controlled experiment's objective of validating general applicability calls for wide-spread and commonly used workload. The selection tool of decision template T-D9 selects the STREAM and HPL benchmarks (cf. Appendix B), accordingly. Two form template T-F5 instances detail the workload below.

[1]For a detailed hardware description the reader is referred to `www.raspberrypi.org`, Cox et al. [105], and Klinger [234] who built the considered cluster.

F1 – Unique ID

STREAM benchmark

F2 – Workload model



STREAM : Benchmark

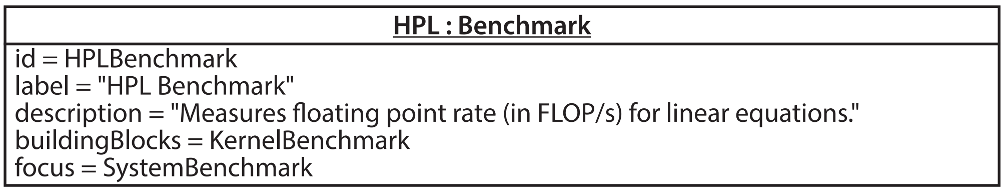
id = STREAMBenchmark
label = "STREAM benchmark"
description = "Measures sustainable memory bandwidth (in MB/s) for kernels."
buildingBlocks = SyntheticBenchmark
focus = PartialBenchmark

Form template T-F5 instance.

F1 – Unique ID

HPL benchmark

F2 – Workload model



HPL : Benchmark

id = HPLBenchmark
label = "HPL Benchmark"
description = "Measures floating point rate (in FLOP/s) for linear equations."
buildingBlocks = KernelBenchmark
focus = SystemBenchmark

Form template T-F5 instance.

T-A3:A7 – Document assumptions

F1 – Unique ID

ClusterExpandability

F3 – Assumption

It is assumed that the number of powered RPi nodes in T-S5:⟨Nodecount⟩ can be adjusted, since it is defined as configuration parameter.

Form template T-F6 instance.

T-A3:A8 – Document constraints

F1 – Unique ID

NoLicences

F3 – Constraint

The prototypical and not economic nature of the reasoning project requires a cost-free access to all involved artifacts, particularly to integrated models.

Form template T-F7 instance.

T-A3:A9 – Create reasoning function skeleton

$$f(\underbrace{Nodecount}_{\text{Modification parameter}}, \underbrace{Benchmark}_{\text{Configuration parameter}}) = \begin{pmatrix} PowerConsumption \\ Performance \end{pmatrix} \tag{7.1}$$

7.2.2 Phase B – Reasoning function compilation

The phase transforms the given reasoning function skeleton to a fully functional reasoning function in several iterations (cf. Section 4.2). According to the prototypical nature of the discussed reasoning project and for the sake of compactness, only one iteration is executed and explained.

T-A3:A10 – Define iteration objectives

Sentence template T-S6 instance
Iteration aims at ⟨*adding*⟩ of ⟨*Benchmark, Power Consumption*⟩ in the reasoning function.

T-A3:A11 – Operationalize single objective

F1 – Unique ID
addNodeCountPerformance
F2 – Objective
Add the ⟨*Benchmark*⟩ configuration parameter and the attribute ⟨*Power Consumption*⟩ in one step, since they are likely to use the same model or measuring setup. The iteration considers a fixed number of 10 nodes in the RPi cluster, since the ⟨*Nodecount*⟩ parameter is added in the next iteration.

Form template T-F8 instance.

T-A3:A12 – Select existing model

The operationalized iteration objectives call for a model that predicts the power consumption of the Raspberry Pi cluster for a STREAM or HPL benchmark run. According to the prototypical nature of the reasoning project, only a rough inquiry employing decision template T-D11 was conducted, resulting in no suitable model integration candidate.

T-A3:A13 – Create model proxy

Since action T-A3:A12 does not provide any model integration candidate, activity template T-A5 is executed to create a model proxy:

T-A5:A1 Use decision template T-D12 to chose simple regression (↗KB p. 268) as function creation method, a wide-spread approach that is also applied by other work, e.g., Lee et al. [250] dynamically predict performance and power using regression models [71].

T-A5:A2 Execute the STREAM and HPL benchmarks.

T-A5:A3 Execute activity template T-A2 to gather raw data for the chosen regression by measurement:

> T-A2:A1 Define measuring objectives using sentence template T-S4. The template instance is shown below.
>
> T-A2:A2 Select the *Energenie EGM-PWM-LAN* measuring instrument employing decision template T-D2.
>
> T-A2:A3 Design the measuring system in the style of the Green500 list measuring system (cf. [363, Fig. 1]) using checklist template T-C2.
>
> T-A2:A4 Document the measuring system in the form template T-F2 instance shown below (is analogue for HPL benchmark).
>
> T-A2:A5 Execute the measurement and document gained results. Figure 7.2 depicts fictive measurement results as box plot. According to the iteration's objectives in the form template T-F8 instance above, the depicted chart shows a value range for each workload execution, respectively.

T-A5:A5 Evaluate the raw data gained by the measurement.

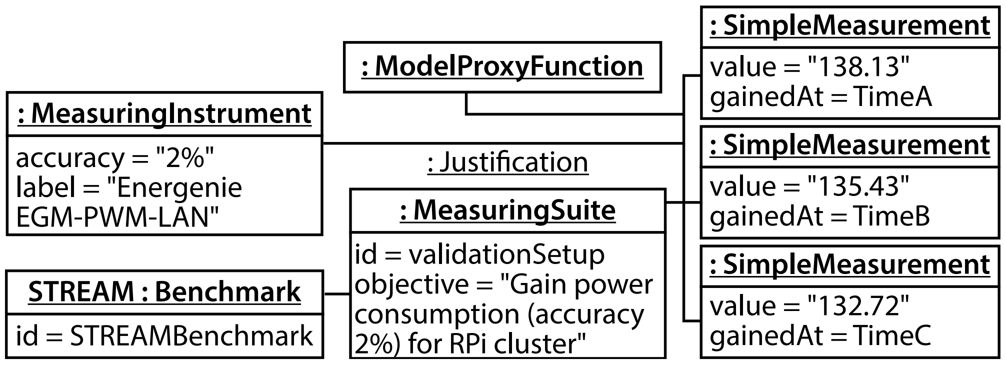
T-A5:A6 Derive the regression function in Equation 7.2 from the raw data.

Sentence template T-S4 instance
Gain ⟨*power consumption*⟩ with an accuracy of ⟨±2%⟩ on ⟨*RPi cluster*⟩ using ⟨*MeasurePowerConsumptionOfCluster*⟩.

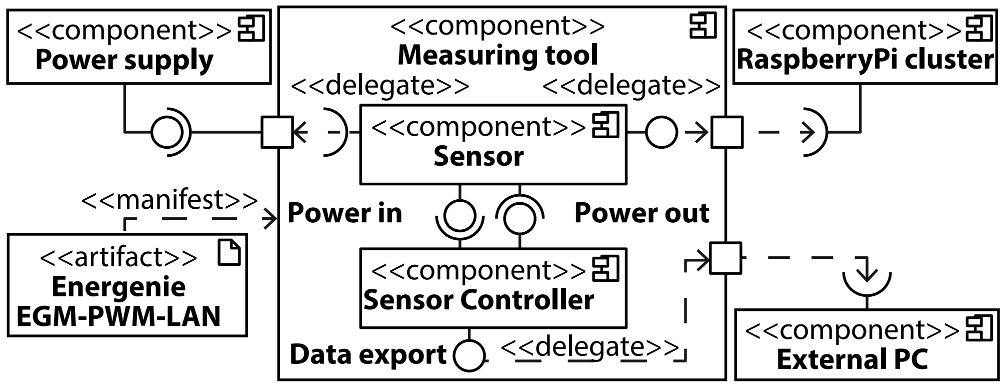
F1 – Unique ID

MeasurePowerConsumptionOfCluster

F2 – Measuring information model



F3 – Physical design



Form template T-F2 instance.

$$power(Benchmark) = \begin{cases} 135.43 & \text{if benchmark=STREAM} \\ 143.60 & \text{if benchmark=HPL} \end{cases} \tag{7.2}$$

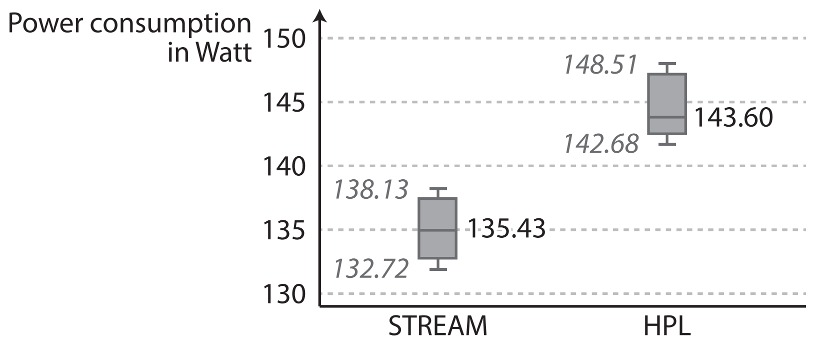


Figure 7.2: Box plot of fictive power consumption measurements of the RPi cluster while executing the STREAM an HPL benchmark.

T-A3:A14 – Examine model behavior

This iteration does not expect any unpredictable model behavior, because it processes only two measured values for two benchmark executions on a fixed number of nodes. Hence, there are no scaling or other aspects that might induce model anomalies (cf. Section 6.2.5). In contrast, later iterations will use checklist template T-C5 to evaluate the model behavior, e.g., adding the $\langle Nodecount \rangle$ parameter could be evaluated using the high level specifications (cf. Section 2.3.2) of both benchmarks to assess the function's scaling and extrapolation behavior. Adding the $\langle Performance \rangle$ vector item can use results of other research groups, like Cox et al. [105], for comparison.

T-A3:A15 – Incorporate iteration results

Since the results of the currently discussed iteration wouldn't alter Equation 7.2, a further detailing in this direction is omitted. Instead, Equation 7.3 depicts directly the final reasoning function after all iterations are executed: the first vector entry is the power consumption, the second entry is the performance (cf. Equation 7.1). $\langle Power\ Consumption \rangle$ values expose a linear scaling. The $\langle STREAM \rangle$ TTC grows linearly as defined in its high level specification [271], the $\langle HPL \rangle$ TTC grows logarithmic.[2]

[2] As Section 7.2 introductory insistently explains, the reasoning project is a prototypical implementation focusing on aspects of the process model's feasibility and applicability. Thus, the logarithmic and linear TTC scaling is theoretically derived from the high level specifications of the considered benchmarks. In particular, it is not derived from real measured data, as the measuring costs would contradict the prototypical idea. However, this does not affect the validation of the process model's *feasibility* and *applicability*.

$$f(Nodecount, Benchmark) = \begin{pmatrix} 103.76 + \begin{cases} Nodecount \times 4.9 & \text{if benchmark=STREAM} \\ Nodecount \times 3.7 & \text{if benchmark=HPL} \end{cases} \\ 0.2 + \begin{cases} Nodecount \times 1.2 & \text{if benchmark=STREAM} \\ \log(Nodecount) \times 3.4 & \text{if benchmark=HPL} \end{cases} \end{pmatrix} \tag{7.3}$$

T-A3:A16 – Evaluate reasoning function

The currently discussed iteration purely bases upon measurements and the reasoning function assembled by the iteration is consistent with Equation 7.2. The measuring tool employed to compile the reasoning function's version exposes an accuracy of ±2%, which renders an additional evaluation unnecessary, because this value complies to the demands specified in the sentence template T-S4 instance for power consumption (cf. action T-A3:A3). In later iterations, the replacing refinement would be compared to the replaced function element as described in Section 6.2.7.

7.2.3 Phase C – Reasoning execution

The phase employs the compiled reasoning function to conduct the actual reasoning. It selects an appropriate reasoning tool, like optimization or what-if analysis, analyzes different parameter combinations, and triggers a direct reaction on the reasoning results, if necessary.

T-A3:A17 – Select reasoning tool

Recalling the reasoning project's intent in Section 7.2.1, it aims at answering the question "Would additional Raspberry Pi nodes increase the cluster's performance and would the performance increase outperform the expected power consumption increase?". The variable T-S5:⟨Nodecount⟩ limits the amount of nodes to a range of 1 to 20. Due to the range's small size, decision template T-D14 selects What-if analysis as reasoning tool.

T-A3:A18 – Generate input values

For both reasoning function parameters, the modification parameter and the configuration parameter, the value ranges are already predefined by the reasoning suite: action T-A3:A4 limits the amount of nodes to a range of 1 to 20, action T-A3:A6 defines the workload as STREAM and HPL benchmark.

T-A3:A19 – Execute reasoning

Executing a What-if analysis using reasoning function in Equation 7.3, and the input values of action T-A3:A18 computes the numbers depicted in Figure 7.3. These results can be used to decide, how many Raspberry Pi nodes should be contained in the cluster.

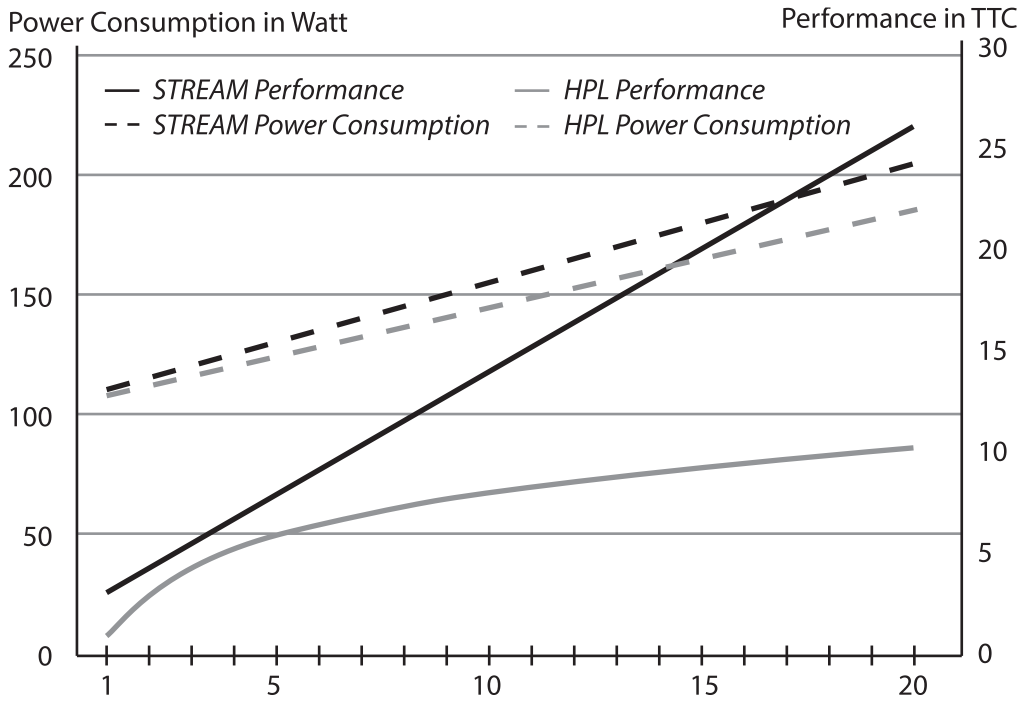


Figure 7.3: What-if analysis results in the prototypical reasoning project.

T-A3:A20 – Trigger activity

Based on the reasoning results computed in action T-A3:A19, the physical assembly of the RPi cluster is triggered.

7.3 Field study

→ Fig. 7.1 (p. 200)

The section constitutes the second validation part (cf. Section 7.1), being an *observational* validation method that considers the process model's (theoretical) use in two real-world projects in the research *Environment*. It employs the evaluation tool (cf. Section 3.5) of the RS to emphasize that the process model satisfies "the requirements and constraints of the problem [it was] meant to solve" [187, p. 85]. In particular, Section 7.3.1 compares the process model's capabilities and characteristics to the functional requirements, Section 7.3.2 to the non-functional requirements identified in and extracted from the real-world scenarios *SuperMUC* and *DRIHM* (cf. Section 3.2).

7.3.1 Validation against functional requirements

Table 7.1 itemizes the results of validating the process model against the functional requirements of the RS (cf. Section 3.3.2).

✓	UC-1	**Initiate reasoning activity**
Creating a reasoning suite in *Phase A* of activity template T-A3 implicitly provides the "cogent reason" required by UC-1, because 1) its specification references the reasons, and 2) without a cogent reason, it wouldn't be possible to collect all information necessary for a complete reasoning suite.		
✓	UC-1.1	**Negotiate SLA and attributes**
Especially action T-A3:A3 and T-A3:A4 fulfill this requirement, as their creation is closely related to the negation of concrete attribute values that could be stated in SLAs.		
✓	UC-2	**Define reasoning objectives**
The entire *Phase A* of activity template T-A3 extracts and defines not only reasoning objectives, but also concrete reasoning parameters. Besides, its eight actions implicitly decompose the tasks in a way that ensures result compatibility, since every single action in activity template T-A3 is aligned to and guided by the reasoning suite specifications.		
✓	UC-2.1	**Define attribute(s)**
Sentence template T-S4 in action T-A3:A3 formalizes attribute definitions and produces a list of attributes reasoning should consider, as requested by UC-2.1. Besides, the mandatory fields of an attribute concept (cf. Section 5.6), formalized in the provenance information model package `attributes` (cf. Section 6.1.3), ensure that each list entry contains (at least) a name, a scale, and a description. Finally, the central position of the attribute definitions in the reasoning suite fosters a common understanding about the attributes between all involved actors, as required by UC-2.1.		

continued on next page

continued from previous page

✓	UC-2.2	**Select workload**
Decision template T-D9 in action T-A3:A6 formalizes workload selection for a particular reasoning activity, resulting in a list of workload reasoning should consider and use, as requested by UC-2.2. Classifiers in the provenance information model package `workload` (cf. Section 6.1.6) ensure that each list entry provides a description and selection justification. Besides, T-D9 facilitates the selection of workload *relevant* to represent the analyzed circumstances, as called by UC-2.2.		
✓	UC-2.3	**Select IT infrastructure component(s)**
Actions T-A3:A3 and T-A3:A4 collectively assemble a list of IT infrastructure component types according to the targeted reasoning parameters. Consequently, the list contains only elements that are relevant to the specific reasoning project and ensure a cost-saving and effective IT infrastructure modeling in action T-A3:A5, as required by UC-2.3.		
✓	UC-3	**Model IT infrastructure**
Activity template T-A4 in action T-A3:A5 formalizes IT infrastructure modeling, resulting in an UML object diagram based on the classifiers of the provenance information model package `itinfrastructure` (cf. Section 5.4.5). The achieved IT infrastructure model is bound to the reasoning suite in general and the component type list in particular, compliant to UC-3 that requires a useful, focused, and preferably small model.		
✓	UC-3.1	**Model part of IT infrastructure**
The flexible notion of IT infrastructure components (cf. Section 5.4), the applied black box approach (cf. Section 5.4.3), and the provenance information model package structure (cf. Section 5.3) enable partial modeling of IT infrastructures, especially of large-scaled and complex ones. For instance, a group responsible for compute nodes can model storage elements as black boxes to emphasize a dependency, and request a black box refinement from the storage related group.		

continued on next page

continued from previous page

✓	UC-3.2	**Import IT infrastructure information**
The process model's underlying IT infrastructure notion (cf. Section 5.4) explicitly provides a semantic interface to third party models. As Section 5.4.2 details, both, the unique component identifier and the component type list, facilitate the mapping between IT infrastructure components and entities in third party models to foster actuality, consistency, and cost-efficient work, as required by UC-3.2. The feasible interaction with CIM databases that uses values of the `ManagedElement.InstanceID`, implicitly enables information import from other information models, because CIM supports "mappings to and from other network and system management information models" [390, p. 678].		
✓	UC-3.3	**Update IT infrastructure model**
The import functionality (cf. validation of UC-3.2) and the capability of partial IT infrastructure modeling (cf. validation of UC-3.1) facilitate model updating and collectively fulfill UC-3.3: partial modeling enables specific stakeholders to update their model parts, and the import functionality eases this updating process. In addition, using the `ComponentIdentifier` and `Type` classifiers in the provenance information model package `itinfrastructure` (cf. Section 5.4.5) for mapping modeled components to entities in third party models might render updating the IT infrastructure model obsolete in some situations.		
✓	UC-4	**Select model for attribute and component(s)**
Decision template T-D11 in action T-A3:A12 formalizes and guides the selection of an existing model as integration candidate to the reasoning function compliant to reasoning suite specifications. The decision template also provides a dedicated selection support tool to address the extend of existing models as requested by UC-4.		
✓	UC-4.1	**Create model proxy**
Activity template T-A5 in action T-A3:A13 formalizes the creation of a proxy that substitutes a missing model integration candidate. Guided by the (co) domain of the substituted model, activity template T-A5 describes the selection of a suitable model proxy creation method as well as the method's execution and result validation. The activity results in an alternative way to provide a value for the attribute/component combination, and to enable reasoning as requested by UC-4.1.		

continued on next page

continued from previous page

✓	UC-4.2	**Create load profile**
Action T-A3:A18 prepares input values for the assembled individual reasoning function. It particularly covers the load profile creation using workload selected in action T-A3:A6 and consequently, fulfills UC-4.2.		
✓	UC-5	**Execute reasoning**
The entire *Phase C* of activity T-A3 and particularly action T-A3:A19 cover and conduct the reasoning execution according to the reasoning suite specifications and employing the prepared reasoning function, as required by UC-5. The fulfillment is further detailed for UC-5.1 and UC-5.2.		
✓	UC-5.1	**Execute What-if analysis based reasoning**
Decision template T-D14 in action T-A3:A17 supports the selection of What-if analysis, which is used by action T-A3:A19 to execute reasoning.		
✓	UC-5.2	**Execute optimization based reasoning**
Decision template T-D14 in action T-A3:A17 supports the selection of optimization, which is used by action T-A3:A19 to execute reasoning.		
✓	UC-5.3	**Execute descriptive statistics based reasoning**
Decision template T-D14 in action T-A3:A17 supports the selection of descriptive statistics, which is used by action T-A3:A19 to execute reasoning.		
✓	UC-6	**Trigger activity**
Action T-A3:A20 handles the potential need to react to a particular reasoning outcome by allowing to trigger a suitable activity.		

Table 7.1: Results of evaluating the process model against functional requirements specified in the RS.

7.3.2 Validation against non-functional requirements

Table 7.2 itemizes the results of validating the process model against the non-functional requirements of the RS (cf. Section 3.4).

✓	NFR-1	**Individual component type sets**
The support of individual component type sets is threefold: **IT infrastructure notion** Considers an IT infrastructure component as graph node, an arbitrary type can be assigned to (cf. Section 5.4.1). **Modeling** In its `itinfrastructure` package, the provenance information model provides a dedicated classifier `ComponentType` that is individually assigned to a particular component, and that also supports the description of a component type hierarchy. **Reasoning function compilation** Defining the reasoning function's domain in action T-A3:A3 and T-A3:A4 is highly flexible and not bound to a fixed set of component types. Additionally, model integration selection in action T-A3:A12 is only required to comply to the reasoning suite, but there are no constraints about IT infrastructure component types.		
✓	NFR-2	**Individual attribute sets**
The support of individual attribute sets is twofold: **Definition** Sentence template T-S4 in action T-A3:A3 exclusively formalizes (meta) information that must be fixed for a considered attribute. In contrast, it does not dictate, which attribute should be considered for a particular situation. In addition, the sentence template applies the process model's underlying attribute decomposition in a concept and multiple instances (cf. Section 5.6), and consequently, supports individual attribute instances compliant to NFR-2. **Selection and binding** Actions T-A3:A12 and T-A3:A13 can select and compile a different model or model proxy, for any attribute instance of interest. The result, in turn, can be flexibly bound to any suitable component in the IT infrastructure model (cf. Section 5.6.2).		

continued on next page

continued from previous page

✓	NFR-3	**Multiple granularity levels**
The support of multiple granularity levels is threefold: **Graph-based notion** Represents IT infrastructure components as graph nodes (Section 5.4.1) that can be selectively replaced by a detailing sub graph. This enables the consideration of some components (nodes) on a very coarse-grained level, while other parts of the graph are (extremely) fine-grained. **Black box approach** Supports the alluded graph-based notion by providing the dedicated and tangible terms *black box* and *white box* describing a (non) refinement node (cf. Section 5.4.3), and by formulating a black box refinement algorithm that also describes the propagation of a refinement within the entire graph based on component types. **Provenance information model** In its `itinfrastructure` package, the `Component` classifier applies the composite pattern (cf. Section 5.4.5) to apply punctual and partial refinements of the IT infrastructure model. **Attribute binding** Allows binding an attribute concept and/or its instances(s) on every granularity level (Section 5.6.2), ranging from the entire IT infrastructure down to a single core.		
✓	NFR-4	**Workload consideration**
The compiled reasoning function consumes individual parameters, especially (work)load values (cf. Section 4.1), as illustrated in the controlled experiment in Section 7.2.1. The workload to execute and consider is selected by action T-A3:A6 and used in action T-A3:A13 and T-A3:A18 for model proxy creation and input value gain, respectively. Besides, checklist template T-C3 supports the identification of alternative workload in case the original one cannot be executed. Consequently, workload consideration is deeply enmeshed in the presented process model.		
✓	NFR-5	**Job cancellation**
The ability to formulate the considered workload as reasoning input parameter (cf. NFR-4) implicitly supports job cancellation, as the parameter value can be set to `null`, whenever necessary.		

continued on next page

continued from previous page

✓	NFR-6	**Development over time**
As detailed in Section 6.3, the input values of the reasoning function can be defined on a time step basis, which means that for each time step, a different input vector can be defined. This time step, in turn, can be set individually. Consequently, the reasoning function co domain implicitly supports investigating the development over time, because the reasoning function is "executed" at each time step.		
✓	NFR-7	**Simplicity**
The process model attains simplicity in two regards: **Modeling** The provenance information model classifiers represent only mandatory core elements and provide dedicated extension capabilities by 1) applying an object oriented approach that supports inheritance and sub classing, and by 2) specifying dedicated interfaces to third party models that can be used for importing or only referencing related information. The achieved objects can easily be reused due to the provenance information model's package structure. For instance, a `Workload` object can be assigned to a reasoning suite definition and simultaneously to a load generation in the context of a measuring. Together with the support of multiple granularity levels (cf. NFR-3), this reduces the amount of stored information to a minimum. **Reasoning activity** The specification of a reasoning suite in *Phase A* of activity template T-A3, especially before reasoning function compilation in *Phase B* and reasoning execution in *Phase C*, assures a simple, focused, and cost-effective mode of operation, since every single action and selection is guided by and aligned to the reasoning suite. In other words, every unessential effort is omitted such that it leads to a (local) optimal trade-off between accuracy and effort. For instance, the specification of reasoning interests, of a minimum model accuracy level, and of constraints in actions T-A3:A2, T-A3:A3, and T-A3:A8, respectively, at the beginning of a reasoning project avoids cost and time profuseness: model integration candidates are discard due to constraint violation even before a costly analysis, the reasoning function integrates only models compliant to the reasoning interest, and model proxy creation omits the expensive achievement of a unnecessary high accuracy level. Finally, the reasoning objectives driven refinement (cf. Section 4.3) assures a focused and highly fitting reasoning function.		

continued on next page

continued from previous page

✓	NFR-8	**Efficient use**
Two approaches enable the efficient use of the presented process model: **Complete formalization** Five template types (cf. Section 5.2.2) formalize every action to take, decision to make, and information to store during a reasoning project in a standardized way and thus, heavily ease the execution of a reasoning project. **Flexibility** The process model's capability to cover individual component type and attribute sets (cf. NFR-1 and NFR-2), and to respect development over time (cf. NFR-6), addresses the lion's share of situations, and renders adaptions for the bulk of reasoning projects unnecessary.		

Table 7.2: Results of evaluating research results against non-functional requirements in the requirements specification.

7.4 Related research analysis

→ *Fig. 7.1 (p. 200)*

The section constitutes the third validation part (cf. Section 7.1), being a *descriptive* method that comparatively discusses the process model and related work "use[ing] information from [related research] to build a convincing argument for the [process model's] utility" [187, p. 86]. According to this objective, related research analysis does purposely not aim at contemplating single models or approaches in detail, but at demonstrating the process model's advance and particularly its expansion of existing knowledge.

Section 2.4 motivates and explains the high relevance of IT infrastructure attributes, which causes a mass of related research. Section 7.4.1 explains the structuring approach to address this situation and to enable the alluded comparative discussion. Following this structuring, Sections 7.4.2, 7.4.3, and 7.4.4 present validation results, and Section 7.4.5 describes candidates for (partial) contribution to the process model.

7.4.1 Structuring of related research

The high relevance of quantitative IT infrastructure attributes (cf. Section 2.4), and the long history of their investigation produce a mass of related research. Since the above motivated comparative discussion can reasonably cover only a limited set, a taxonomy is applied to structure related research. A taxonomy is chosen as it constitutes a long-term structuring solution: taxonomy classes tend to change seldom, but class representatives might change and appear frequently. Besides, comparative discussion can investigate class characteristics and cover (all) class members simultaneously.

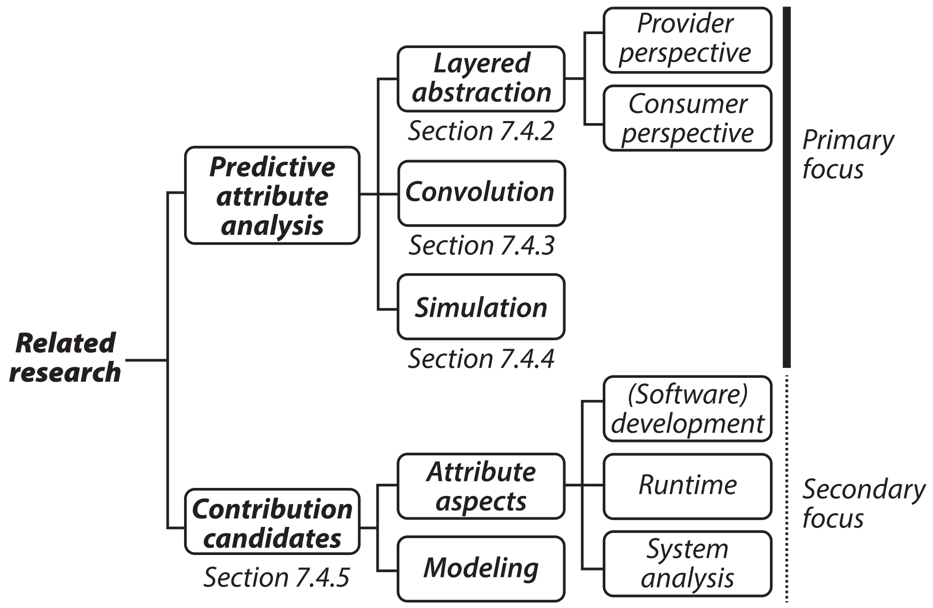


Figure 7.4: Taxonomy classifying related research for a comparative discussion with the presented process model.

Figure 7.4 depicts the concretely applied taxonomy. It classifies related research according to the pursued objectives and applied focus, and splits related research in the following classes:

- **Predictive attribute analysis** Class members deal with prediction of and reasoning about quantitative IT infrastructure attributes in a wider sense and without prerequisiting (full) system hardware being available. The class' role of the process model's "competitor" for comparative discussion puts it in the primary focus of the related research analysis, as indicated in Figure 7.4. The class members underlying methodologies further split the class in three sub classes:

  Layered abstraction Class members assemble their predictive models in a four-layered abstraction process. As Section 7.4.2 details, the process applies on real-world objects analytic and empirical tools to gain information, which in turn provide the basis for deducing the final predictive model, mostly a mathematical formula. Class discussion is split in a provider and a consumer perspective (cf. Section 2.1.1) to respect that the "approach of hardware and software separation [is used] in many modeling activities" [228, p. 215], like the PACE system [297].

Convolution Class members assemble their predictive model using convolution, a process that combines an *application signature* and *machine profile*, as detailed in Section 7.4.3. The class is not further refined, as most convolution based research deals with quantitative IT infrastructure attributes from a consumer perspective.

Simulation Class members employ simulation to investigate quantitative IT infrastructure attributes, as discussed in Section 7.4.4.

For each of the three itemized classes, discussion characterizes aspects important to the related research analysis, and extracts the class' underlying general methodology. This methodology is validated against non-functional requirements of the RS (cf. Section 3.4). In contrast, validation omits the functional requirements, since a non-fulfillment of the more generic non-functional requirements renders a validation of the more focused functional requirements unnecessary. Class discussion closes with a consideration of a selected set of class representatives to underpin validation results.

- **Contribution candidates** Class members potentially contribute to the presented process model by providing insights, tools, and information schemes. The thesis covers class members in manifold places that puts the class in the secondary focus of the related research analysis. Still, it is considered in Section 7.4.5 to emphasize the differing goals compared to the presented process model, and to highlight the at most contributing and particularly not competing character of the class members, although they are concerned with quantitative IT infrastructure attributes. The class is further split in two sub classes:

  Attribute aspects Related research that does not predict or reason about quantitative IT infrastructure attribute values, but deals with them in any other and especially purely descriptive way. Besides, class members mandatorily prerequisite the full system hardware being available. The application life cycle sub-divides the class in [297, 228]
  - *Development*, which covers the development phase and considers model developer support,
  - *Runtime*, which examines scheduling algorithms and (real-time) execution optimization, and
  - *System analysis*, which investigates productive and running systems like HPC clusters, supercomputers, or Grids.

Modeling Contains approaches for modeling elements related to the process model, like the IT infrastructure.

Section 2.3.1 justifies the focus on scientific applications due to their challenging demands on multiple especially quantitative IT infrastructure attributes [373]. Besides, Section 2.3.1 states that the contained concepts, demands, and techniques (mostly) apply equally well to other application classes. Correspondingly, related research analysis focuses on work dealing with IT infrastructure attributes from a provider and consumer perspective in the context of scientific applications. In addition, validation results are assumed to be applicable to work in other fields according to the alluded argumentation. For instance, Chen et al. [92] investigate performance modeling of component-based applications built of CORBA, *Java Enterprise Edition* (J2EE), and .NET in industrial software engineering projects, but the validation results discussed in Section 7.4.2 can be applied, as well.

7.4.2 Layered abstraction

→ *Fig. 7.4 (p. 220)*

Related research in this class derives in several ways predictive models for approximating and calculating one or at most two quantitative IT infrastructure attributes. Figure 7.5 summarizes the general model derivation process that underlies the bulk of class members. It abstracts in five layers the final predictive model from the considered object(s) (*EG-7.1*):

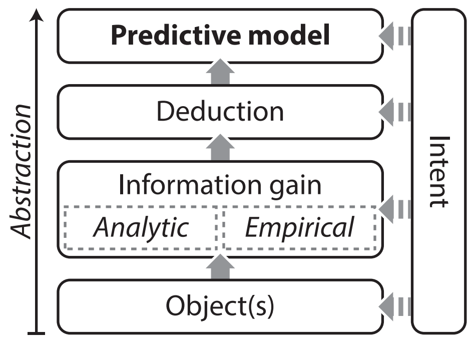


Figure 7.5: The common process of predictive model assembly.

- **Intent** Represents the derivation process' general goal, describing the targeted IT infrastructure attribute and the object(s) of interest. In this role, the intent influences all steps of the derivation process, as it guides object(s) selection, information gathering, and deduction method application.

- **Object(s)** Contains the examined hardware or workload a predictive model is derived for. Borrowing the understanding of measuring (↗KB p. 256), each object owns a (theoretically) infinite set of characteristics, describing an aspect of the object's nature.

- **Information gain** Selects a (small) sub set of the alluded infinite characteristic set that is of relevance for the targeted predictive model, also called *predictors* [319]. It is achieved by analytically and/or empirically analyzing the system, like source code inspection, algorithm analysis, communication pattern recognition, or measurement.

- **Deduction** Based on the gained information, an abstraction and formalization produces the predictive model, e.g., using regression.

- **Predictive model** The (abstract) predictive model describes correlations within and information about the examined object(s), mostly in a mathematical and parametric way that approximates and calculates quantitative IT infrastructure attribute values.

The *intent* of Kerbyson et al. [226] is considering performance and scalability of large-scale applications. This directly impacts *object* selection, resulting in the focusing on SAGE, a large-scale parallel code written in FORTRAN90 that uses MPI for inter-processor communications. *Information gain* for this object is twofold: an *analytic* part executes "an analysis of the code [and an] inspection of key data structures" [226, p. 1], an *empirical* part conducts an "analysis of traces gathered at run-time" [226, p. 1]. *Deduction* processes this information into a set of abstracting mathematical formulas that collectively assemble the final predictive model, i.e., "a performance model that encapsulates the code's crucial performance and scaling characteristics" [226, p. 1]. EG-7.1
The *intent* of Pfeiffer et al. [319] is the prediction of application performance on parallel computers (*objects*), and they summarize the model assembly as follows: "model generation begins with measured values [...] and application run times [(*information gain*)]. Then an automated series of least squares fits is made using backward elimination to ensure statistical significance [(*deduction*)]" [319, p. 11]. The final predictive model is an equation that calculates the run time of several application aspects, like non-overlapping computation and communication time.

The variety within each derivation process layer (cf. Figure 7.5) produces a plurality of predictive models. They (heavily) differ in the pursued intent and hence, not only in the considered objectives, applied information gain and deduction methods, but also in the capabilities of the finally assembled predictive model. Despite the class' extent, class members are not an alternative for the presented process model, since the underlying methodology does not validate successfully against the non-functional requirements of the RS, as Table 7.3 explains.

×	NFR-1	**Individual component type sets**
The high complexity, extend, and correlations exposed by the *object(s)* produce a big set of aspects to cover when deriving an accurate and comprehensive predictive model. Bailey et al. state that to address "the difficulty in producing a truly comprehensive model, present-day performance modeling researchers generally limit the scope of their models to a single system and application, allowing only the system size and job size to vary" [30, p. 186]. Hence, related research in the considered class addresses the alluded extent by (strongly) concentrating on a small set of details. Also Denzel et al. arrive at this conclusion, stating that "considering established methods and tools [...], problems either concentrate on partial aspects only or cannot scale sufficiently" [118, p. 331]. This concentration leads to mature and precise predictions, but at the same time, a particular predictive model can be applied only to the specifically considered IT infrastructure components as defined in the derivation's intent, and it is notably not possible to use the predictive model for individual component type sets, as requested by NFR-1.		
×	NFR-2	**Support individual attribute sets**
The variety of factors influencing quantitative IT infrastructure attributes (cf. Section 2.4) in the derivation process' lowest layer confirms the focusing of related research as described for NFR-1 above. This constraints consideration to only one or at most two quantitative IT infrastructure attributes at the same time, what causes a non-fulfillment of NFR-2. In addition, the required selection of a deduction method in the development process' third layer either excludes analysis of inter- and intra-attribute reciprocity and potential conflicts, or it hard wires reciprocity analysis in the model, which also contradicts NFR-2.		

continued on next page

continued from previous page

×	NFR-6	**Development over time**
Most class members fulfill NFR-4 as they address the central role of workload execution and the caused load in an either implicit or even explicit way. In contrast, consideration of development over time is omitted, due to the above alluded research and modeling focus. The non fulfillment is closely related to the support of job cancellation in NFR-5: if the development over time is not supported, job cancellation is automatically not supported as well, and hence, not further detailed.		

Table 7.3: Validation of related research in the layered abstraction class.

Provider perspective

→ *Fig. 7.4 (p. 220)*

The subsequent list describes class representatives that apply a provider perspective, as they mainly focus on hardware aspects when predicting and modeling quantitative IT infrastructure attributes. Although discussed work does not fulfill non-functional requirements, they are all potential model integration candidates (cf. Section 6.2.3).

- **Chip level power consumption** Basmadjian et al. [37] propose a model that estimates the dynamic power consumption (NFR-2 ×) of multi-core processors (NFR-1 ×). Their model is split in three areas, the processor's chip, components within a die, and components within a core, to address differing behaviors in terms of resource sharing techniques, such as on-chip caches, and energy-saving mechanisms. Each area can be refined by a dedicated formula. This results in an elaborated model that exposes high achievements compared to other approaches, but it does not fulfill NFR-1 and NFR-2.

- **HPC network performance** Martinasso et al. [276] present a model for predicting the "elapsed time" or TTC (NFR-2 ×) of communications that take place concurrently on high performance networks (NFR-1 ×). They introduce the notion of dynamic contention graphs to handle concurrent accesses, and to address dynamics of contention behavior, an achievement compared to existing predictive models in this field.

- **Grid performance** Cao et al. [79] consider the TTSD (NFR-2 ×) of an agent-based service discovery facility for Grids (NFR-1 ×). Their model covers the resource management infrastructure using a hierarchy of homogeneous agents, each being both, a service provider and requester.

The predictive performance model particularly focuses on the agent hierarchy, the services, the requests, and optimization strategies.

Consumer perspective

→ *Fig. 7.4 (p. 220)*

The subsequent list describes class representatives that apply a consumer perspective, as they mainly focus on workload aspects when predicting and modeling quantitative IT infrastructure attributes. Although discussed work does not fulfill non-functional requirements, they are all potential model integration candidates (cf. Section 6.2.3).

- **TTC of message-passing applications** Abandah et al. [13] develop a performance model for estimating the TTC (NFR-2 ×) of message-passing applications based on node communication in a cluster (NFR-1 ×). They analyze the application's high-level source code and measure a set of basic communication patterns to assemble their predictive model. Although the development over time could be incorporated in the model by setting the message length to zero bytes, it is not contained in the presented version (NFR-6 ×).
- **TTC of ASCI workload applications** The research team of Hoisie[3] et al. is concerned with performance models of the *Department of Energy* (DOE) *Accelerated Strategic Computing Initiative* (ASCI) workload [228] on Petascale and Exascale systems [31] (NFR-1 ×). In particular, they focus on analytic performance (TTC) (NFR-2 ×) and scalability models of Sweep3D [193], SAGE [226, 112], and VPIC (cf. Appendix B), as the codes are "representative of the workload of the ASC program at Los Alamos and elsewhere" [192, p. 7]. Resulting models take into account "the main computation and communication characteristics of the entire code [... and ...] encapsulate the code's crucial performance and scaling characteristics" [226, p. 11]. Although they provide a kind of model assembly methodology in [31], and although the models are highly elaborated and expose a very good prediction accuracy, as proven by several machine analysis discussed in Section 7.4.5, they are limited to a single attribute and a small set of applications. In addition, the predictive model does not provide a time parameter to support the development over time (NFR-6 ×). Instead, they predict the TTC as "a function of primary computation and communication parameters" [193, p. 3].

[3]Dr. Hoisie's profile page at Pacific Northwest National Laboratory: `http://www.pnl.gov/computing/staff/staff_info.asp?staff_num=7501` (Visited 2014-07-05).

- **Power consumption of applications** Brochard et al. [71] present a model for predicting the power consumption (NFR-2 ×) of HPC applications and particularly a given benchmark. They empirically assemble the model by executing SPEC2006 benchmarks, measuring the power consumption using *Active Energy Manager* measuring tools, and deriving a model based on the characteristics measured at frequency f. The resulting predictive model consumes the microprocessor's frequency (NFR-1 ×) and the performance, described in CPI (cf. Section 2.4.2), as input parameters. The model does not provide any parameters for the development over time (NFR-6 ×).

7.4.3 Convolution

→ Fig. 7.4 (p. 220)

Related research in this class bases upon convolution, a "computational mapping of an application signature onto a machine profile" [82, p. 926]. As Section 2.3.3 further details, an application signature is a "summary of the operations to be carried out by an application [...] independent of any particular machine" [83, p. 337], a machine profile is a "characterization of the rates at which a machine can (or is projected to) carry out fundamental operations abstract from the particular application" [84, Sec. 2].

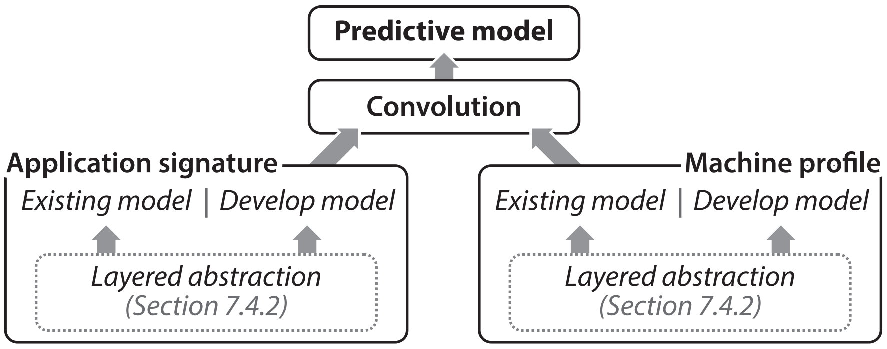


Figure 7.6: The general convolution process to assemble a predictive model from an application signature and a machine profile.

A fictive convolution approach uses an existing application signature, and creates an individual machine profile for HPC clusters. This creation employs and relies on the layered abstraction process, which inherently induces its validation results (cf. Table 7.3) to the machine profile. This, in turn, influences the validation results of the convolution approach that is constraint to HPC clusters.

EG-7.2

Figure 7.6 pulls elements together and emphasizes the extending character of the convolution class compared to the layered abstraction class discussed in Section 7.4.2. As Figure 7.6 depicts at its bottom, the application signature and/or the machine profile might already exist or is individually created. In either case, the layered abstraction approach is applied to achieve the corresponding model. This causes a validation result "inheritance" (*EG-7.2*). Table 7.4 explains the extension of inherited validation results.

×	NFR-1	**Individual component type sets**
Despite the generic and methodical nature of convolution, most approaches are constraint to a particular IT infrastructure, as "performance modeling techniques primarily rely on combining the performance profile of an application on a **well-known** HPC architecture [30, 372]" [45, p. 1]. In other words, at least one input parameter of the computational mapping is (mostly) limited to a specific IT infrastructure type or even a specific system, which in turn disqualifies convolution based approaches for supporting individual component type sets. Besides, the limited machine profiles can rarely be exchanged by potentially more flexible ones, because "the performance enhancing features of novel processing devices can be significantly different from a conventional microprocessor-based system, [... resulting in a..] limited applicability" [45, p. 1].		
×	NFR-2	**Individual attribute sets**
The lack of supporting individual attribute sets is caused by the inheritance of the layered abstraction's validation results: Most models, convolution consumes as input, were created using the layered abstraction process. In turn, the input model is already limited to a particular or very small set of quantitative IT infrastructure attributes, which results in a limitation of the convolution based predictive model, as well.		

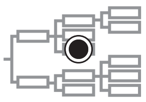
Table 7.4: Validation of related research in the convolution class.

The subsequent list discusses class representatives, and particularly highlights validation result inheritance alluded above:

- **Performance of large-scale scientific computation** Bailey et al. [30] present a convolution based performance prediction approach. Both, the application signature and the machine profile, are created using the layered abstraction development process (cf. Section 7.4.2). This causes validation result inheritance: the machine profile, for instance, is created by executing low level benchmarks and measuring the system's behavior (*information gain*) while focusing on memory access and communication patterns (*object(s)*, NFR-1 ×).

- **HPC cluster** The research team of Carrington[4] et al. is concerned with "gaining insight into the factors that affect performance" [84, p. 1] by employing convolution methods. In particular, Carrington et al. work on predicting the performance (NFR-2 ×) of scientific applications on HPC platforms (NFR-1 ×). Both, the consumed application signatures and machine profiles, are created using the layered abstraction: an application's sequential part is reflected by memory traces (*objects*) collected via the MetaSim Tracer [85] (*information gain*), an application's parallel part is reflected by MPI communication traces (*objects*) collected by MPIDtrace [85] (*information gain*) [82] (NFR-1 ×). Using the MetaSim Convolver, both are mapped to the corresponding information in a machine profile to get the performance model of a full parallel application (*deduction*) [85].

7.4.4 Simulation



→ *Fig. 7.4 (p. 220)*

Simulation is "an imitation (on a computer) of a system as it progresses through time" [334, p. 2] that aims at "predict[ing] the behavior of a system under particular circumstances when it is impossible, or at least undesirable, to experiment with the system itself" [72, p. 1]. This behavior includes, amongst others, communication processes in parallel applications [439] or interactions between various hardware and software components [209]. Figure 7.7 depicts the commonly applied simulation approach (*EG-7.3*), and the subsequent itemization explains contained elements.

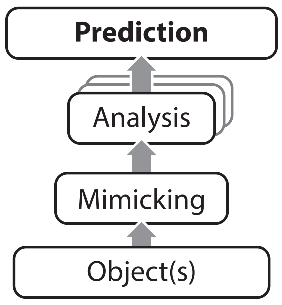


Figure 7.7: Commonly applied simulation approach for predictive modeling.

[4]Dr. Carrington's profile page at San Diego Supercomputer Center at University of California: `http://users.sdsc.edu/~lcarring/` (Visited 2014-07-05).

- **Mimicking** Imitates in a preferably exact way the considered system or target machine in terms of exposed (real-time) behavior and characteristics. Mimicking is executed only once [228]. A widely applied approach is discrete even simulation, which represents “only the points in time at which the state of the system changes” [334, p. 15]. In other words, mimicking describes the system as a series of events, which are sometimes split in *scheduled* bound events and conditional events that depend on the *system’s state* [334].

- **Analysis** Consumes the mimicking results to compute predictions. It is executed as often as required, focusing on different aspects and details.

EG-7.3

BigSim is a “simulation framework that aims at providing fast and accurate performance evaluation [...] large parallel systems using much smaller machines” [439, p. 221]. Its **mimicking part** (cf. Figure 7.7) is a parallel emulator [440] that provides a virtualized execution environment using out-of-core execution. It produces a set of event logs capturing the application’s computation and communication behavior. These logs are consumed by the **analysis part** for post-mortem analysis using a trace-driven parallel simulator [441] that predicts parallel performance on multiple resolutions.

The variety within each simulation layer (cf. Figure 7.7) produces a plurality of simulation tools, ranging from full system simulators to very focused ones. Despite this extent, class members are not an alternative for the presented process model, since the underlying methodology validates not against the non-functional requirements of the RS, as Table 7.5 explains.

×	NFR-1	**Individual component type sets**
Usually, the imitated real world objects expose a high complexity and own a huge amount of details, e.g., “each application evidences idiosyncratic memory usage, communication pattern, and load balance issues” [439, p. 221]. Especially mimicking aims at exactly imitating real world objects and at copying their (physical) configuration as accurate as possible [4]. This situation strongly requires the simulation approach to apply a narrow focus, because “the very high degree of detail does not allow the scaling [...] of simulation to HPC systems of many thousands of interconnected processors, [because] this would be too time- and resource-consuming, if not even practically impossible” [118, p. 332]. The narrow focus, in turn, disqualifies supporting individual component type sets, and “recorded measurements might not be useful when attempting to predict the performance of an entirely different type of system” [439, p. 222].		
(×)	NFR-2	**Individual attribute sets**
A couple of simulation-based approaches focus on mimicking and omit analysis. Although this additional concentration (mostly) results in an even higher accuracy level, it lacks any attribute prediction or reasoning capability at all. Instead, the approaches act as data providers.		
×	NFR-3	**Support multiple granularity levels**
The intent of accurately copying real world objects alluded in the validation of NFR-1 also results in the fixation of a single global granularity level, or in other words, multiple granularity levels are not supported. The fixed granularity level is mostly very high to enable consideration and analysis of cycle accurate processes in microprocessors or memory to obtain precise application execution times, IPC values, or cache miss rates [118]. Bailey et al. argue that “cycle-accurate simulation is the performance modeling baseline [and] given enough time [and] details about a machine, we can always explain and predict performance by stepping through the code instruction by instruction” [30, p. 190].		

Table 7.5: Validation of related research in the simulation class.

The subsequent list discusses class representatives and highlights aspects related to the validation above:

- **DRAMSim2** Is a DDR2/3 memory system simulator [340] (NFR-1 ×), which includes a detailed cycle-accurate model (NFR-3 ×) of a memory controller that issues commands to a set of DRAM devices attached to a standard memory bus. DRAMSim2 provides a lot of tools, like an object-oriented programming interface and a robust visualization tool, and it creates achievements in the simulation domain, since "many of these CPU simulators overlook the need for accurate models of the memory system" [340, p. 16]. Yet, the DRAMSim2 approach lacks the analysis part and hence, is not able to analyze quantitative IT infrastructure attributes at all (NFR-2 ×).

- **End2End HPC cluster** Denzel et al. [118] extend the OMNeT++ simulation framework [398] to achieve a full-system event-driven end-to-end simulation of versions of the *Productive, Easy-to-use, Reliable Computing System* (PERCS) HPC architecture (NFR-1 ×). They replace statistical packet generators of the OMNeT++ simulation framework with a "new abstract computing node model that is driven by real-world application traces" [118, p. 332] of MPI calls and computing events in the application software. The presented approach generates a set of values for performance instances (cf. Section 2.4.2), but does not cover other quantitative IT infrastructure attributes (NFR-2 ×).
- **Reliability** Jones et al. [216] use simulation to study the impact of sub-optimal checkpoint interval assignment (cf. Section 2.4.3) on application efficiency. They cover reliability and performance, but they do not support individual attribute sets (NFR-2 ×).

7.4.5 Contribution candidates

→ *Fig. 7.4 (p. 220)*

The class contains related research that does not "compete" with the presented process model as the previously analyzed classes. Instead, it contains candidates for potential (partial) contributions to the presented process model by providing insights, tools, and information schemes. Since the entire thesis extensively covers and discusses class members whenever they are related to a certain aspect, the class is in the secondary focus of the related research analysis and examined only selectively. The class is discussed according to the structuring explained in Section 7.4.1.

Attribute aspects

→ *Fig. 7.4 (p. 220)*

The class contains research that deals with quantitative IT infrastructure attributes in a *descriptive* and particularly not *predictive* way. The class is considered anyway to highlight the at most contributing and particularly not competing character of the class members, although they are concerned with quantitative IT infrastructure attributes. The application life cycle structures the subsequent class discussion [297, 228].

→ *Fig. 7.4 (p. 220)*

- **(Software) development** Contains approaches, tools, and models that support the development and design of a particular system or workload, e.g., to support application designers and programmers in optimizing their source code. Although class members tend to (strongly) influence quantitative IT infrastructure attributes, they expose neither predictive capabilities, nor contain models that could be extracted for integration. Instead, they can be used to achieve a specific attribute target value that has been generated by a reasoning project (cf. Section 8.2.6).

  Programming models Describe the way how to design and encode a particular (computational) problem for machine-based execution. Bigot et al. [48], for instance, present a programming model that aims at supporting performance portability by enabling code reuse, particularly of hardware agnostic parts.

  Libraries Provide a set of tools to solve recurring (small) tasks. Browne et al. [73] introduce the *Portable API* (PAPI), a standard programming interface for accessing hardware performance counters available on most modern microprocessors, to improve the collection of performance data. They aim at addressing poor documentation, instability, or unavailability of vendor specific performance counter access to the user level program. PAPI provides a consistent interface and methodology for performance counter use to support application designers and engineers. Although PAPI might (heavily) improve the performance of a particular workload, it does not provide any prediction or reasoning capability.

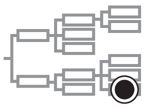
→ *Fig. 7.4 (p. 220)*

- **Runtime** Contains approaches, tools, and models related to quantitative IT infrastructure attributes during workload or system runtime. In contrast to the (software) development class, runtime class members often implicitly contain models that could be extracted for integration in an individual reasoning function (cf. action T-A3:A12).

  Frequency adaptions The concept of *Dynamic Voltage and Frequency Scaling* (DVFS) is one of the major power-saving techniques. At a glance, it adapts the "voltage and frequency of compute cores or other processor parts to the current system load" [354, p. 481]. There is a bulk of related research dealing DVFS, for instance:
  - Ge et al. present a "run-time scheduler [...] that supports system-wide, application-independent, fine-grained, DVFS-based power

management for generic power-aware clusters" [162, p. 18]. Although this scheduler (obviously) does not provide any reasoning capabilities, the included predictive models describing power consumption and workload could be used for integration in action T-A3:A12, or for input value generation in action T-A3:A18, respectively.

- Schöne et al. present a "configurable CPU frequency governor that adapts processor frequencies based on performance counter measurements instead of processor load" [354, p. 481]. Based on the measured number of executed instructions and the number of *last level cache misses* (llcm) in a certain time interval, they determine the rate of *instructions/llcm* that "presumably allows [them] to estimate how strongly the current CPU load is bound in its performance due to main memory accesses" [354, p. 482]. Their work is an example for a purely insights contributing related research intent, as they neither provide reasoning capabilities nor potential model integration candidates.



→ Fig. 7.4 (p. 220)

- **Machine analysis** Contains approaches, tools, and models that investigate a particular machine, either executing day-by-day workloads or in separate experiments. Compared to the previous two classes, research about machine analysis tends to provide the most contribution, because it generates a lot of insights, especially about factors that influence quantitative IT infrastructure attributes, and it often employs models that could be extracted for integration in action T-A3:A12.

  An established research team is the one around Adolfy Hoisie (cf. Section 7.4.2). They use their predictive models of Sweep3D, SAGE, and VPIC (cf. Section 7.4.2) to investigate a broad set of world-leading machines. The following alphabetically ordered list points out some work of the team, and of other researchers. Each entry starts with a general technical description of the analyzed system, followed by selected details. Appendix B overviews the mentioned benchmarks.

  AppleTV cluster Fürlinger et al. [157] built a cluster of four *second generation AppleTV* (ATV2) devices, each connected to an Ethernet switch and communicating via MPICH 2. An ATV2 runs an Apple A4 processor that combines an ARM Cortex-A8 running at 1 GHz with a PowerVR SGX535 GPU, and 256 MB RAM. To run parallel jobs, they installed an MPI distribution. They evaluated the cluster's power and performance characteristics, aiming at the provision of a "data point on the current state of energy efficient parallel and distributed computing on ARM powered consumer electronic devices" [157, p. 2]. Using the Coremark, HPL, and membench benchmarks, they analyzed the performance of CPU cores of embedded systems, of the interconnect, and the parallel TTC of the cluster, resulting in a peak performance of 160.4 MFLOP/s (cf. Section 2.4.2) in double precision arithmetic, and about 16 (MFLOP/s)/Watt.

ASCI Q Within the scope of the *Advanced Simulation and Computing* (ASCI) program, ASCI Q was installed at the *Los Alamos National Laboratory* (LANL) in 2003. It consisted of 2048 HP AlphaServer ES45 nodes, which are interconnected by two rails of the Quadrics QsNet fat-tree network [316]. As part of the alluded Hoisie research team, Kerbyson et al. [228] analyzed the performance of ASCI Q with the predictive performance model of SAGE, using a problem size of 13,500 cells per processor.

BlueGene/L The IBM BlueGene/L [160] is a massively parallel supercomputer built of 65.536 dual-processor nodes that communicate via five interconnect networks for I/O, debug, and various types of interprocessor communication. BlueGene/L achieved a peak performance of 360 TFLOP/s. Hoisie et al. [59, 192] discuss its performance using the Sweep3D and SAGE models, and compare (normalized) performance numbers with other systems [192], like the Cray Red Storm (XT3), and ASCI Purple (IBM Power5). Borill et al. [59, 60] used the MADbench2 benchmark to investigate specifically the (a)synchronous I/O performance of BlueGene/L's parallel file systems.

Columbia Built by *Silicon Graphics* (SGI) for the *National Aeronautics and Space Administration* (NASA) in 2004, Columbia was a SGI Altix HPC cluster, consisting of 10.240 processors. It achieved a peak performance of 63 TFLOP/s. Biswas et al. [50] characterized its parallel performance in terms of floating-point performance, memory bandwidth, and message passing communication speeds. In doing so, they executed a subset of the HPC Challenge benchmark and a variety of scientific applications to investigate different configuration options available on Columbia with regards to their impact on performance. Borill et al. [59, 60] used the MADbench2 benchmark to investigate specifically the (a)synchronous I/O performance of Columbia's parallel file system.

Earth Simulator Is a Japanese parallel vector HPC system. It was built as distributed memory system consisting of 640 processor nodes that were inter-connected by a single stage full crossbar network [351, 350]. It achieved a peak performance of 40 TFLOP/s, and was initially expected to achieve a sustained performance of 5 TFLOP/s (12.5% of system-peak) on atmospheric applications [427, 227]. Carter et al. [86] used the *Microwave Anisotropy Dataset Computational Analysis Package* (MADCAP) [58] to analyze the Earth Simulator's performance. As part of the alluded Hoisie research team, Kerbyson et al. [228] analyzed the performance of the Earth Simulator with the predictive performance model of SAGE, and compared the results with the performance numbers for the ASCI Q system.

Jaguar A Cray supercomputer consisting of 224.256 dual-core 2.6 GHz AMD Opteron processors, located at the *Oak Ridge National Laboratory* (ORNL). Jaguar achieved a peak performance of 1.75 PFLOP/s. Borill et al. [59, 60] use the MADbench2 benchmark to investigate specifically the (a)synchronous I/O performance of Jaguar's Lustre parallel file system.

Roadrunner An HPC system whose design represents a careful balance between components available at the market and technological advance. It consists of the PowerXCell 8i, a processor IBM implemented separately for Roadrunner, and having a peak performance of 108.8 GFLOP/s on double-precision operations. Barker et al. [32] employed the Sweep3D model to examine Roadrunner's memory and communication performance.

Modeling

→ *Fig. 7.4 (p. 220)*

The class covers modeling approaches and solutions related to the process model in every regard expect quantitative IT infrastructure attributes. In particular, it deals with IT infrastructure modeling, since extending one of the multiple existing models would benefit from third party experience, and ease integration in and interaction with existing (management) tools. Although there are manifold and widespread used (meta) models and languages to describe a network, a computer system, or a distributed system, their inherent overhead would outweigh the benefits of extending an existing model, as demonstrated for two commonly used IT infrastructure models [353]:

- **CIM** The *Common Information Model* (CIM) [123] of the *Distributed Management Task Force* (DMTF) provides a common definition of management information for systems, networks, applications, and services, and allows for vendor extensions [103] in an object-oriented and user-extensible way [121]. It aims at "enabl[ing] interoperability and information sharing between various [network and system management] systems" [390, p. 677], and is used in manifold areas. In

Grids (cf. Section 2.2.3), for instance, it provides the basis for a set of Grid monitoring systems [265, 329]. The alluded intent of enabling interoperability results in a very extensive information model. Two reasons disqualify CIM for use or as extension base for the process model's provenance information model:

- CIM contains several hardware related details, like the `MaxBlockSize` of a media device, which are completely dispensable for the presented process model. The caused overhead strongly contradicts the aim of generating significant reasoning statements in a good or even optimal time- and cost scale (cf. Section 4.3). Nevertheless, the provenance information model's object orientation supports adding arbitrary hardware details in case they are required (cf. Section 5.3).
- CIM classes mix up details and aspects that are explicitly separated by the provenance information model's package structure to foster task delegation (cf. Section 5.3.1).

As a result, the required adaptions of CIM to the presented process model's needs would heavily outweigh the benefits of using an existing information model.

- **GLUE** The *Grid Laboratory Uniform Environment* (GLUE) information model of the *Grid Schema Working Group* in *Open Grid Forum* (OGF) [166] provides capabilities for describing Grid entities, based on "the experience of several modeling approaches being used in [...] production Grid infrastructures" [22, p. 5]. Just like CIM is GLUE defined in an object-oriented way and described as UML class diagrams (↗KB p. 271), what enables individual extensions while ensuring compliance to the standardization progress of the OGF [39]. The GLUE information model is used in several Grid middlewares, like gLite or the Globus Toolkit, especially for information services (cf. Section 2.2.3). Two reasons disqualify GLUE for use or as extension base for the process model's provenance information model:

  - GLUE focuses on Grid specific aspects and defines classes like `Computing Element`, `Storage Element` or `Endpoint`, which are all suitable for the information model's targeted IT infrastructure component types, but cause overhead for other IT infrastructures, like a single system (cf. Section 2.2.1).
  - Just like CIM, GLUE classes mix up details and aspects that are explicitly separated by the provenance information model's package structure to foster task delegation (cf. Section 5.3.1).

As a result, the required adaptions of GLUE to the presented process model's needs would heavily outweigh the benefits of using an existing information model.

Chapter 8

Conclusion

The chapter summarizes the thesis and points out further (research) directions. Section 8.1 recapitulates the research question, and describes the presented process model in a compressed overview, referencing several sections to indicate detailed motivations, discussions, and explanations. Section 8.2 highlights potentials for improvements and possible applications of the process model, before Section 8.3 provides closing remarks.

8.1 Thesis summary

Quantitative IT infrastructure attributes describe an IT infrastructure during workload execution (cf. Section 5.6). They are used in a variety of ways and situations, e.g., for describing (legally binding) provisioning specifications in *Service Level Agreements* (SLA). Decisions regarding attribute alignment to (externally) given target values are required to build on (complete) facts, and not (purely) on educated guesses or partial information (cf. Section 1.1). This can be achieved by *reasoning*, “the process of forming conclusions, judgments, or inferences from facts or premises”. Despite the plethora of existing approaches to do reasoning for a variety of attributes and nearly all IT infrastructure components, they are all limited in terms of IT infrastructure coverage and attribute coverage (cf. Section 1.3.4, 7.4). Therefore, the thesis and presented results aim at answering the following research question:

How to facilitate reasoning about quantitative IT infrastructure attributes to support decision-making while respecting IT infrastructure complexity and scale as well as inter- and intra-attribute correlations?

The great variety of reasoning aspects and intents (cf. Section 6.1) as well as the objective of generating a sustainable solution renders a “one-size-fits-all” model insufficient (cf. Section 4.1). Thus, the thesis analyzes how to achieve a flexible and sustainable solution that simultaneously benefits from the great extend of existing approaches (cf. Section 1.3.4). The result is a process model that generically prescribes and formalizes *how* to compile an *individual* and casuistic reasoning model, suitable for a specific reasoning intent (cf. Section 4.1). The produced reasoning model is a mathematical mapping, called a *reasoning function*, of a parameter set on a vector of quantitative IT infrastructure attribute values. Equation 8.1 depicts the reasoning function f in its most generic form and highlights the semantic decomposition of the reasoning function's domain in *modification* and *configuration* parameters (cf. Section 4.1).

$$\underbrace{f(\underbrace{mod_1, ..., mod_n}_{\text{Modification parameters}}, \underbrace{conf_1, ..., conf_m}_{\text{Configuration parameters}})}_{\text{Domain}} = \underbrace{\underbrace{\begin{pmatrix} attr_1 \\ ... \\ attr_z \end{pmatrix}}_{\text{Attribute values}}}_{\text{Co domain}} \tag{8.1}$$

The process model bases upon three design concepts (cf. Section 4.2):

- **Integration of existing artifacts** The high relevance of IT infrastructure attributes (cf. Section 2.4) produces an abundance of specialized (and mature) models, instrumented components, and gained measurements, covering a variety of partial aspects of a reasoning intent. The process model bases upon integration to benefit from this situation and to use elaborated, established, and especially validated artifacts, each addressing specific issues and respecting the individual characteristics of a certain reasoning situation. In particular, the process model describes the selection of suitable models and their incorporation in the reasoning function.

- **Iterative function refinement** Although a variety of factors influence an attribute, reasoning might be interested in only a (small) subset. The process model addresses this varying demand by beginning the reasoning function compilation with a coarse-grained function skeleton and iteratively refining it until the function covers the domain and co domain on a sufficient (predefined) accuracy level and extent.

- **Flexible attribute binding** The huge set of influencing factors on and objectives of an attribute causes manifold attribute specializations (cf. Section 2.4). The process model addresses this variety by decomposing the notion of a quantitative IT infrastructure attribute in a generic concept and multiple concept instances (cf. Chapter 5.6).

Figure 8.1 overviews the process model, and arranges the term *reasoning project* as an execution instance. Besides, Figure 8.1 depicts the process model's building blocks and their grouping in two parts:

- **Artifacts and procedures** Defines and formalizes notions, models, and methods that are employed by a reasoning project. It consists of seven parts, as depicted at the bottom of Figure 8.1 (cf. Chapter 5).
- **Reasoning methodology** Formalizes the reasoning activity as a "comprehensive integral series of techniques and methods creating a general system theory" [199] (cf. [315]), considering a method as a "system of rules and guidelines for a consistent procedure" [315, p. 41]. It consists of three phases, as depicted at the top of Figure 8.1 (cf. Section 6).

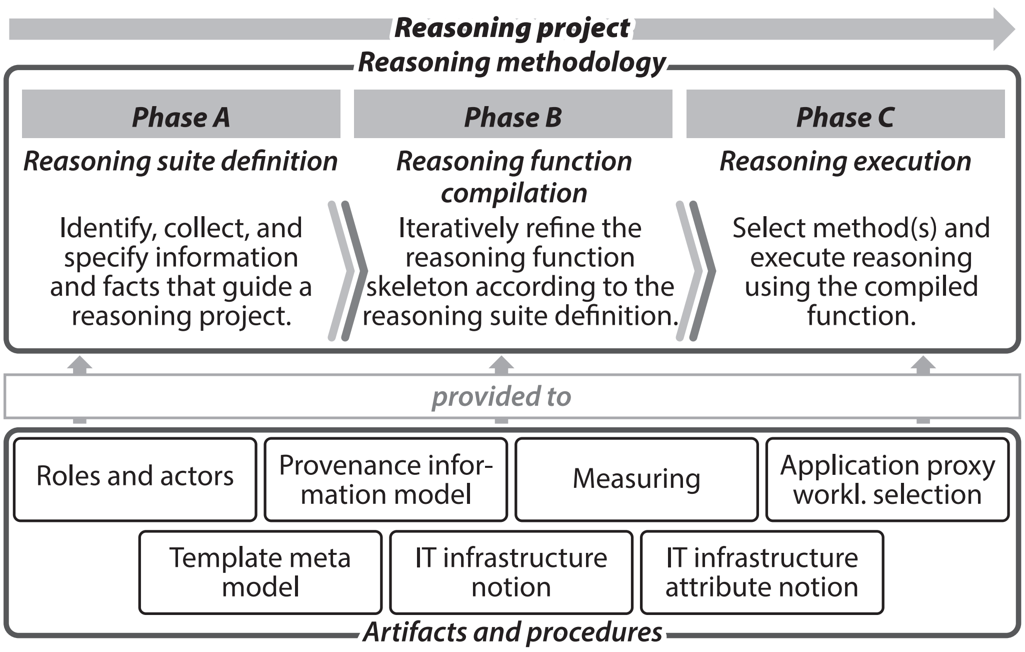

Reasoning project
Reasoning methodology
Phase A
Reasoning suite definition
Identify, collect, and specify information and facts that guide a reasoning project.
Phase B
Reasoning function compilation
Iteratively refine the reasoning function skeleton according to the reasoning suite definition.
Phase C
Reasoning execution
Select method(s) and execute reasoning using the compiled function.
provided to
Roles and actors
Provenance information model
Measuring
Application proxy workl. selection
Template meta model
IT infrastructure notion
IT infrastructure attribute notion
Artifacts and procedures

Figure 8.1: Overview of the presented process model for the integrated reasoning about quantitative IT infrastructure attributes.

Besides, research resulted in a set of publications (cf. Section 1.3.3).

8.2 Future Work

Future work covers potential research tasks and directions in terms of widening and using the presented process model. Figure 8.2 overviews the directions and groups them according to the time axis that underlies a reasoning project execution:

- **Preparation** Covers aspects *before* reasoning project execution, particularly the quantification of attributes in Section 8.2.1.
- **Extension** Covers enhancements of the presented process model. The group contains the coverage of human factors in Section 8.2.2, of Cloud computing in Section 8.2.3, and of IT infrastructure surroundings in Section 8.2.4, as well as automation support in Section 8.2.5.
- **Use** Covers aspects *after* reasoning project execution, particularly the generation of modification recommendations based on reasoning project results in Section 8.2.6.

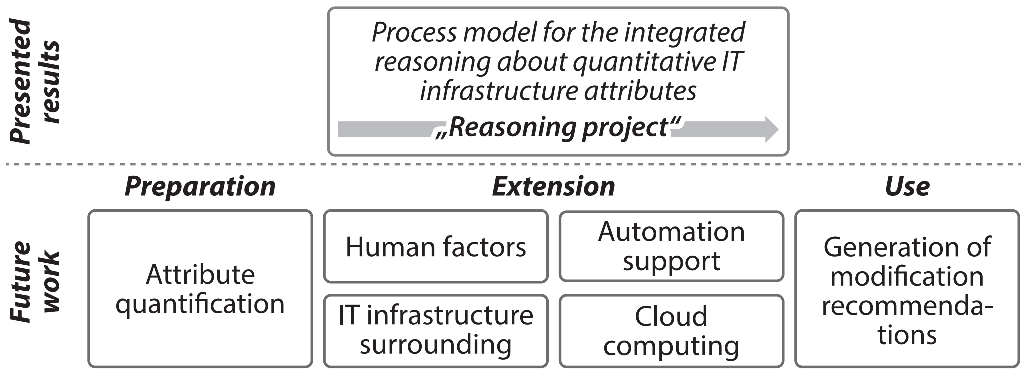

Presented results
Process model for the integrated reasoning about quantitative IT infrastructure attributes
„Reasoning project“
Preparation
Extension
Use
Future work
Attribute quantification
Human factors
Automation support
IT infrastructure surrounding
Cloud computing
Generation of modification recommenda-tions

Figure 8.2: Potential research tasks and directions regarding the presented process model.

The subsequent sections detail the alluded future work, respectively. Resulting research tasks are more generic for the *Preparation* and *Use* group, as they cover wider fields and research disciplines. In contrast, research tasks in the *Extension* group are provided more concretely.

8.2.1 Attribute quantification

The presented process model prerequisites *quantitative* attributes, e.g., performance in *FLOP/s* (cf. Section 2.4.2). Although quantitative descriptions exist for several attributes, especially for the highly relevant attributes performance, energy efficiency, availability, and reliability [170] (cf. Section 2.4), for other attributes only *qualitative* descriptions are available, as the two following prevalent examples illustrate:

- **Information Security (IS)** Aims at ensuring confidentiality, integrity, and availability of services and data [1, 352]. Instead of objectively assessing, measuring, and comparing the security situation of an IT infrastructure, this is mostly and forcedly based upon the (reliable) gut feeling of responsible security experts and a set of standardized IS controls, e.g., specified in ISO/IEC 27001 [204]. The wide distribution of hypothesis-based IS consideration [210, 246], derived from known risks or attacker models, and the "Directions in Security Metrics Research" published by NIST in 2009 [208], collectively further underpin the qualitative nature of IS consideration.
- **Interoperability** Describes the interaction between computing components [90, 243]. It is understood as the "ability to exchange information and to mutually use the information which has been exchanged" [305, p. 43], which has been identified being a key issue [171]. In the realm of e-Government [24], for instance, services need to be provided on a countrywide base involving manifold agencies and institutions while avoiding vendor lock-in. Interoperability is mainly addressed in a qualitative way by relying on technical specifications all involved agencies *should adopt* [171].

The lack of standardized measurement units cause the (inherent) difficulty of quantifying qualitative attributes. Covering qualitative attributes in the reasoning process requires their quantification, because otherwise, it would neither be possible to describe them in the reasoning functions (co) domain (cf. Section 4.1), nor to apply (numerical) optimization problem solving approaches (cf. Section 6.3).

8.2.2 Human factors

In comparison to science and especially to scientific applications (cf. Section 2.3), literature about IT infrastructures in industry does not provide concentrated statements about contained components or architectures. Instead, it discusses correlations, reciprocity, and dependencies of IT and business related aspects, e.g., linkage between IT and firm performance [323, 399, 378], alignment of IT and business strategy and processes [257], risk assessment for IT investments [404], and handling influencing factors of environmental uncertainty on IT governance [423]. Even if literature agrees on IT infrastructure's important role for every firm's operations [300, 341, 335] and on its business value creating nature [66], there are no common technical characteristics or definitions of IT infrastructure.

Nevertheless, there are recurring patterns, concepts, and notion that can be used to sharpen the understanding of IT infrastructure application in industry. Several of them can be described by the morphological field in Section 2.5, e.g., a long-term planning horizon [411, 197, 46], non-federated resource provisioning by only one internal department or an external outsourcing company. In contrast, there is an important and distinguishing aspect for IT infrastructures in industry, which is neither covered by the morphological field nor the presented reasoning process model: the central role of human and intellectual factors.

Basically, an IT infrastructure in industry is perceived as a combination of "physical assets, intellectual assets, shared services, and their links" [268]. McKay et al. [274] presented in 1990 a consolidating three-layered model of this combination, Figure 8.3 summarizes from bottom to top:

- **Technical Commodities** Comprises mainly general purpose commodities readily available in the market place, like hardware platforms, operating systems, network and telecommunication technologies.
- **Human Skills** Collects human knowledge, skills, and experience related to IT component development, operation, and integration as well as IT related planning, budgeting and managing.
- **Shared IT Services** Is built by the "mortar" of human skills that bounds the technical commodities into functional IT services [141].

The unique fusion of technical and human elements is identified as important source for value creation in a mass of articles (cf. [33, 68, 67, 70, 69, 95, 110, 131, 177, 197, 220, 225, 224, 230, 248, 257, 274, 295, 300, 335, 346, 361, 364, 393, 395, 411, 410, 421]). In an inter-disciplinary research effort, e.g., of human resources, psychology, and computer science, it could be investigated how to cover this "human mortar" or the "human IT-Infrastructure" [410] in the reasoning activity. This would also cover IT infrastructure consumers, since there are "not only technical issues [...] but also more people-centric relating to collaboration and the sharing of resources and data" [189, p. 1029]. Results could identify additional entries in the reasoning function's (co) domain to handle not only technical implications, but also human related aspects. For instance, introducing redundancy to address short-time breakdowns (cf. Section 2.4) could be reasonable from a technical point of view, but the implied need of human skills to achieve this goal could finally render the effort being too expensive.

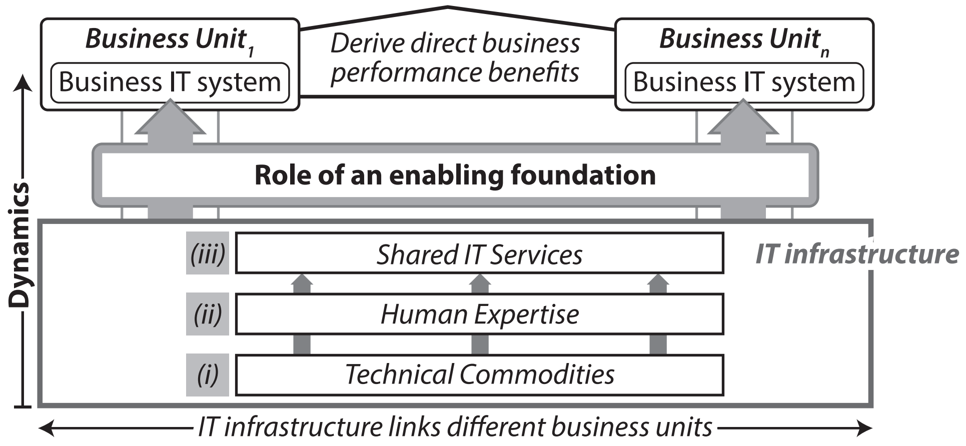

Business Unit₁
Business IT system
Derive direct business performance benefits
Business Unitₙ
Business IT system
Role of an enabling foundation
Dynamics
(iii) Shared IT Services
(ii) Human Expertise
(i) Technical Commodities
IT infrastructure
IT infrastructure links different business units

Figure 8.3: The notion and role of IT infrastructures in industry.

8.2.3 Cloud computing

Cloud computing is an emerging provisioning and use paradigm of IT infrastructures [436, 222, 76, 221] (cf. Section 2.2). As Armbrust et al. [25] carry out in their research, there are manifold approaches to define this paradigm [394] and to confine it against clusters (cf. Section 2.2.2), Grids (cf. Section 2.2.3), and other concepts. For instance, it is the "underlying usage or business model" [328, p. 102][149, 436], or the provision of computing facilities "like an utility [310]" [436, p. 8] operated by a third party [149].

All these details aside, the relevant aspect is that "many key features, such as dynamic resource assignment, are only made available through *virtualization* technologies" [436, p. 9]. Generally speaking, virtualization can be understood as the "(recursively) applied mapping of m interfaces of [architecture] layer s to n functionally consistent interfaces of layer $s-1$ using [...] abstraction" [255, p. 13]. In the context of Cloud computing, virtualization provides "the illusion that each [Virtual Machine] has control of a physical machine" [388, p. 385].

Both, the applied abstraction and the resulting illusion, prohibit a clear sight on the employed IT infrastructure. Besides, existing model integration candidates rarely address potential implications and side effects of the abstracting virtualization, which might cause delusive findings. The arising research task covers the identification of virtualization aspects influencing the reasoning outcome, and their incorporation in the presented process model. Supposable starting points are a set of correction parameters, placed in the reasoning function, or the exclusive use of models specialized for virtual environments, dictated as global constraint in action T-A3:A8.

8.2.4 IT infrastructure surrounding

The morphological field in Section 2.5 scopes the research to focus on the IT infrastructure itself and to omit its physical surrounding, e.g., the building or cooling facilities. This scoping is confirmed by the on-going research efforts in terms of delimiting and defining the surrounding, of identifying (relevant) characteristics, and of interacting with related disciplines, like engineering. Collectively, these challenges call for a dedicated research effort that is not within the scope of this thesis.

Nevertheless, some attributes can be described and considered in more detail if reasoning covers also the IT infrastructure's surrounding. A major example is the attribute energy efficiency (cf. Section 2.4.1). Several work underpins and emphasizes the consideration of the IT infrastructure's surrounding, e.g., Barroso et al. state that "typical power delivery inefficiencies and cooling overheads will easily double that energy budget" [34, p. 50]; Auweter et al. declare that "a preparatory step to improve the energy efficiency in HPC consists of the fine-grained assessment of the power consumption of the entire HPC system encompassing [...] also site infrastructure components for cooling, monitoring and power supply" [27, p. 19]. Another recent example is the "4 Pillar Framework for energy efficient HPC data centers" from Wilde et al. [417]. They aim at answering the question, what aspects of an HPC data center play an important role for energy efficiency improvement and identify the building infrastructure as one of the four pillars, especially because it is involved in several energy efficiency related *Key Performance Indicators* (KPI) like *Power Usage Effectiveness* (PUE), *Water Usage Effectiveness* (WUE) or *Carbon Usage Effectiveness* (CUE). Especially the PUE is highlighted by recent organizations, e.g., the PUE is supported by the "Green Grid" [42]. Resulting research tasks are the

1. identification of connection points to the presented process model,
2. extraction of parameters for the reasoning function's (co) domain,
3. analysis of potential impacts on reasoning results, and the
4. incorporation of gained insights in the presented process model by expanding the provenance information model (cf. Section 5.3) and activity template T-A3, especially in *Phase A* (cf. Section 6.1) and in *Phase B* (cf. Section 6.2).

8.2.5 Automation support

A reasoning project exposes a comprehensive and versatile nature, as particularly the compilation of an individual reasoning function covers manifold tasks and areas (cf. Chapter 6). Hence, (semi) automation could ease and assist reasoning project execution. The presented process model recommends three areas for this intent and provides prepared extension points and interfaces, respectively:

- **Data exchange** The data import from third party models and tools:
  - For IT infrastructure modeling in action T-A3:A5, data import functionality could be implemented to avoid model staleness and reduce modeling cost. Data sources are existing management tools, like GLUE or CIM based databases (cf. Section 7.4).
  - For model proxy function creation in action T-A3:A13, data import functionality could be implemented to omit separated measurements. Exemplary sources are Nagios (cf. Section 2.2.3), or an approach from Aguilera et al. [14] who provide data about performance bottlenecks in black-box distributed systems based on passive message tracing and offline analyze algorithms like convolution.

  For both intents, implementation covers data reading, potential data preprocessing, and suitable data use, respectively. Especially for the first task, the provenance information model provides several dedicated starting points (cf. Section 5.4.2).

- **Code generation** The automated tool creation for executing certain actions within activity template T-A3. For instance, a *Model Driven Development* (MDD) based process (↗KB p. 260) could generate a graphical editor for IT infrastructure modeling in action T-A3:A5, or parallel code from the reasoning function to conduct optimization runs in action T-A3:A19 on an HPC cluster.

- **Semantic checks** The flexible and sustainable encoding of two semantic concepts to facilitate their computer-aided processing:

  Formulas Covers the encoding of (mathematical) formulas that are employed for value aggregation (cf. Section 6.1.3) or quantity description (cf. Section 5.3.2), for instance.

  Assumptions Covers the encoding of assumptions made in action T-A3:A7, especially to assist their computer based evaluation for plausibility, contradictory, and integrity.

For both intents, the provenance information model provides a dedicated class for direct result incorporation, namely the `Formula` class in the `datatype` package (cf. Section 5.3.2), and the `Assumption` class in the `reasoningproject` package (cf. Section 5.3.4), respectively.

8.2.6 Generation of modification recommendations

The presented process model operates *before* a modification's investigation and its potential execution, as the process model reasons about quantitative IT infrastructure attribute values and conflicts that might be induced by the modification. Based on the reasoning results, the modification is further investigated or directly initiated (cf. Use Case UC-6 in Section 3.3.2).

There are several ways to achieve a particular attribute value that has been calculated by a reasoning project. Reliability, for instance, can be achieved by increasing redundancy or introducing checkpointing (cf. Section 2.4). This diversity could be addressed by an automated recommendation or *referral system* using a reasoning project's results. This referral system would have to cover reciprocities, cause-effect-chains, weightings, and cost functions for attributes and modifications to compile sustainable and thorough recommendations, which modification(s) to choose or to execute in order to achieve the attribute vector compiled by activity template T-A3. The challenge of achieving the outlined referral system is to tackle the conflicting dimensions of flexibility, universality, and the plethora of details regarding a particular situation as well as the clear-sighted definition of "good" target values [1].

8.3 Closing remarks

The thesis presents not only a process model for the integrated reasoning about quantitative IT infrastructure attributes, but also validates the presented results to underpin research result's quality, effectiveness, and utility [36, 231, 434, 187]. Validation consists of three pillars:

- **Feasibility** A *controlled experiment* illustrates the feasibility and broad applicability of the presented process model (cf. Section 7.2) by executing a complete reasoning project about the power consumption and performance of a custom-built *RaspberryPi cluster*, assembled of 20 of-the-shelf RPis in the laboratory of the author's chair.
- **Use** A *field study* evidences that the process model fulfills and satisfies "the requirements and constraints of the problem [they were] meant

to solve" [187, p. 85] (cf. Section 7.3), i.e., 17 functional and eight non-functional requirements in the RS evaluation tool (cf. Section 3.5).

- **Relevance** A *related work analysis* underpins the research's level of innovation by comparing the presented process model to existing related work (cf. Section 7.4), which is split in several groups distinguished by the pursued objectives and the applied level of granularity.

Appendices

Appendix A

Knowledge Base

The appendix contains the Knowledge Base of the Design Science paradigm that underlies the presented process model (cf. Section 1.4). It provides "raw materials from and through which [...] research is accomplished" [187, p. 80], and the foundation for rigorous design science research [186]. The Knowledge Base furnishes elementary frameworks, instruments, models, and methods to the *Design Cycle*, and methodologies and guidelines to the *Rigor Cycle* [187]. The following material is provided in alphabetical order in an extend suitable for the thesis:

- Design Science (p. 253)
- Grammar (p. 255)
- Measurement (p. 256)
- Meta Model Hierarchy (p. 259)
- Model Driven Architecture (p. 260)
- Optimization (p. 262)
- Quantity (p. 263)
- Requirements Analysis (p. 264)
- Scale (p. 267)
- Statistics (p. 268)
- Unified Modeling Language (p. 270)
- What-if analysis (p. 276)

Design Science

Design Science is a prescriptive discipline that builds and evaluates IT artifacts intended to attain goals and to meet or solve identified needs or problems, respectively [368, 187]. Hevner et al. [187, 186] developed a framework for "understanding, executing, and evaluating [Design Science] research" [187, p. 79]. Based on Zelkowitz et al. [434, 435], the framework provides, amongst others, a taxonomy that splits methods for IT artifact evaluation in five groups, overviewed in Table A.1 (taken from [187, Table 2]).

1. Observational	
Case Study	Study IT artifact in depth in business environment.
Field Study	Monitor use of IT artifact in multiple projects.
2. Analytical	
Static Analysis	Examine structure of artifact for static qualities (e.g., complexity).
Architecture Analysis	Study fit of artifact into technical information system architecture.
Optimization	Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior.
Dynamic Analysis	Study artifact in use for dynamic qualities (e.g., performance).
3. Experimental	
Controlled Experiment	Study artifact in controlled environment for qualities (e.g., usability).
Simulation	Execute artifact with artificial data.
4. Testing	
Functional (Black Box) Testing	Execute artifact interfaces to discover failures and identify defects.
Structural (White Box) Testing	Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation.
5. Descriptive	
Informed Argument	Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility.
Scenarios	Construct detailed scenarios around the artifact to demonstrate its utility.

Table A.1: Methods for IT artifact evaluation, provided by the *Design Science* framework of Hevner et al. [186] (taken from [187, Table 2]).

Grammar

Mainly applied in the discipline of theoretical computer science, a *grammar* G describes syntactical *production rules* P of how to form strings from an *alphabet* $\sum$. The production rules are applied on the alphabet by replacing occurrences of a rule's left side by its right side in a string. The result is called a *language* being each arbitrary (and infinite) sub set of $\sum^*$. Formally speaking, a grammar is a four tuple $G = (V, \sum, P, S)$ having the following characteristics [355]:

V A finite set of non terminal symbols, also called *variables* as placeholders for syntactical items;

$\sum$ A finite set of terminal symbols, also called *alphabet*, fulfilling $V \cap \sum = \emptyset$;

P A finite set of syntactic *production rules*;

$S \in V$ Start variable;

Noam Chomsky defined four groups according to a grammar's production rules shape:

- **Type 0** Production rules do not have to fulfill or respect any constraints;
- **Type 1** Production rules of a type 1 or *context sensitive* grammar describe rules of the form $uAv \rightarrow uxv$, saying that A is only replaced by x if A is *in the context* of u and v;
- **Type 2** All production rules of a type 2 or *context free* grammar do not define any context-sensitive replacements;
- **Type 3** Extends type 2 grammars by constraints about the right side of the defined production rules, stating that every rule's right side is either a single terminal symbol or a terminal symbol and a variable.

In contrast to the outlined syntactic procedure, a grammar does not describe any semantic aspects or meanings of the strings. Instead, a grammar can be used to generate a language or to recognize a string as language member, well-known as automata theory. The thesis uses the former function and applies the *(Extended) Backus Naur Form* ((E)BNF), a compact notation of type 2 grammars [355]. Table A.2 overviews the EBNF notation elements, providing a title, the summarized rules, the short EBNF notation, and an explanation from top to bottom, respectively.

Merge	Option	Recursion
$A \rightarrow \beta_1$ $A \rightarrow \beta_2$ $A \rightarrow \beta_n$	$A \rightarrow \alpha\gamma$ $A \rightarrow \alpha\beta\gamma$	$A \rightarrow \alpha\gamma$ $A \rightarrow \alpha B\gamma$ $B \rightarrow \beta$ $B \rightarrow \beta B$
$A \rightarrow \beta_1 \vert \beta_2 \vert \beta_n$	$A \rightarrow \alpha[\beta]\gamma$	$A \rightarrow \alpha\{\beta\}\gamma$
Summarizes several rules having the same left side.	The word β can optionally be inserted betw. α and γ.	The word β can be inserted zero to n times betw. α and γ.

Table A.2: The three short notations of the *Extended Backus Naur Form* (EBNF) to describe type 2 grammars.

Measurement

The Knowledge Base entry summarizes measurement concepts, terminology, and approaches for its employment in the thesis. In contrast, the Knowledge Base entry does not aim at providing an exhaustive discussion of the very broad field of measurement in general, and refers the reader for detailed information to the mass of literature, ranging from natural science philosophy by Carnap [80] to physical fundamentals formalized in international standards, like the DIN 1319 [35].

A *measurement* "perform[s] certain operations to determine the value of a [measurand]" [35, p. 3], which is a "physical quantity" [35, p. 2]. Generally speaking, a measurement is the reasonable, exclusively descriptive, assessment-free, and rule-based mapping of a finite set of characteristics at a discrete point in time t on a symbol set. Figure A.1 (based on [55, 56, 80], lower part taken from [35, Figure A.1]) arranges the definition's elements and their interplay and highlights the three structuring realms top-down:

- **Facts** Contains the objects being measured [35], also called the *objects of measurement*. Each object owns a (theoretically) infinite set of characteristics, describing an aspect of the object's nature. Characteristics are considered as facts, since they are as they are, independently from a project, measurement or task.

- **Measurement** Enables a (reasonable) processing of the variety and extent of facts by reducing information's complexity, dimensions, and scale to a manageable extent [55, 56]. This abstraction is achieved by

1. (implicitly) reducing the set of considered characteristics, and
2. mapping the remaining characteristics' values at a discrete point in time on a *scale* (↗KB p. 267) [312, 352, 382] (cf. [352]).

Besides, the realm contains a (potential) set of influence quantities that are "not subject of measurement but which [...] have an influence on the value of the measurand" [35, p. 6], and an error component causing a deviation between a measurand's true and indicated value.

- **Result** Is the "value attributed to the measurand, which is normally indicated by a measuring instrument or system" [35, p. 8] and collectively defined by the above itemized elements. The *measurement result* at the bottom of Figure A.1 is a super set of measured values, containing all measured values "obtained in a series of n measurements under repeatability conditions" [35, p. 8]. The measured values often range around an expected value $\overline{x}$, which is the value "approximated by the mean of all measured values" [35, p. 8], i.e., the result of measurement. Figure A.1 depicts the measured values distribution as normal distribution, the involved error values e_x and the employed measuring instrument(s) are detailed subsequently.

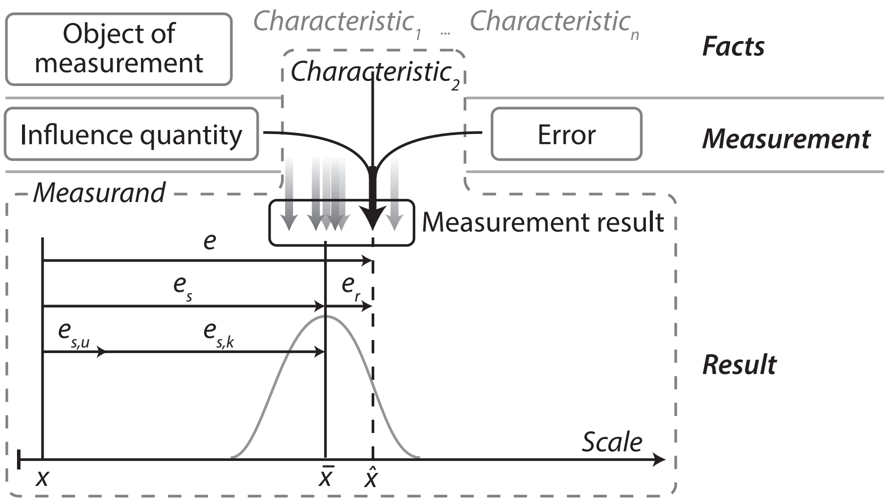


Figure A.1: Measurement concepts and elements (based on [55, 56, 80], lower part taken from [35, Figure A.1]).

Deviation from a measurand's true value

A measurement's error e is the difference between a measurand's $\hat{x}$ attributed value and true value [35]. Equation A.1 (cf. [35, 377]) and Figure A.1 depict the constitution of $\hat{x}$, incorporating the influence quantity and error of the measurement realm. x is the true value that would be obtained by a perfect measurement [35] and that is "obfuscated" by e. It consists of the not precisely known random error e_r and the systematic error e_s, which in turn consists of a known component $e_{s,k}$ and a unknown component $e_{s,u}$.

A measurement's absolute error also determines its *uncertainty* describing the range of "values within which the true value [x] of a measurand is estimated to lie" [35, p. 11]. According to the ISO/GUM code of practice [251], uncertainty describes the variations of obtained measured values. For instance, an uncertainty of 1% means that $|\frac{\hat{x}-x}{x}| \leq 0.01$, $\hat{x} \in [0.99x, 1.01x]$, and $\hat{x} \in [\frac{\hat{x}}{0.99}, \frac{\hat{x}}{1.01}]$. An explanation about the differing deterministic and stochastic uncertainty can be found in Stiller [377], further standardization in part 3 and 4 of DIN 1319 [154, 155].

$$\begin{aligned} \hat{x} &= x + \underbrace{e_r + \underbrace{e_{s,k} + e_{s,u}}_{e_s}}_{e} \\ e &= \Delta x = \hat{x} - x \\ \frac{e}{x} &= \frac{\Delta x}{x} = \frac{\hat{x} - x}{x} \end{aligned} \tag{A.1}$$

Measuring system

Gaining a measurand's values employs one or multiple *measuring instruments*, which is a "device used alone or together with other devices to perform a measurement" [35, p. 15]. It consists of a set of specialized components, like a *sensor* or an *amplifier*. Figure A.2 depicts the components of a common measuring instrument and their interplay.

A measuring instrument in turn is used to assemble a *measuring system*, which is a "complete set of measuring instruments and any other equipment used to carry out a measurement" [35, p. 15]. A "series of elements in a measuring instrument or system, constituting the path of a measurement signal from the input to the output" [35, p. 15] is entitled a *measuring chain*.

Using a measuring instrument and in particular its sensor, might require *instrumentation*, which is "the process of adding probes to generate

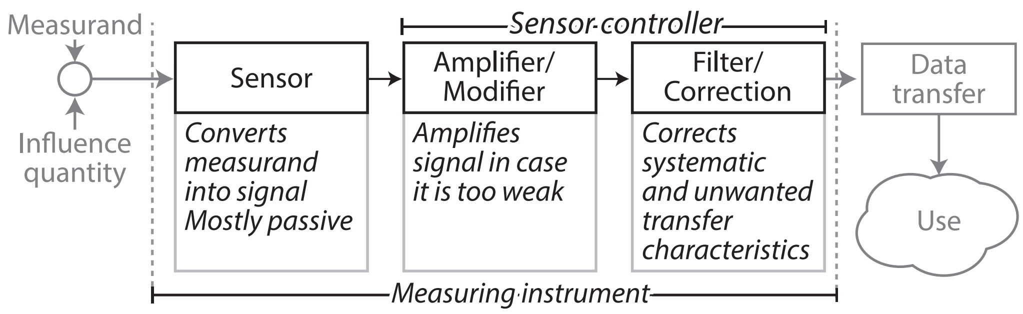

Measurand
Influence quantity
Sensor-controller
Sensor
Converts measurand into signal Mostly passive
Amplifier/ Modifier
Amplifies signal in case it is too weak
Filter/ Correction
Corrects systematic and unwanted transfer characteristics
Data transfer
Use
Measuring instrument

Figure A.2: Components of a measuring instrument [377, 35].

monitoring event data" [194], or generally speaking, the execution of modifications to enable measuring [55]. Exemplary instrumentation is the physical incorporation of a sensory tool's sensor in the observed hardware, or the adaption of software by including additional libraries and modifying source code. Both can be very extensive, e.g., using Intel's *Running Average Power Limit* (RAPL) counters requires not only incorporating `enopt_*` calls in the code [283], but also the *Performance Application Programming Interface* (PAPI) library [73].

Meta Model Hierarchy

A meta model hierarchy consists of three layers [302] (*EG-A.1*), as depicted in the left side of Figure A.3 (partially taken from [302, Fig. 7.8]):

- **Meta Model** Provides language elements to the model and defines the semantics for model element instantiation [302]. It can be used by several models that share the same language elements and concepts.
- **Model** Describes the modeled objects in an abstracting way by focusing on the relevant elements and attributes while omitting all other details [334, 433, 324].
- **Modeled objects** Comprise the elements of interest, e.g., real world physical elements (a car) or concrete object-oriented classes (Car).

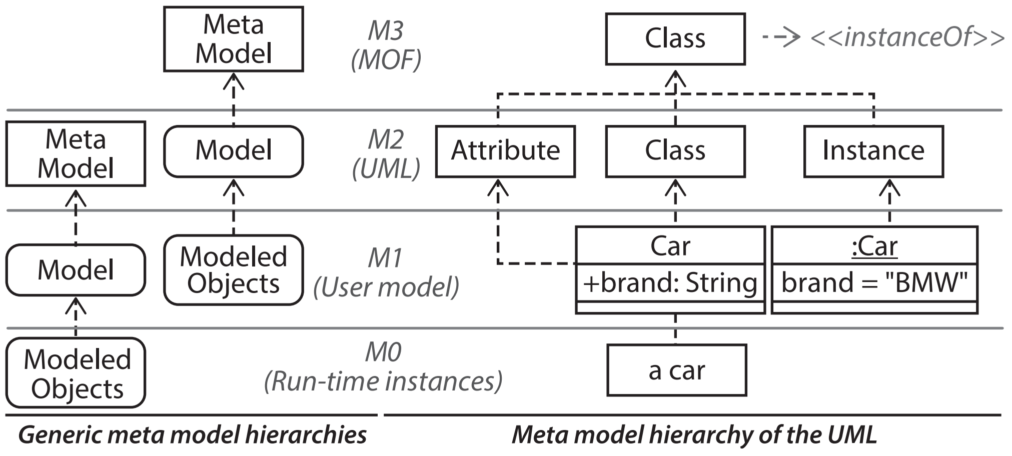


Figure A.3: The common three layered meta model hierarchy and its exemplary application in the UML context.

EG-A.1

Figure A.3 depicts on its right the UML underlying meta model hierarchy: the meta model in layer "M2" provides the language element *Class* to the layer "M1" that models with it a *Car*, which in turn abstracts the real world element *a car* in layer "M0". The example also illustrates the hierarchy's recursive manner, i.e., a model can be used as a meta model [302]: the UML language elements in layer "M2" act as meta model for the user model in layer "M1", and at the same time are an instance of the meta model *Meta Object Facility* (MOF) [301] in layer "M3".

Model Driven Architecture

The *Model Driven Architecture* (MDA) is an incremental software development approach [150, 232, 326] that provides an "infrastructure for arbitrary [software development] methodologies" [315, S. 48]. MDA aims at automatically generating genuine source code by processing and transforming a set of models at different abstraction levels. The transformation rules, also called *mapping*, are defined for the meta models of the source and target model, respectively [315, 279, 169, 237], which renders a meta model hierarchy a mandatory prerequisite for MDA [270]. The three following concepts establish its name:

- **Model** In MDA, models do not only perform (partial) system description and documentation tasks, but are considered equally to code and possess a central and active role [376].

- **Driven** Precise, machine readable models at different abstraction levels and (automated) model transformations *drive* the entire software development process [357, 315]. Only the process' very last step generates and occasionally manually extends genuine software code.

- **Architecture** An architecture specifies a system's parts and connectors and defines rules for part interactions based on the specified connectors [285]. MDA considers the aspired software product as system and prescribes the use of certain model kinds as well as their preparation, relationships, and interaction [285], instead of dictating particular methodologies or modeling languages.

The alluded automated generation of software code based on model(s) is achieved by looking at a (software) system not from only one perspective, but by applying four views, "a representation of that system from the perspective of a chosen viewpoint" [198]. The views' characteristics and interactions as well as examples (incorporated from [315]) are depicted in Figure A.4 (developed from [285, 315]) and subsequently described (*EG-A.2*):

- **CIM** The *Computation Independent Model*, also called *domain model* or *business model* [334, 433, 324], describes a system and requirements from a hardware and software agnostic perspective, mostly in natural language. It pursues a common terminology and a well-founded understanding of the developed system [315] while hiding (all) information related to the use of automated data processing systems [285].

- **PIM** The *Platform Independent Model* formalizes the CIM in a technical but still platform independent way. The CIM specification's mostly natural shape forces a *manual* CIM to PIM transformation (cf. Figure A.4) [315].

- **PSM** The *Platform Specific Model* describes the system for a specific platform, which is represented by a *Platform Model* (PM) that describes the platform the aspired software product will be executed on. The PIM's high formalization level allows an automated transformation consuming both, the PSM and the PM.

- **PSI** The *Platform Specific Implementation* is the genuine source code. The high formalization level of a PIM allows an automated transformation. Although source code can be understood as a model, too, the differing abstraction levels of PSM and PSI require a distinction to allow manual adaptions accordingly.

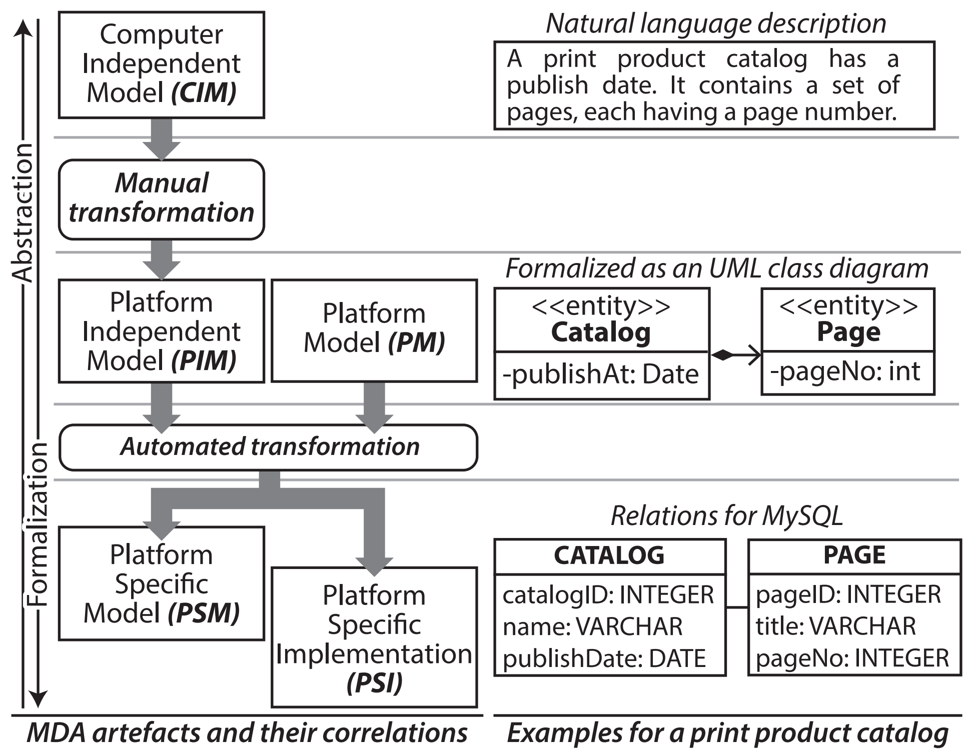

Abstraction
Formalization
Computer Independent Model (CIM)
Manual transformation
Platform Independent Model (PIM)
Platform Model (PM)
Automated transformation
Platform Specific Model (PSM)
Platform Specific Implementation (PSI)
Natural language description
A print product catalog has a publish date. It contains a set of pages, each having a page number.
Formalized as an UML class diagram
<<entity>> Catalog
-publishAt: Date
<<entity>> Page
-pageNo: int
Relations for MySQL
CATALOG
catalogID: INTEGER
name: VARCHAR
publishDate: DATE
PAGE
pageID: INTEGER
title: VARCHAR
pageNo: INTEGER
MDA artefacts and their correlations
Examples for a print product catalog

Figure A.4: The four MDA views on a system and their correlations as well as an exemplary MDA model set for a printed product catalog.

EG-A.2 Figure A.4 exemplary illustrates the discussed MDA system views for a print product catalog. The CIM "models" the catalog to be printed by describing in natural language its nature. These information are manually transformed in a PIM that formalizes the information as UML class diagram, which is a platform independent formalization. This high level of formalization allows an automated transformation in a MySQL database scheme, a platform specific implementation.

Optimization

Several (research) disciplines employ the concepts and algorithms of (mathematical) optimization, some of them even longer than computer science. All optimization problems share the mandatory definition of two elements:

- **Search space** A set S of objects available to or valid for the optimization selection, also called *choice set.* The set can be (in)finite and specified by a function or not. In either case, it must be clearly specified [168].
- **Objective function** The function φ *quantitatively* rates elements of S. Depending on the applying discipline and the optimization characteristic, the function is also called a *cost function* or *indirect utility function* for minimization problems, or a *utility function* or *fitness function* for maximization problems.

Employing these elements, an optimization problem aims at finding an *optimal solution*, which is an element x^* in the optimization problem's *feasible region* F. The element $x^* \in F$ is optimal in a sense that $\varphi(x^*) \le \varphi(x) \forall x \in S$ for a minimization problem or such that $\varphi(x^*) \ge \varphi(x) \forall x \in S$ for a maximization problem. Equation A.2 arranges the elements in a formal way and generically specifies an instance of an optimization problem [168].

$$n \in \mathbb{N} \wedge F \subset S \subset \mathbb{R}^n \wedge \varphi : S \mapsto \mathbb{R} \wedge \text{opt} \in \{\text{min, max}\} \qquad \text{(A.2)}$$

Depending on the particular objectives and characteristics of the optimization problem instance, there are several classes, of which the following list can only provide a small subset:

- **Evaluation problem** Identify the infimum or supremum for $\{\varphi(x) : x \in F\}$ for a minimization or maximization problem, respectively.
- **Feasibility problem** Check if $F = \varnothing$ is valid.
- **Linear optimization problem** An optimization problem working on vectors and matrices.
- **Mathematical programming** An optimization problem $max\varphi(x) : x \in S, g(x) <= 0, h(x) = 0$, specifying $g(x)$ and $h(x)$ as constraints. The data type of S, the constraint functions, and the solving behavior assemble several sub classes, like biconvex or dynamic problems.

Quantity

A quantity can be regarded as a countable or comparable property or aspect of an entity. It implicitly contains a scale (↗KB p. 267) that provides the building blocks for counting and comparison. There are three quantity classes:

- **Primary** A primary quantity, like *time* or *distance* [80], is indispensable for the definition or measurement of other quantities.

- **Additive** An additive quantity is assembled of primary quantities having the *same* scale (↗KB p. 267). An additive quantity also inherits the creation rules of the processed primary quantities.

- **Derived** A derived quantity is assembled of primary quantities having *different* scales [80], e.g., combining *Megabit* and *seconds* to *MBit/second*. An important aspect is that derived quantities are average quantities, since they do not consider a particular point in time, but a time frame ("per second" can only be an average). Hence, deviation is required to get a discrete point in time, e.g., $\frac{dMBit}{dt} = \lim_{\Delta t \to 0} \frac{\Delta MBit}{\Delta t}$.

Requirements Analysis

Based on existing, established glossaries, like the *IEEE Standard Glossary of Software Engineering Terminology* [200], the Knowledge Base entry summarizes concepts and representation syntax, the *Relevance Cycle* uses in Chapter 3 for developing a *Requirements Specification* (RS). It employs the following elements that are ordered according to their mutual use:

- **Requirement** A requirement is a "condition or capability needed by a user to solve a problem [...] and that must be met or possessed by a system or system component" [200, p. 62]. The employed keywords "must", "must not", "shall", "shall not", "should", "should not", and "may" are to be interpreted as described in RFC 2119 [64]. Requirements can be split in two groups:

  Functional Describes a concrete function that can be implement, which means "a requirement that specifies a function that a system or system component must be able to perform" [200, p. 35].

  Non-functional All requirements that are "not functional" [342], in particular ancillary conditions and quality criteria of the developed system. Also non-functional requirements have to be measurable to enable fulfillment validation [233].

- **Stakeholder** A single person or institution that formulates requirements [233], and interacts with the considered system.

- **System** Implements and provides functionality and behavior, which is exposed to and used by the stakeholders.

- **Actor** Represents a role, human, sensor, or any other technical device that interacts with the system [344, 303]. The actor sending a request to the system is called *primary actor* [100]. Table A.3 depicts the *actor specification template* that is used for actor description in the RS according to related literature, like Marcu [267].

- **Use Case** In a generic form, a Use Case can be understood as "a contract between the stakeholders of a system about its behavior" [100, p. 1]. It addresses several criteria a RS should meet and is a suitable tool to derive requirements. In particular, it "tells *coherent* stories about how the system will behave in use" [100, p. 15] and each described action of the system "yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system" [303, p. 606].

- **Representation** System, actor, Use Cases, and their interactions can be described in may different ways, e.g., text form, flow charts, Petri Nets, or programming languages [100]. As requirements engineering literature recommends, the RS uses the following formats:

  Use Case template Kleuker et al. [233] state a set of quality criteria Use Cases should comply to. Correspondingly, the Use Case template in Table A.4 (combined from [100, 233]) dictates from top to bottom an unambiguous identification, and a general description of objectives as well as a list of involved actors, of abstraction sources, and of pre and post conditions relative to the Use Case's execution.

  Use Case diagram The Use Case diagram is part of the *Unified Modeling Language* (UML) syntax (↗KB p. 270) and represents in a graphical way the system, actors, Use Cases, and their interactions, as Figure A.5 exemplifies. It represents the system as a rectangle labeled with a short name [344], the actor as a stick man [344], and Use Cases as ellipses with a name at the center. Besides, Figure A.5 illustrates the three Use Case relation types that aim at enabling an efficient description of commonalities and at avoiding redundancy [233]. A solid unlabeled arrow represents common object oriented inheritance, stating that an element inherits all attributes and relations [344]. An `include` relation describes the non-optional inclusion of Use Case B by Use Case A: Use Case A imports the behavior and functionality of Use Case B and hence, B is mandatory for A's success [344, 233]. An `extend` relation represents the extension of Use Case A by Use Case A_2 if the assigned condition evaluates to true [233].

Actor	***Human-readable name of the actor***	Unique Identifier
Description of the actor, its duties, responsibilities, and characteristics.		
→ **Abstraction source** – Describes details the actor is abstracted from. Uses the black circle (❶) to reference details in the research *Environment* (cf. Chapter 2) and in scenario descriptions (cf. Section 3.2).		

Table A.3: Actor specification template.

Use Case	***Use Case name***	Unique identifier
General description of the Use Case and its objectives.		
ACT-1 Involved actor(s) and their duties.		
→ **Abstraction source** – Describes details the Use Case is abstracted from. Uses the black circle (❶) to reference details in the research *Environment* (cf. Chapter 2) and in scenario descriptions (cf. Section 3.2).		
PRE ☐ Pre-conditions for Use Case execution.	☐ Post-conditions after Use Case execution, e.g., achieved results. ***POST***	

Table A.4: Use Case specification template.

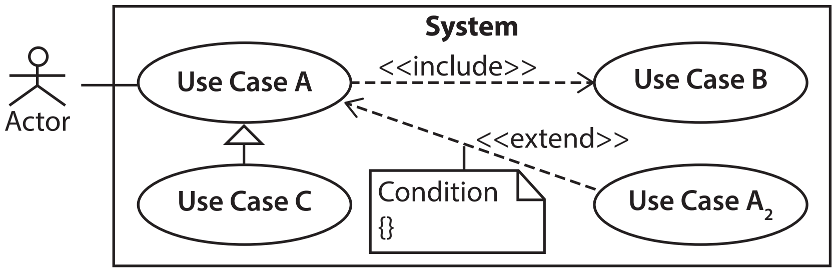


Figure A.5: UML compliant graphical representation of Use Cases, the providing system, and involved actors.

Scale

A *scale* structures or splits a domain in categories or blocks, respectively. Figure A.6 exemplary illustrates the structuring of real numbers ($\mathbb{R}$) using the scale *centimeters* in blocks of the same size. According to the scale's expressiveness and delimitation, there are the three following scale classes [179], whose containment hierarchy is emphasized by the formalizing equations, denominated for each class, respectively:

- **Nominal (categorical)** Describes the (simple) assignment of objects to a qualitative category according to a considered feature ($F(object)$). Thus, a nominal scale contains a (limited) category set, *denominates* a particular object's category, and labels objects being either *equal* or *unequal* in their category [375]. The scale's expressiveness is positively correlated to the category width: the higher the focus of the used categories, the higher is the (implicitly) contained information. This information gain approach is often employed in taxonomies, e.g., botany zoology uses *animal*, *vertebrate*, *mammal*, *dog*, *poodle* (taken from [375, p. 5]). Equation A.3 formalizes a nominal scale.

$$Equal(objectA, objectB) := (F(objectA) = F(objectB)) \tag{A.3}$$

- **Ordinal (comparative)** Extends the nominal scale by defining relationships between objects in different categories. The relationship can be used to *compare* two objects, e.g., *warmer* and *colder*. Hence, an ordinal scale is a tuple consisting of a category set and an order relation on them. Noteworthy, the order relation describes only a hierarchy, but no distances. An exemplary ordinal scale is the transport quality of fruits, consisting of A, B, and C (taken from [375, p. 6]). Collectively, Equations A.3 and A.4 formalize an ordinal scale.

$$Compare(objectA, objectB) := (F(objectA) < F(objectB)) \tag{A.4}$$

- **Metric** Extends the ordinal scale by defining distances between categories that enhance the information of *bigger/smaller* with a discrete value. A metric scale carries the most information, as it assigns a category (*nominal*), states a category hierarchy (*ordinal*), and also describes distances between categories and elements. An exemplary metric scale is *kilometers per hour* (taken from [375, p. 7]). Collectively, Equations A.3, A.4, and A.5 formalize a metric scale.

$$\begin{aligned} MinVaue &:= x_{min} \in \mathbb{R} \\ MaxVaue &:= x_{max} \in \mathbb{R} \\ Distance(objectA, objectB) &:= (F(objectA) \odot F(objectB)) \end{aligned} \tag{A.5}$$

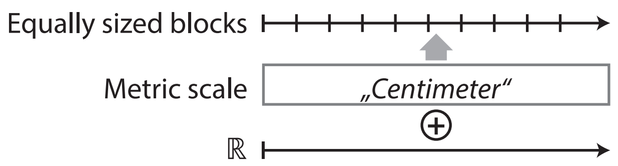


Figure A.6: Structuring of real numbers (ℝ) in the metric scale *centimeters*.

Statistics

Statistics is the "branch of scientific inquiry that provides methods for organizing and summarizing data, and for using information in the data to draw various conclusions" [120, p. 1]. The considered data is called the *population*, a portion or subset of the population is called a *sample* (*EG-A.3*).

EG-A.3

> Assuming that the population is all U.S. colleges and universities, a sample would be {Stanford University, Oberlin College, Iowa State University} (taken from [120, p. 1]).

Three sub branches organize statistics. Their description below focuses on aspects relevant for the thesis. For further reading please refer to groundwork literature, like Devore [120], Casella et al. [87], or Specht et al. [375].

- **Descriptive** Covers methods for organizing and summarizing data sets [287] to process and explain a sample or a list of all population members, and to identify correlations within the data using tables, charts, and statistical parameters [375].The branch consists of *visual representation* and *numerical (summary) measures*. The former contains methods like *Stem-and-Leaf Displays* (cf. [120, p. 9]) or *Histograms* (cf. [120, p. 13]), but is not further investigated due to the above alluded focus. The latter is further detailed in Table A.5 (summary of [287, 120, 375]) and Figure A.7 and consists of two groups:

  Measures of location Characterize the data set, and convey some of its salient features by assembling summarizing numbers [287]. This summarizing nature implies that measures of location do "not necessarily provide enough information" [287, p. 23] to describe the data set exhaustively.

  Measures of variability Enhance the provided location information by additionally describing the data set's behavior in relation to the measures of location [375].

- **Inferential** Draws conclusions about a population based on a (small) sample [375]. For instance, an engineer designing a new computer chip manufactures a prototype and might want to draw conclusions about device collaboration once they are in full-scale production [287].
- **Stochastic** Serves as link between descriptive and inferential statistics, as it provides the necessary probability tools to both branches.

1	**Absolute and relative frequency**	n_i, $h_i = \frac{n_i}{n}$
	Amount of observations having the same value absolutely (n_i) and in relation to the data set's size (h_i).	
2	**Cumulative absolute frequency**	$N_i := \sum_{j=1}^{n} n_j$
	Count of observations whose value is equal to x ($F_n(x)$), bigger than x ($1 - F_n(x)$), or between x_1 and x_2 ($F_n(x2) - F_n(x1)$).	
3	**Mean/Expected value**	$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$
	The data set's arithmetic average, and a measure of location.	
4	**Median**	$\tilde{x} = \begin{cases} x_{([n+1]/2)} & n \text{ odd} \\ \frac{x_{(n/2)} + x_{([n/2]+1)}}{2} & n \text{ even} \end{cases}$
	The middle value of an ascending ordered sample, i.e., point that divides the sample in two equal halves [287].	
5	**Mode**	
	Observation that occurs most frequently.	
6	**Quartiles**	q_1, q_2, q_3
	Divide data in four equal parts. Second quartile is the median.	
7	**Sample range**	$r = max(x_i) - min(x_i)$
	Difference between the largest and smallest observation.	
8	**Standard Deviation**	$\sqrt{(a_i - \overline{a})^2}$
	Deviation of the ith observation from the mean, and a measure of dispersion.	

Table A.5: Overview of descriptive statistics figures and tools.

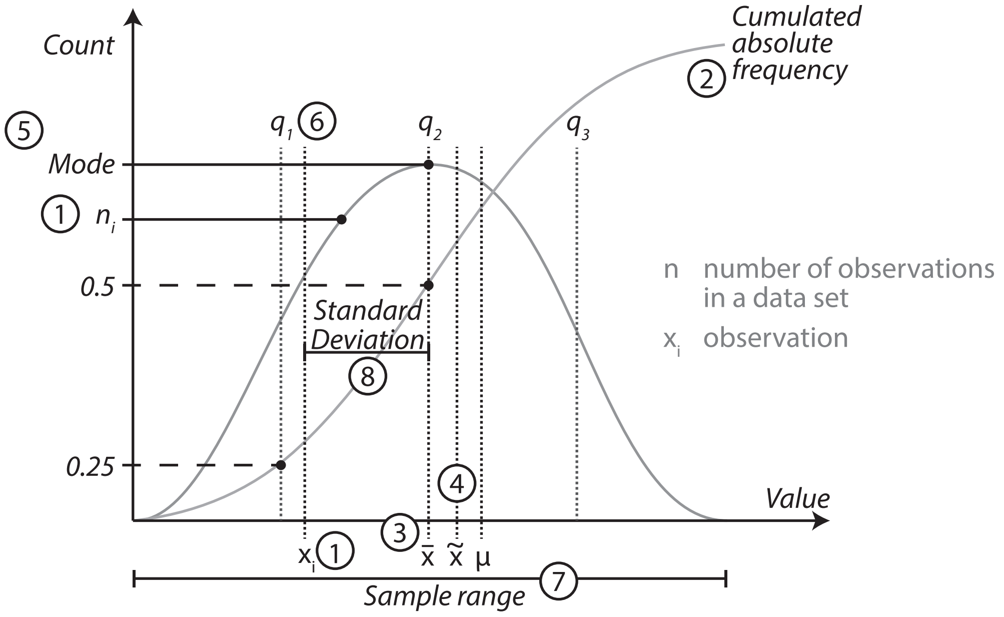


Figure A.7: Overview of descriptive statistics concepts.

Unified Modeling Language

The *Unified Modeling Language* (UML)[1] is a general-purpose, visual language that deals with domain-independent modeling, documenting, specifying, and visualizing complex (software) systems and their artifacts [344, 303]. It aims at "provid[ing] system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes" [302].

UML comes with a diversity of diagram types, each focusing on a specific field. The Knowledge Base entry discusses five diagram types in alphabetical order that are relevant for the thesis. For further detailed reading, please refer to the OMG UML specification [302, 303] and Rupp et al. [344].

- Activity diagram (p. 270)
- Class and object diagram (p. 271)
- Component diagram (p. 273)
- Package diagram (p. 274)
- Use Case diagram (p. 276)

Activity Diagram

In the understanding of the OMG, an activity is "the specification of parameterized behavior as the coordinated sequencing of subordinate units

[1]Presented and maintained by the *Object Management Group* (OMG), `www.omg.org`

whose individual elements are actions" [303, p. 324]. An action is a single step that contributes to the realization of behavior described by the alluded activity [344]. In other words, an action is a "named element that is the fundamental unit of executable functionality. The execution of an action represents some transformation or processing in the modeled system, be it a computer system or otherwise" [303, p.243]. An object is "an abstract activity node that is part of defining object flow in an activity" [344, p. 276].

An activity diagram is a directed graph that provides capabilities to "model and retrace complex processes" [344, p. 259], and to visually represent the alluded activities, covering concurrency, alternatives, conditions, and parametrization. One of its fundamental concepts is the *token* (cf. Petri-Nets [331]) that logically indicates the current position within an activity and whose arrival triggers an action's execution. Figure A.8 depicts an exemplary activity diagram containing the following elements:

- **Action node** Represents an action as rounded corner rectangle. It either represents a not further fractionized "atomic" behavior, or it represents an *activity nesting*, which is additionally labeled by a ⋔.

- **Object node** Represents an object as rectangle. An object node models variables, constants or other results that might be produced by an action. These variables can be defined within the activity, like `Object node` at the bottom of Figure A.8, or outside the activity and passed as parameter, like the `Activity input` on the right side of Figure A.8.

- **Activity edge** Connects action and object nodes, being an "abstract class for directed connections between two activity nodes" [344, p. 282]. They are depicted as arrows that indicate the token flow direction between the connected nodes.

- **Token flow** Is routed alongside defined activity edges and further controlled by *control nodes*. *Decision/merge* nodes define optional or conditional paths, *fork/join* nodes describe a parallel execution.

Class and object diagram

In object-oriented modeling, a class abstracts "a set of objects that share the same specifications of features, constraints, and semantics" [303, p.48] by describing an object's attributes and operations. The former are common characteristics of an object set and consist of, amongst others, name, multiplicity, and data type. The latter is a "behavioral feature of a classifier

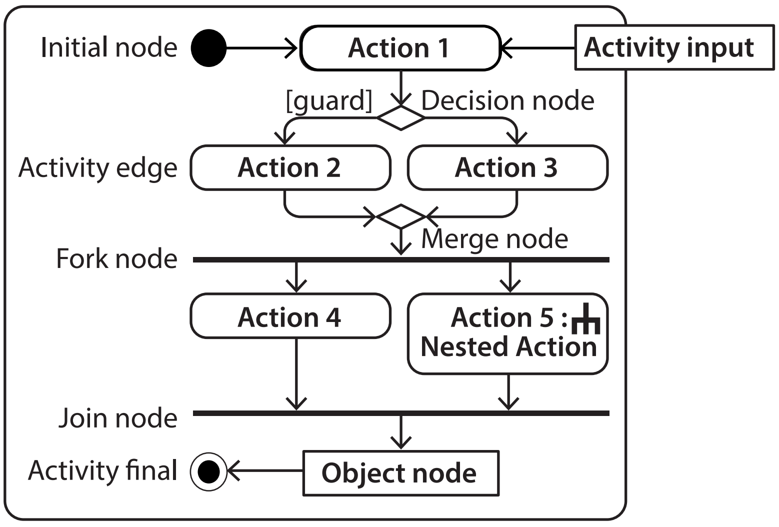


Figure A.8: Exemplary UML activity diagram.

that specifies the name, type, parameters, and constraints for invoking an associated behavior" [303, p. 104]. In contrast to the abstracting class, the object is a non-abstract (real world) instance of a class, allocating concrete values to the class' attributes.

A class diagram provides capabilities to model and describe a system's static properties and their relationships [344] in an object-oriented way, an object diagram is an instance of the class diagram, representing a snapshot of concrete values. Figure A.9 depicts an class and object diagram in its upper and lower part, respectively. It contains the following elements:

- **Class** Represents a class as rectangle, providing the class' name, attributes, and operations from top to bottom, as illustrated in Figure A.9 by class `ClassA`.

- **Generalization** Is a "taxonomic relationship between a more general classifier and a more specific classifier" [303, p.70]. The specific classifier inherits all attributes and operations (features) from the general classifier. As depicted in Figure A.9, `ClassB` (specific classifier) inherits two attributes and two operations form `ClassA` (general classifier).

- **Association** Describes a "set of tuples whose values refer to typed instances" [303, p.36]. In other words, it describes a set of homogeneous class relationships, like the association between `ClassA` and class `ClassB` in Figure A.9. A specialized association is the **aggregation** describing a "consists of" relationship. In Figure A.9, `ClassA` consists of one or multiple ClassD objects, e.g., a group of people (ClassA) consists of humans (ClassC). A stricter form of aggregation is the **composition** that additionally states the lifetime of the contained elements. In Figure A.9, ClassA consists of one or multiple ClassE objects with the additional constraint that all ClassE objects are destroyed when ClassA is destroyed. For instance, a cocktail (ClassA) consists of ingredients (ClassE), but if the cocktail is destroyed, also the ingredients are destroyed (taken from [344, p. 147]).

- **Object** Is a class instance, allocating discrete values to the class' attributes. An instance of a class diagram is graphically represented by an *object diagram*. Figure A.9 depicts in its lower part an exemplary object diagram for the above discussed exemplary class diagram, containing objects (class instances) and links (association instance).

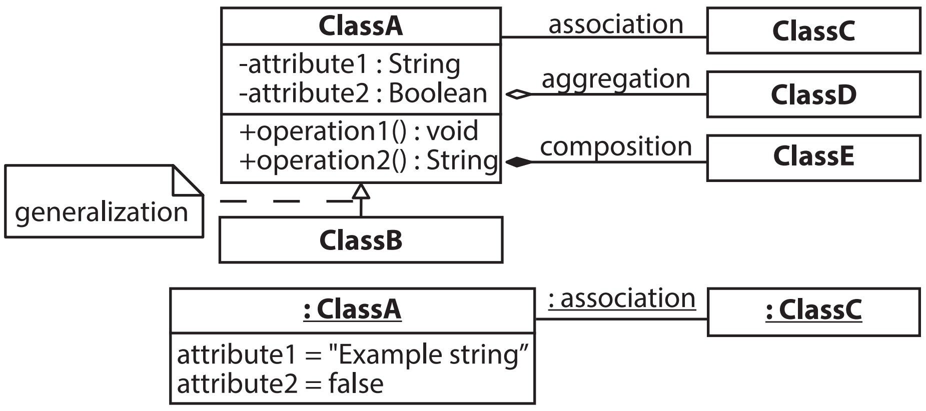


Figure A.9: Exemplary UML class and object diagram.

Component diagram

A component represents a "modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment" [303, p.149], and that has a "concisely confined behavior that is accessible through clearly defined interfaces" [344, p. 212]. Hence, a component defines its behavior in terms of provided and required interfaces [303]. This behavior can be implemented in many ways, and a concrete implementation can be replaced by another, as long as the initially provided set of interfaces is supported. Summarized, a component is not only a data provider, but a self-contained application [296] (cf. [344, p. 217]).

A component diagram provides capabilities for describing a system's structure during run time with a strong focus on physical and technical aspects, and for organizing a system's elements in the alluded components and their dependencies. Figure A.10 depicts an exemplary UML component diagram containing the following elements:

- **Component** Represents a component according to the introduced notion as rectangle and the component symbol ⊟ in its upper right side. Similar as actions in activity diagrams, components can be not further fractionized "atomic" components, represented by `ComponentB`, or describe component nesting, represented by `ComponentA`.

- **Port** Is a "property of a classifier [(component)] that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts" [303, p.186]. In other words, a port encapsulates a component and is exposed for the interaction with a component's surrounding [344]. Ports are depicted by a square.

- **Interface** Connects the alluded ports and specify a contract that has to be fulfilled by any component instance that realizes the interface [303]. In Figure A.10, `ComponentA` realizes `InterfaceA` (ball symbol), which is consumed by `ComponentB` (semi circle symbol). The interface communication is handled by the ports `PortA` and `PortB`. Interfaces are also used to seal off internal components from external access, as shown for `Subcomponent`, which is accessible through `InterfacePrivate` that is delegated to `PortA` of `ComponentA`.

- **Artifact** Represents the implementation of a component and is "the specification of a physical piece of information that is used or produced by a software development process, or by deployment and operation of a system" [303, p.203]. In Figure A.10, the behavior and the interfaces described by `ComponentA` are implemented by the artifact `example.jar`, which in turn uses the artifact `library.jar`.

Package diagram

In UML, a package "is used to group elements, and provides a namespace for the grouped elements" [302, p. 160]. It bundles classifiers, like classes, interfaces, or associations, that are related in some way [344]. The package's namespace is used to extend unqualified identifiers of the contained

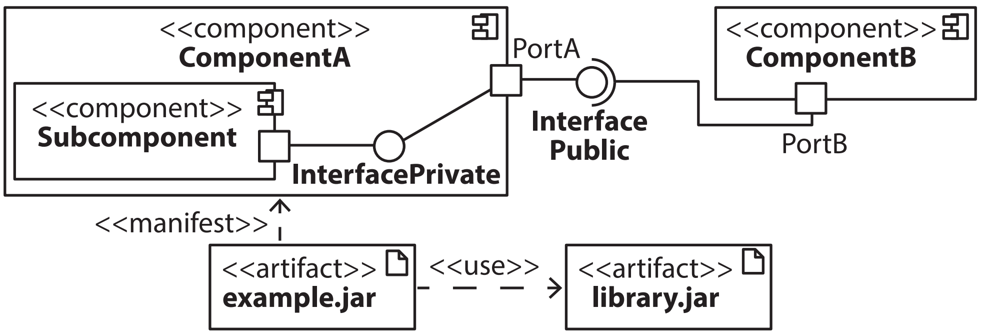


Figure A.10: Exemplary UML component diagram.

classifiers to fully qualified identifiers, consisting of the package's name and the classifier's name. For instance, a class `A` contained in package `P` would have the fully qualified identifier `P::A` (taken from [344, p. 169]). Package structuring is a common concept to organize extensive and complex UML class diagrams, models, or systems, e.g., it is extensively applied in the UML Superstructure specification [303] and it is a common tool for describing the layered structure of software [247] (cf. [344]).

A package diagram abstracts from the alluded classifiers and focuses solely on the bundling packages while omitting detailed insights. Consequently, the main objective of package diagrams is providing an overview of a system. Figure A.11 depicts an exemplary package diagram, consisting of the following elements:

- **Package** Is represented as rectangle, printing the package name in the tab or in the rectangle's header, as Figure A.11 depicts for `Package1` and `Package2`, respectively. Besides, both packages show nesting.

- **Association** Describes a package relation in two ways:

  Import Is a "relationship that allows the use of unqualified names to refer to package members from other namespaces" [303, p. 112]. In other words, it allows a package to use the classifiers of the imported package by making them visible [344]. Figure A.11 illustrates this for package `Package4` that can use the classifiers of `Package3`.

  Merge Defines "how the contents of one package are extended by the contents of another package" [303, p. 113] and implicitly creates new specialized classifiers that can be further adapted [344]. Figure A.11 depicts the (implicit) creation of new classifiers on the right hand side in `Package5'`: the already existing classifier `A` in `Package5` is sub classed by a specializing classifier in the merge resulting `Package5'`.

In addition to this simple example, there are many more merge alternatives, as elaborated by Rupp et al. [344, p. 173].

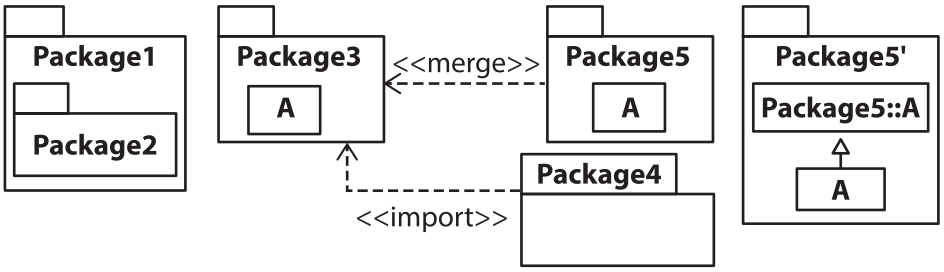


Figure A.11: Exemplary UML package diagram.

Use Case diagram

Is listed for the sake of completeness. It is explained in the context of requirements analysis on page 264 of the Knowledge Base.

What-if analysis

A what-if analysis, also called *sensitivity analysis*, describes the process of changing the used input value set of a function, also called *scenario*, to examine how those changes affect the function's outcome. It is often used to compare different scenarios and their potential outcomes based on changing conditions. In case a cross product of all parameters is used, it is also called a *parameter sweep*.

Appendix B

Benchmark overview

The following list alphabetically overviews benchmarks that are employed in the thesis. Each list entry consists of the benchmark's name, its building blocks (cf. Section 2.3.2), a summary, and a set of paper references that provide the details. Summary description employs the four layers *High level specification* (HLS), *Low level implementation* (LLI), *Compilation* (C), and *Execution* (E) introduced in Section 2.3.2.

Dhrystone (Synthetic)

The Dhrystone benchmark was designed in 1984 at Siemens to measure the Integer performance of small machines with simple architectures. Its name is a wordplay on the Whetstone benchmark, a floating-point performance benchmark popular at that time (cf. below). Based on a literature survey on "the distribution of source language features in non numeric, system-type programming" [407, p. 70], the HLS of the Dhrystone benchmark defines 12 procedures in one measurement loop, called a *Dhrystone.* The LLI slightly differ, as the C and ADA version consist of 103 and 101 statements, respectively. The execution (E) results are usually given in *Dhrystones per second.* RISC machines generally beat CISC machines on the Dhrystone benchmark, because the larger number of registers and the localities of code and data strongly react on Dhrystone's explicit consideration of operand locality [255, 407, 408, 409, 122, 258].

LINPACK (Kernel)

Originally not designed as benchmark, the LINPACK evolved as one of the most famous benchmarks which is heavily used, e.g., to categorize supercomputers [128]. LINPACK is a package (name's eponym) of linear algebra subroutines often used in FORTRAN programs and hence, a kernel

Name	Matrix dimension	Optimization allowed	Parallel processing
LINPACK 100	100	Compiler	Compiler parallelization possible
LINPACK 1000	1000	Manual	Multiprocessor implementations allowed
LINPACK Parallel	1000	Manual	Yes
HPLinpack	Arbitrary	Manual	Yes

Table B.1: Overview of LINPACK versions (taken from Dongarra et al. [127]).

benchmark. Authored by Jack Dongarra in 1976, the benchmark is used to characterize the floating-point performance of machines by *Millions of floating-point operations per second* (MFLOP/s, cf. Section 2.4.2). The HLS of LINPACK defines operations on a large matrix, particularly factoring or decomposing the matrix "into a product of simple, well-structured matrices which can be easily manipulated to solve the original problem" [127, p. 803]. Three configuration parameters influence its LLI (taken from [407]):

Single/double Use FORTRAN single or double precision values.

Rolled/unrolled Optimize loops at the source code level by loop unrolling.

Coded BLAS/FORTRAN BLAS Use the fundamentally important sub package *Basic Linear Algebra Subroutines* (BLAS) in assembly language or as FORTRAN library.

Depending on the matrix dimension, the allowance of optimization, and parallel processing, there are four LINPACK types, overviewed in Table B.1 (taken from Dongarra et al. [127]). The LINPACK 1000 benchmark is also known as *Toward Peak Performance* (TPP) or *Best Effort* version, the *Highly-Parallel LINPACK* (HPLinpack) benchmark is also known as the N $\times$ N LINPACK benchmark [407, 127, 124, 129, 157].

MADBench2 (Kernel)

The building blocks of the MADBench2 benchmark are kernels extracted from a cosmology application analyzing *Cosmic Microwave Background* (CMB) data sets. This is seen as a distinctive feature compared to other I/O micro benchmarks, because it is “derived directly from an important scientific application” [59, p. 1]. Designed as I/O micro benchmark, MADBench2 operates primarily on floating-point matrices that are too large to maintain simultaneously in main memory, what requires several reading and writing operations to and from disk during calculation. MADBench2 succeeds *MADBench*, the authors presented upfront as a lightweight version of the *Microwave Anisotropy Dataset Computational Analysis Package* (MADCAP) [57, p. 119], consisting of algorithms for reducing data sets of CMB measurements [81, 86, 59, 58, 57].

NAS Parallel Benchmark Suite (Kernel)

The *Numerical Aerodynamic Simulation* (NAS) Parallel Benchmark Suite (NPB) is a small set of programs that were designed in the 1990s to “to study the performance of parallel supercomputers” [13, p. 1], to “analyze performance and scalability of hardware platforms” [93, p. 75], and for “testing the capabilities [...] of parallelization tools” [50, p. 3]. The contained benchmarks are derived from and mimic the computation and data movement characteristics of large-scale*Computational Fluid Dynamics* (CFD) applications. The NPB suite initially consisted of five kernels, three pseudo-applications, and eight problem sizes, provided as “pencil-and-paper” specification. Reference implementations use common programming models and languages, like MPI, Java, and Performance FORTRAN [162, 349, 29, 13, 93, 373, 50, 213, 291].

SAGE (Kernel)

The benchmark kernels were extracted from *SAIC’s Adaptive Grid Eulerian* (SAGE) code, a parallel, large-scale, multidimensional (1D, 2D, 3D), multi-material, Eulerian hydrodynamics code with adaptive mesh refinement. It is used in a wide variety of scientific and engineering problems, like water shock, energy coupling, and hydrodynamic instability problems. Hence, the benchmark represents a large class of production *Advanced Simulation and Computing* (ASC) workloads that run on thousands of processors for months at a time. Network bandwidth, network latency, and dimensions of the used torus’ subset equally influence the benchmark’s (communication) performance. Its LLI is provided in FORTRAN90 using MPI for inter-processor communications [112, 31, 192, 226].

STREAM (Synthetic)

The synthetic STREAM benchmark aims at decoupling the memory consideration from the hypothetical “peak” performance of the machine, and at measuring sustainable memory bandwidth in MB/s. In doing so, its HLS defines four operations (copied from [273]):

Copy Measures transfer rates in the absence of arithmetic.

Scale Adds simple arithmetic operation.

Sum Adds third operand for multiple load/store ports.

Triad Allows chained/overlapped/fused multiply/add operations.

The four operations are considered as representative building blocks of long vector operations. For meaningful results, the benchmark works with data sets and code structures that cannot be stored completely in the available cache, but require memory traffic. The benchmark’s LLI is written in standard FORTRAN77 and a corresponding version in C. Continuously updated results are presented on the benchmark homepage [273]. The STREAM benchmark is, amongst others, part of the NERSC-8/Trinity Benchmark Suite [292] and the HPC Challenge Benchmark Suite [271, 273, 272]. Besides, there are some forks of the STREAM benchmark, like MAPS, a “benchmark probe used to measure the rate at which a single processor can sustain rates of loads and stores depending on the size of the problem and the access pattern” [84, p. 3].

Sweep3D (Kernel)

The benchmark kernels were extracted from time-independent, Cartesian-grid, single-group, “discrete ordinates” deterministic particle transport code taken from the DOE *Advanced Simulation and Computing* (ASCI) workload. In particular, the benchmarks reflects ASCI workloads at the *Los Alamos National Laboratory* (LANL) and has characteristics of the computations and communications which consume the vast majority of the cycles. The benchmark’s LLI is provided in FORTRAN77 using MPI for inter-processor communications and said to be sensitive to the latency of both, the memory and the network [32, 112, 193, 192, 31, 112].

Whetstone (Synthetic)

Based on the analysis of 949 real world applications, the Whetstone benchmark was designed in 1972 to measure the *Floating-point* performance of small machines with simple architectures. As “first program in the literature explicitly designed for benchmarking” [407, p. 66], Whetstone set industry standards of performance, particularly for minicomputers. Its HLS comprises several modules, each addressing another typical behavior of common applications, like procedure calls or integer operations. In particular, the modules and contained segments are defined such that "the distribution of Whetstone instructions for the synthetic benchmark matched the distribution observed in the program sample" [407, p. 66]. Common LLI are provided in C and Pascal, the first LLI was written in Algol 60. The benchmark produces speed ratings in *Whetstone instructions per second* [414, 106, 122, 407].

Appendix C

Research contributions

The extend of some artifacts of the presented process model recommend their detailing in the appendix, instead in a certain section or chapter. Hence, the following artifacts are further detailed, ordered according to their use in the thesis:

- Actors (p. 283)
  Referenced in Section 3.3.1.
- Use Cases (p. 289)
  Referenced in Section 3.3.1.
- Non-functional requirements (p. 306)
  Referenced in Section 3.3.1.
- Provenance information model (p. 312)
  Referenced in Section 3.3.1.
- Action flow of activity template T-A3 (p. 314)
  Referenced in Section 3.3.1.

Actors

Figure C.1 recapitulates the actor hierarchy Chapter 3 extracts from real-world scenarios for the *Requirements Specification* (RS). This section details the depicted actors bottom-up relative to the inheritance hierarchy described in Section 3.3.1. In particular, actors at the bottom of the hierarchy are immediately extracted from the SuperMUC and DRIHM scenario, and are described first. Afterwards, actors in higher levels of the inheritance hierarchy describe commonalities of several actors and hence, are more abstract. Detailing uses wide-spread actor templates (↗KB p. 264), and employs black circle flags ❶ for referencing specific aspects in the scenario descriptions in Section 3.2.

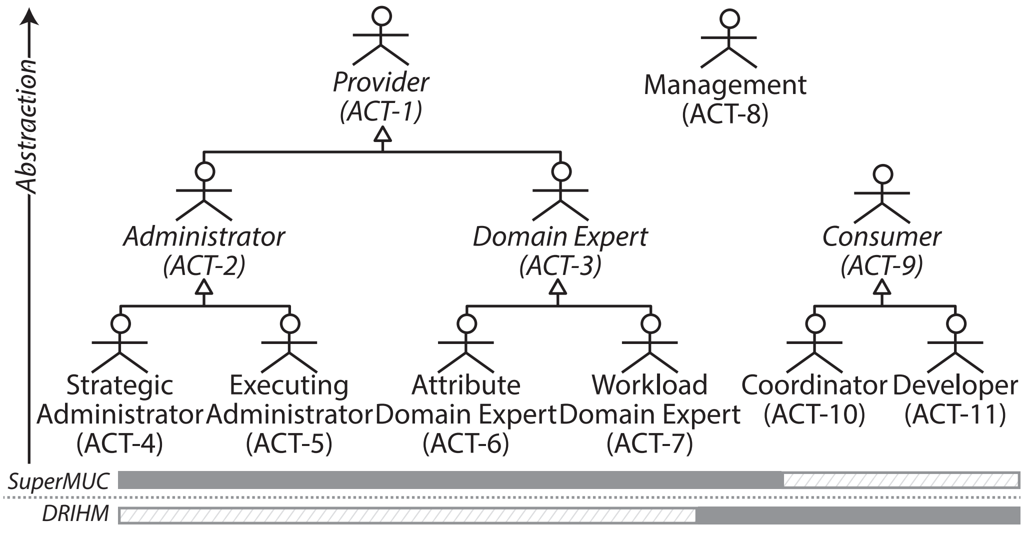


Figure C.1: Overview of actors extracted from examined real-world scenarios.

Actor	***Strategic Administrator***	ACT-4
Is responsible for architectural and technical long-term decisions regarding the IT infrastructure and evaluates technological trends and innovations in terms of suitability for the IT infrastructure. For small IT infrastructures, the actor might overlap with actor ACT-5.		
→ **SuperMUC** – Head of LRZ's *High performance systems* department and of the *HPC services* group 6. → **DRIHM** – EGI steering committee and heads of the organizations that provide resources to the *drihm.eu* VO 34.		

Table C.1: Actor "Strategic Administrator" (ACT-4).

Actor	***Executing Administrator***	ACT-5
Executes the concrete operation and maintenance of the IT infrastructure, e.g., the replacement of broken hardware, incorporation of (reviewed) modifications, maintenance of management databases, or implementation of required software. Compared to actor ACT-4, the actor doesn't make any (long-term) decisions about the IT infrastructure's architecture. In contrast, the actor acts according to externally given decisions and orders. For small IT infrastructures, the actor might overlap with actor ACT-4.		
→ **SuperMUC** – Members of LRZ's *HPC services*, *Distributed resources*, and *Application support* groups ❻. → **DRIHM** – Administrators at the resource providing sites (34).		

Table C.2: Actor "Executing Administrator" (ACT-5).

Actor	***Administrator***	ACT-2
Is responsible for and realizes all physical operation activities on the IT infrastructure and acts as counterpart of actor ACT-3. The actor is marked as abstract to force a refinement in planning and executing activities by actor ACT-4 and ACT-5, respectively. Depending on the IT infrastructure's scale, complexity, and organizational structure (cf. Section 2.2), the actor might be responsible for the entire IT infrastructure or only a sub set.		
→ Super class of actor ACT-4 and ACT-5.		

Table C.3: Actor "Administrator" (ACT-2).

Actor	***Attribute Domain Expert***	ACT-6

The actor is highly experienced in one or more attributes, e.g., performance or energy efficiency (cf. Section 2.4), especially for the IT infrastructure at hand. In particular, the actor profoundly knows about the modeling, measuring, and other influencing factors of a set of attributes. In some situations, the actor might overlap with actor ACT-7, for instance, when benchmarks are executed to generate load for measuring (cf. Section 2.3.3).

→ **SuperMUC** – Members of the *HPC services* division at LRZ work on energy efficiency and performance modeling [27, 174] 16.

→ **DRIHM** – Stuff members at the resource provider sites 34. They might be experienced about a broader range of common hardware types, compared to the *HPC services* group members at LRZ in the SuperMUC scenario (cf. above) that are very focused on SuperMUC to tackle its custom construction.

Table C.4: Actor “Attribute Domain Expert” (ACT-6).

Actor	***Workload Domain Expert***	ACT-7

The actor is highly experienced in the development and execution of workload (cf. Section 2.3), especially for the IT infrastructure at hand. Additionally, he is skilled in identifying and predicting the load that is about the be generated by executing workload on the IT infrastructure. The actor collaborates with actor ACT-6 for load generation.

→ **SuperMUC** – Members of the *Application support* group at LRZ 7 work on workload analysis and application tuning [283].

→ **DRIHM** – Members of the “Services for researchers” provided by EGI [136], especially the “Consulting and support” division.

Table C.5: Actor “Workload Domain Expert” (ACT-7).

Actor	***Domain Expert***	ACT-3
In contrast to actor ACT-2, the actor deals with all non physical aspects of IT infrastructure operations, covering attribute and workload modeling and prediction. To address the variety of issues and challenges in these fields, the actor is marked abstract to force a refinement in attribute and workload related aspects by actor ACT-6 and ACT-7, respectively.		
→ Super class of actor ACT-6 and ACT-7.		

Table C.6: Actor "Domain Expert" (ACT-3).

Actor	***Provider***	ACT-1
Subsumes actors being experienced in and responsible for IT infrastructure operations and maintenance in a wider sense. The actor does not make any strategic decisions or SLA negotiation, but works according to the decisions made by actor ACT-8. Besides, he ensures proper IT infrastructure provisioning to actor ACT-9, and provides consulting services to actor ACT-8. The actor is marked abstract to force refinement.		
→ Super class of actor ACT-2 and ACT-3.		

Table C.7: Actor "Provider" (ACT-1).

Actor	***Management***	ACT-2
Is responsible for the IT infrastructure in general, communicates with (potential) consumers (actor ACT-9), and negotiates SLAs. Actor ACT-2 initiates reasoning activities and interprets results to fulfill his responsibility for the IT infrastructure, to address the urgent need of considering external influencing factors like electricity prices or national law, and to respect the consumer demands. If necessary, he triggers modifications, actor ACT-1 implements.		
→ **SuperMUC** – LRZ's *Board of directors* ❽. → **DRIHM** – Management of each resource provisioning site.		

Table C.8: Actor "Management" (ACT-2).

Actor	***Coordinator***	ACT-10
In the scientific field (see focus on scientific applications in Section 2.3), contracts about IT infrastructure use are mostly made with a research project but a single person. Actor ACT-10 represents the project's head, usually called the *project coordinator*, and negotiates SLAs about attribute value ranges to ensure sufficient IT infrastructure capabilities for his project.		
→ **DRIHM** – The DRIHM project is coordinated by WP1 and in particular the CIMA foundation (cf. Section 3.2.2).		

Table C.9: Actor "Coordinator" (ACT-10).

Actor	***Developer***	ACT-11
Develops workload and especially real world applications (cf. Section 2.3.1) that run on the IT infrastructure. The actor is mostly a computer science applying entity, e.g., an HMR or biology scientist, but a computer scientist.		
→ **SuperMUC** – Users that execute the software on SuperMUC 15. → **DRIHM** – Developer of the executed models 38, e.g., the *Centre National de la Recherche Scientifique* (CNRS) develops Meso-NH [245] 39.		

Table C.10: Actor "Developer" (ACT-11).

Actor	***Consumer***	ACT-9
Uses the IT infrastructure to solve a particular (scientific) problem (cf. Section 2.3). Compared to actor ACT-1, the actor is mostly not a computer scientist, but member of an applying discipline, like HMR or biology. Hence, there is a differentiation between consuming and providing actors. This is of special relevance for actor ACT-7 and ACT-11. The actor is marked as abstract to enforce a refinement in workload related and contract related roles by actor ACT-11 and ACT-10, respectively.		
→ Super class of actor ACT-11 and ACT-10.		

Table C.11: Actor "Consumer" (ACT-9).

Functional requirements

Chapter 3 extracts 17 functional requirements from real-world scenarios for the RS. This section details the functional requirements, using wide-spread Use Case templates (↗KB p. 264), and employing black circle flags ❶ to reference specific aspects and details in the scenario descriptions in Section 3.2.

Use Case	***Initiate reasoning activity***	UC-1

(Potentially) high costs and efforts of (exhaustive) reasoning bans the execution of a reasoning activity without a cogent reason. Use Case UC-1 addresses this demand for a reasoning justification by generically describing situations that explicitly require a reasoning activity. For instance, changing customer needs, alternating varying regulations, or procurement. Even though the list of tangible situations is rather short, there is a dedicated Use Case to foster a generic and broadly applicable requirements list. In addition, the Use Case provides input to Use Case UC-2 that extracts and defines the objectives of a particular reasoning activity for Use Case sub systems *B* and *C*.

ACT-8	Is responsible for the IT infrastructure in general and applies a provider and a consumer perspective at the same time. Hence, the actor identifies situations that call for a reasoning activity.

→ **SuperMUC** – In the SuperMUC scenario, there are mainly three situations that initiate a reasoning activity: changing customer needs ㉓, changing external factors ㉗, and procurement ㉛.

→ **DRIHM** – Since the DRIHM scenario applies the consumer perspective, the rather generic Use Case is abstracted only from the provider perspective applying SuperMUC scenario. In contrast, the DRIHM scenario is important for the specializing Use Case UC-1.1.

Table C.12: Use Case “Initiate reasoning activity” (UC-1).

Use Case	***Negotiate SLA and attributes***	UC-1.1

A special situation that might initiate a reasoning activity is the negotiation of SLAs in general, and of attribute ranges and thresholds, in particular. The situation is described by a dedicated Use Case, because it is the most common situation and it involves all three actor groups (cf. Section 3.3.1), i.e., the provider perspective, the management, and the consumer perspective. The Use Case inherits from Use Case UC-1 and aims at 1) covering potential conflicting interests of all involved partners, 2) processing them, and 3) passing them to Use Case UC-2.

ACT-6	Actor ACT-10 is mostly no computer scientist and hence, might pose unrealistic or conflicting demands in terms of expected attribute ranges. To address this potential problem, actor ACT-6 participates in a consulting role and intervenes if necessary.
ACT-8	Inherited from parent Use Case UC-1. The actor is interested not only in winning actor ACT-10 over to use the IT infrastructure, but also in preserving the provider's aims and rules.
ACT-10	The project coordinator, representing the IT infrastructure consuming entity, is interested in negotiating attribute ranges that are beneficial from a consumer perspective, e.g., a high performance.

→ **SuperMUC** – SuperMUC exposes its capabilities and capacities to manifold organizations (1), like PRACE or the Gauss Centre for Supercomputing, for a diversity of applications (10). Each of them poses differing demands on the SuperMUC and formulates an individual "cocktail" of attribute constraints. PRACE, for instance, is interested in both (24) SuperMUC's performance and reliability, to provide high quality HPC services to its members. Besides, interest conflicts might arise, e.g., high performance versus power consumption (28).

→ **DRIHM** – The DRIHM IT infrastructure consists of several resources from multiple resource providers (34), e.g., a set of resources within EGI. Each resource inclusion in the DRIHM IT infrastructure precedes a negotiation with the providing organization, even within EGI (42).

Table C.13: Use Case "Negotiate SLA and attributes" (UC-1.1).

Use Case	***Define reasoning objectives***	UC-2

Use Case UC-2 covers the extraction, delimitation, and explicit definition of relevant reasoning objects and details according to the reasoning initiating input given by Use Case UC-1. In addition, the Use Case decomposes reasoning tasks and prepares their distribution or delegation to relevant actors in a way that ensures result compatibility for later result composition. The Use Case defines the reasoning objectives in order to foster an efficient and cost saving reasoning by reducing unnecessary attempts, and to ensure reproducibility.

ACT-8	Specifies the reasoning activity in terms of considered attributes, workload, and IT infrastructure components. Implicitly, ACT-8 does a first prioritization of reasoning aspects in the context of high-level management decision making. ACT-8 requests information from and delegates tasks to other actors.

→ **SuperMUC** – The LRZ employs several groups (6) that collectively maintain and operate SuperMUC. Each group and also each group member has its particular experience and knowledge, which have to be combined and integrated in the reasoning activity to achieve a profound result. This, in turn, requires task decomposition and synchronization.

→ **DRIHM** – The DRIHM IT infrastructure is built by several resource providers (34), each pursuing individual objectives, which calls for clearly defined reasoning objectives.

→ **Environment** – Reasoning activities tend to be complex and comprehensive, since they cover an IT infrastructure and several attributes. In addition, multiple actors are involved in terms of knowledge, experience and (manual) execution. Especially the "possibility factor" requires a clearly delimited reasoning objective, since people tend to build more complex models if their tools support it [334].

☐ A definition confines the reasoning activity and considered details.
☐ Reasoning sub tasks can be delegated to actors, e.g., IT infrastructure modeling or attribute model selection.

POST

Table C.14: Use Case "Define reasoning objectives" (UC-2).

Use Case	***Define attribute(s)***	UC-2.1

Guided by the reasoning objectives from Use Case UC-1, Use Case UC-2.1 selects relevant attributes and defines them, respectively. The definition is formulated in a way that 2) fosters a common understanding between all involved actors, that 2) aligns delegated reasoning tasks, that 3) ensures reproducibility, and that 4) eases tool preparation in Use Cases sub system *B*, e.g., supporting measuring instrument or model parameter selection.

ACT-6	Selects and defines the attribute(s) as requested by actor ACT-8 in Use Case UC-1. Since there might be multiple attributes, several ACT-6 actors can be involved, each focusing its personal attribute.

→ **SuperMUC** – Although there might be a common notion about the attributes in the focus of SuperMUC's operation, i.e., performance and energy efficiency, this notion might be too broad for a focused and effective reasoning activity. For instance, the performance instances FLOP/s and TTC 17 fundamentally differ in terms of meaning and required measurement tools. In addition, even within an attribute instance further clarification might be required, e.g., include or exclude queuing time 18.

→ **Environment** – An attribute tends to have several instances, , e.g., performance has *time-to-completion* and as *FLOP/s* (cf. Section 2.4.2), each exposing a variety of options and requiring differing assumptions (cf. Section 2.4). This is eminently precarious, since the backgrounds of the involves actors tend to differ (cf. description of actor ACT-9).

POST

- ☐ There is an attribute list reasoning has to cover.
- ☐ Each list entry contains an unambiguous name, a description about the attribute's objectives in the context of the reasoning activity, and the used attribute instance (cf. Section 2.4), consisting of calculation rules and scale (↗KB p. 267).

Table C.15: Use Case "Define attribute(s)" (UC-2.1).

Use Case	***Select workload***	UC-2.2

Guided by the reasoning objectives from Use Case UC-1, Use Case UC-2.2 selects workload that reflects the analyzed circumstances and relates to attributes selected in Use Case UC-2.1, e.g., reason about performance and energy consumption during CFD application execution.

ACT-7	Selects one or multiple workload elements as requested by actor ACT-8 in Use Case UC-1. Since there might be multiple workload classes suitable, several ACT-7 actors might be involved, each focusing its personal workload class.

→ **SuperMUC** – Over 150 differing applications employ the SuperMUC, each posing differing requirements on the IT infrastructure 11, e.g., due to the great diversity of employed programming languages 12.
→ **DRIHM** – Although the amount of applications in the DRIHM scenario is smaller than in the SuperMUC scenario, the applications pose extremely differing demands on the IT infrastructure and especially the attributes. In particular, the tools differ in terms of computation demands and generated load 37.
→ **Environment** – Most attributes depend on workload and in particular the generated IT infrastructure component load (cf. Section 2.3 and 2.4). In addition, the concrete implementation tends to have a sever impact, e.g., implementations of the same benchmark can vary depending on the employed programming language (cf. Section 2.3.2).

☐ There is a workload list reasoning has to cover and use. The list can consist of real world applications, benchmarks, or both. ☐ Each list entry contains a workload description and a selection justification for reproducibility.	***POST***

Table C.16: Use Case "Select workload" (UC-2.2).

Use Case	***Select IT infrastructure component(s)***	UC-2.3

Guided by the reasoning objectives from Use Case UC-1, Use Case UC-2.3 selects IT infrastructure parts and delimits component types that are affected by the considered attributes (cf. UC-2.1) and the executed workload (cf. UC-2.2). Depending on the reasoning objectives, differing IT infrastructure areas might be of interest in terms of (hardware) component types, like CPU or interconnect, and scale, i.e., the number of considered components within a component type, e.g., considering one or all CPU's of an HPC system.

ACT-2	Selects IT infrastructure components for reasoning as requested by actor ACT-8 in Use Case UC-1. In addition, actor ACT-2 prepares component selection for modeling in Use Case UC-3, e.g., in terms of provided meta information.
ACT-6	Consultatory supports actor ACT-2 in the IT infrastructure component selection. For instance, he highlights component types that might have a (strong) influence on the considered attributes.

→ **SuperMUC** – SuperMUC is built of several specialized areas, for instance, computing nodes and storage facilities. This specialization is reflected in focused groups ❻. Even within a specialized area, there is a separation, e.g., the HPC cluster file system GPFS and the backup storage ❹. Depending on the reasoning objectives, not all hardware types are of interest, e.g., reasoning about the power consumption and performance of SuperMUC's compute nodes renders coverage of third level backup storage unnecessary. There are also situations when not all components of the same type must be considered, e.g., in case workload employs only a single compute node ❸.

→ **Environment** – Section 2.2 emphasizes that contemporary IT infrastructures tend to be exhaustive and complex, especially HPC clusters, supercomputers, and Grids. But also small systems can be challenging, depending on the applied level of granularity.

☐ There is an IT infrastructure component list reasoning has to cover.
☐ Each list entry contains a component description and a selection justification for reproducibility.
☐ Optionally, there is an additional list containing a set of component types reasoning has to consider.

POST

Table C.17: Use Case "Select IT infrastructure component(s)" (UC-2.3).

Use Case	***Model IT infrastructure***	UC-3

Create a model of the IT infrastructure elements and details selected in Use Case UC-2. In particular, modeling in Use Case UC-3 is constraint to the given component type set, to the component number, and to the granularity level(s) in order to compile a model that sufficiently supports the reasoning activity at the one hand, but that is also preferably focused and small on the other hand.

ACT-2	The actor's responsibility for (physically) operating and maintaining the considered IT infrastructure results (theoretically) in a high level of experience and knowledge. Together with the tool set and information, this recommends actor ACT-2 for creating the IT infrastructure model.

PRE			*POST*
	☐ There is an IT infrastructure component list modeling has to cover. ☐ The list describes component types, granularity levels, and scale.	☐ There is a model describing the IT infrastructure according to the details defined in Use Case UC-2.	

Table C.18: Use Case "Model IT infrastructure" (UC-3).

Use Case	***Model part of IT infrastructure***	UC-3.1

Especially large-scaled and/or complex IT infrastructures are built of several specialized areas, each exhibiting differing challenges and characteristics that are relevant for modeling the IT infrastructure. Use Case UC-3.1 describes the situation of multiple entities being responsible for the same IT infrastructure and consequently, multiple entities are contributing to the IT infrastructure model.

ACT-2	The actor who is responsible for (physically) operating and maintaining the considered IT infrastructure part.

→ **SuperMUC** – As explained in Use Case UC-2.3, SuperMUC is built of several specialized elements. Each element might be in a specific group's area of accountability, e.g., the *HPC Services* group covers SuperMUC itself, the *Data and Storage Systems* group covers storage elements ❻. Thus, each group contributes to the overall IT infrastructure model of SuperMUC, since each group is experienced in its particular area.

PRE			*POST*
	☐ There are clearly distinguished entities responsible for each part of the considered IT infrastructure. ☐ Reasoning objectives require coverage of at least two of the IT infrastructure parts. ☐ There is no dedicated entity capable of modeling all required IT infrastructure parts.	☐ There is a model describing the requested IT infrastructure part according to the prerequisites stated in Use Case UC-2.	

Table C.19: Use Case "Model part of IT infrastructure" (UC-3.1).

Use Case	***Import IT infrastructure information***	UC-3.2

Three reasons recommend the use of existing information to model the IT infrastructure: 1) existing management tools or databases, like a *Configuration Management Database* (CMDB), often already contain suitable information about the IT infrastructure. After an optional processing step, this information could be used as modeling and hence, reduce modeling efforts; 2) constructing the IT infrastructure model on existing information supports actuality, because the information updated in daily-use management tools can be absorbed. This is of special importance for IT infrastructures exposing high dynamics; 3) using existing information avoids duplicates and integrity flaws. The (potential) high dynamics of IT infrastructures requires an interface to existing management and description approaches and tools to ease keeping the model up to date. An interface would also be beneficial to interact with special purpose approaches, like discrete event simulators, and hence, gaining acceptance.

ACT-2 The actor who is responsible for (physically) operating and maintaining the considered IT infrastructure.

→ **DRIHM** – Uses the BDII information service to store management information about the IT infrastructure. BDII provides a query and export interface 36.

→ **Environment** – Especially for Grids that rely on common and open standards, there are information models describing resources, like the GLUE scheme (cf. Section 2.2.3).

PRE	*POST*
◻ There is a tool storing suitable information about the IT infrastructure in an appropriate format, like CIM or GLUE. ◻ In case the format is not directly applicable, a mapping is possible. ◻ The tool provides an export interface or any other export capability.	◻ The IT infrastructure model describes the considered IT infrastructure using the data that is already maintained in a third party tool. ◻ Redundancy or integrity flaws are avoided.

Table C.20: Use Case “Import IT infrastructure information” (UC-3.2).

Use Case	***Update IT infrastructure model***	UC-3.3

The dynamics of IT infrastructures require updating the IT infrastructure model entirely or partly. Especially large-scaled and/or complex IT infrastructures are built of several specialized areas, each of them exposing differing update cycles. Consequently, updating the IT infrastructure model is on-demand. The updating process also employs the import functionality described in Use Case UC-3.2.

ACT-2	The actor who is responsible for (physically) operating and maintaining the considered IT infrastructure.

→ **DRIHM** – Each of the different resource providers that contribute to the DRIHM infrastructure apply individual maintenance cycles and dynamically add or remove resources 35.

PRE			*POST*
	□ An IT infrastructure model exists. □ The considered IT infrastructure has changed physically and in a way that has to be reflected by the IT infrastructure model.	□ The IT infrastructure model is up-to-date, which means it described the IT infrastructure as it is.	

Table C.21: Use Case “Update IT infrastructure model” (UC-3.3).

Use Case	***Select model for attribute(s) and component(s)***	UC-4

Use Case UC-4 covers model selection according to the demands posed by Use Cases UC-1 and UC-2. For instance, selecting a model to describe the power consumption of processors or the reliability of storage elements. These models are used for reasoning in sub system *C*. Since multiple attributes can be covered by a single reasoning activity, Use Case UC-4 is likely to be executed several times, namely for each considered attribute and/or for the same attribute but differing component types.

ACT-6 The expert for the particular attribute domain.

→ **SuperMUC** – The variety of IT infrastructure components ❷ and attribute instances (16) calls for a clear model selection process.

→ **Environment** – The power set of IT infrastructure aspects and component types (cf. Section 2.2), attributes, and attribute instances (cf. Section 2.4) is huge. For a part of combinations, there are (mature) models, describing a particular situation, e.g., modeling communication performance for message-passing based node communication [13], or estimating the power consumption of an Intel PXA255 processor [104] (cf. Section 7.4). Addressing this variety of existing models to describe relevant attributes prerequisites a selection before reasoning can be executed.

PRE	☐ An attribute and a set of component types were selected.	☐ A model is selected that describes the considered attribute for the given component type(s) on the requested level of detail.	*POST*

Table C.22: Use Case "Select model for attribute(s) and component(s)" (UC-4).

Use Case	*Create model proxy*	UC-4.1

Despite the plurality of existing models describing a variety of attributes and aspects, there might be situations that there is no suitable model integration candidate. This might be the case if 1) no model describes the particular attribute for the given component type(s) at the requested level of detail, 2) the model does not describe the attribute sufficiently, e.g., inaccurate, or 3) other constraints might be violated, like licensing issues. To address this situation, Use Case UC-4.1 describes the creation of a model proxy, which is defined according to the specific reasoning demands, ranging from a discrete number to a function compiled by any suitable method, like measurement-based regression.

ACT-6	As expert for the considered attribute, actor ACT-6 is responsible for the model proxy creation process.
ACT-7	Owns a supporting role in the model proxy creation process, especially when the execution of workload is required during a measurement and function derivation activity.

→ **SuperMUC** – Despite the important role of energy efficiency for SuperMUC, there are no (time-tested) models available to describe this attribute 20. To enable coverage of energy efficiency in a bigger reasoning activity requires the creation of a model proxy.

→ **Environment** – Model creation is mostly faced with the challenging trade-off between accuracy and generality. The more accurate a model is, the less is it applicable to a variety of IT infrastructure components. This is of special importance for supercomputers that are built of highly specialized components (cf. Section 2.2.2). Hence, depending on the IT infrastructure and particularly its level of specialization, there might be no models available describing the attribute(s) of interest.

PRE	☐ An attribute/component combination is specified. ☐ There is no suitable model describing the attribute for the given component type(s).	☐ There is an alternative way to provide a value for the attribute/component combination according to the specific reasoning demands.	*POST*

Table C.23: Use Case “Create model proxy” (UC-4.1).

Use Case	***Create load profile***	UC-4.2

Use Case UC-4.2 describes the creation of a load profile (cf. Section 2.3.3), which is used as input by the reasoning activity in sub system *C*. A load profile is especially required when the workload selected in Use Case UC-2.2 exposes a comparatively long run time.

ACT-7	Is the expert for the considered workload. Since several workloads can be selected in Use Case UC-2.2, for each workload an individual actor might be involved.
ACT-2	Is optionally involved in case actor ACT-7 requires consultative support, e.g., regarding specific workload (profile) characteristics.

→ **SuperMUC** – Applications executed on SuperMUC tend to expose a very long run time of up to 80 hours 13. Whenever the reasoning activity in sub system *C* should cover not only a snapshot, but a period of time, a load profile is required that contains load information of the considered IT infrastructure components.

- ☐ A load profile describes the load value (cf. Section 2.3.3) for each IT infrastructure component of interest.
- ☐ The load profile applies the time granularity required by the reasoning objectives defined in Use Case UC-2.

POST

Table C.24: Use Case "Create load profile" (UC-4.2).

Use Case	***Execute reasoning***	UC-5

Use Case UC-5 acts as container for the wide field of reasoning situations. In particular, it summarizes reasoning execution according to the objectives defined in sub system A, and using the reasoning tools created in sub system B. Due to the variety of possible reasoning approaches and their respective suitability, the Use Case includes the specialized Use Cases UC-5.1, UC-5.2, and UC-5.3.

ACT-8	In its responsibility of making management decisions regarding procurement, modifications, and day-by-day operations, ACT-8 is interested in employing the reasoning tools prepared in sub system B to support the decision making process. Since the actor is the only one having insights in the (long-term) IT infrastructure strategy, he is solely responsible for reasoning conduction.
ACT-6	Consultatory supports actor ACT-8 in the reasoning conduction by providing differing information according to the specific Use Case. Since several attributes might be involved, multiple ACT-6 actors might be involved according to their area of expertise, respectively.
ACT-7	The influence of workload and load on nearly every attribute (cf. Use Case UC-2.2) requires the involvement of actor ACT-7, who particularly provides load profiles (cf. Use Case UC-4.1) as input for the reasoning tools.

→ **SuperMUC** – Changes in the surrounding, use, and operations of an HPC system like SuperMUC cause a variety of situations that require decision making supported by reasoning about quantitative IT infrastructure attributes 22.

Table C.25: Use Case "Execute reasoning" (UC-5).

Use Case	***Execute What-if analysis based reasoning***	UC-5.1

Use Case UC-5.1 extends the abstract Use Case UC-5 by providing What-if analysis functionality, whose basic concept is the variation of inputs, i.e., the number of operators, and the determination of potential effects [334]. In case a cross product of all parameters is used, it is also called a *parameter sweep*. What if functionality is especially required to compare and validate a set of options, e.g., comparing modification possibilities or vendor offerings.

ACT-8	The actor aims at investigating, assessing, and comparing varying inputs with regards to their outcome. Since only actor ACT-8 knows (theoretically) the goals and targeted outcomes, especially in the IT infrastructure strategic context, the actor is solely responsible for an What-if analysis based reasoning.
ACT-6	Consultatory supports actor ACT-8 in defining reasonable input value ranges.

→ **SuperMUC** – The long planning horizon of supercomputers like SuperMUC and the difficulties to apply fundamental modifications 9, e.g., to revise design mistakes, call for a thorough investigation of different configurations and designs reflected in varying reasoning inputs 30. In other words, assess what the result would be if a particular input combination is used, e.g., stated in a vendor's procurement offer.

Table C.26: Use Case "Execute What-if analysis based reasoning" (UC-5.1).

Use Case	***Execute optimization based reasoning***	UC-5.2

Use Case UC-5.2 extends the abstract Use Case UC-5 by providing optimization functionality. Optimization in a mathematical sense aims at finding a (local) maximum, i.e., the "best" element with regard to some criteria from some set of available alternatives, for a set of pre-defined constraints [102]. Reasoning applies this principle if several aspects are (externally) dictated, e.g., by SLAs, regulations, or market prices. These aspects can also formulate a set of constraints, e.g., SLAs could call for high performance, regulations could dictate a minimum reliability value, and market prices could pressure energy efficiency.

ACT-8	The actor aims at finding an optimum for a set of externally given constraints. Before executing the reasoning, the constraints might be prioritized. Since only actor ACT-8 knows (theoretically) the entire "big picture" of the IT infrastructure, especially in a strategic sense, the actor is solely responsible for an optimization based reasoning.
ACT-6	Consultatory supports actor ACT-8 in defining the optimization bounds and constraints.

→ **SuperMUC** – SuperMUC's operation and management, i.e., the board of directors 8, is faced with a variety of (potentially) conflicting attributes, e.g., low power consumption vs. high performance. This situation calls for reasoning using mathematical optimization approaches, since the (externally) given attribute values or ranges are the constraints a (local) maximum is searched for 29.

→ **Environment** – As motivated in Section 2.4.1, attribute improvement attempts tend to clash, e.g., improving single-thread performance by employing speculative path execution imperatively causes a higher power consumption, because power spent on following a speculative execution path is lost whenever the path is not taken. Consequently, a (local) optimum must be found, respecting sharp and soft constraints.
.

Table C.27: Use Case "Execute optimization based reasoning" (UC-5.2).

Use Case	***Execute descriptive statistics based reasoning***	UC-5.3

Use Case UC-5.3 extends the abstract Use Case UC-5 by providing descriptive statistics functionality, i.e., methods for organizing and summarizing data sets [287] (↗KB p. 268). Use Case UC-5.3 is mainly interested in the reasoning outcome and neglects the used input. In other words, the Use Case investigates the data compiled by the reasoning and is searching for correlations and commonalities.

ACT-8	The actor aims at identifying insights extracted from reasoning results. Since only the ACT-8 actor knows (theoretically) the entire "big picture" of the IT infrastructure, especially in a strategic sense, the actor is solely responsible for an optimization based reasoning.

- → **SuperMUC** – The varying demands of SuperMUC's users result in differing SLAs. Especially supporting application development and targeting an "optimal" adjustment of executed applications and the IT infrastructure render descriptive statistics a suitable tool 25.

Table C.28: Use Case "Execute descriptive statistics based reasoning" (UC-5.3).

Use Case	***Trigger activity***	UC-6

Use Case CU-6 covers the potential need of executing a certain activity, depending in the reasoning result compiled in Use Case UC-5. For instance, a modification is initiated if reasoning unfolds its suitability.

ACT-8	In the context of the IT infrastructure strategy, the ACT-8 actor decides whether a reasoning result calls for an activity. If so, he delegates the task to the appropriate entity.
ACT-2	In case a modification is required, the ACT-2 actor is responsible for its implementation.

- → **SuperMUC** – LRZ's board of directors 8 requests the responsible group 6 to perform activities according to reasoning results.
- → **DRIHM** – Reasoning about the DRIHM IT Infrastructure might unfold the obligation of one or multiple resource providers to modify the offered resources 42, e.g., to comply (again) to negotiated SLAs.

Table C.29: Use Case "Trigger activity" (UC-6).

Non-functional requirements

Chapter 3 extracts eight non-functional requirements from real-world scenarios for the RS, which are detailed in this section. Due to the lack of a wide-spread template, detailing uses the alike suitable Use Case template (↗KB p. 264). Besides, it employs black circle flags ❶ to reference specific aspects and details in the scenario descriptions in Section 3.2. Non-functional requirements NFR-1 to NFR-6 were abstracted directly from scenario descriptions, NFR-7 to NFR-8 cover more general aspects, e.g., model efficiency. Thus, only the former must provide abstraction sources.

Non-functional requirement	***Individual component type sets***	NFR-1

Reasoning about quantitative IT infrastructure attributes is required to support individual IT infrastructure component type sets instead of (pre-defined) specific ones, e.g., only CPUs or only memory [30]. Requirement NFR-1 is fulfilled, if every reasoning activity is capable of handling an arbitrary component type set equally, e.g., IT infrastructure modeling in Use Case UC-3 or, model proxy creation in Use Case UC-4.1.

→ **SuperMUC** – Reasoning about both, performance and energy efficiency, is required to cover all component types, as both are composed of and influenced by communication, interconnect, and I/O aspects ⓳㉑. Especially communication components mustn't be forgotten ❺. Furthermore, SuperMUC's complexity, reflected by several responsible groups ❻, hardens the extraction of a single component's contribution ㉜. Finally, SLAs between LRZ and consumers consider attributes as black boxes and do not consider their compilation ㉓.

→ **Environment** – Contemporary IT infrastructures consist of a variety of component types, which closely collaborate to provide the IT infrastructure's capabilities (cf. Section 2.2). Especially the efficient and effective use of supercomputers (cf. Section 2.2.2) rely on a balanced interplay between *all* component types, which in turn requires reasoning to cover all of them. The stake of a variety or even all IT infrastructure components to an attribute was evidenced empirically several times and is incorporated in several existing models (cf. Section 2.4.2).

Table C.30: Non-functional requirement "Individual component type sets" (NFR-1).

Non-functional requirement	***Individual attribute sets***	NFR-2

In contrast to the numerable amount of attribute concepts, like performance or reliability, there is a plurality of possible instances for each attribute, like the performance instances FLOP/s and TTC (cf. Section 2.4). This results in a big set of attributes to potentially reason about. The set is further extended by upcoming attributes, e.g., in the context approaching Exascale systems (cf. Section 2.2.2). Consequently, there is a theoretically infinite set of attributes to reason about. Providing a long-term solution requires the support of an individual attribute set. This flexibility is further emphasized by the manifold objectives of IT infrastructures, e.g., providing high performance or high dependability, and the absence of a consensus about attributes instances to choose for a particular attribute, e.g., energy efficiency [363].

→ **SuperMUC** – LRZ focuses for SuperMUC on the attributes performance and energy efficiency, employing different instances 16.

→ **DRIHM** – The requested support of individual sets covers both, the attribute concepts and instances. DRIHM uses the TTSD 41 performance instance, which can not be applied on SuperMUC that relies on the FLOP/s 17 performance instance. In addition, reasoning in the DRIHM scenario is interested in other attributes 40 than reasoning in the SuperMUC scenario.

→ **Environment** – There are dedicated IT infrastructure types focusing on specific attributes, like an HPC cluster focusing on performance (cf. Section 2.2.2). In addition, there are differences within a single attribute (cf. Section 2.4).

Table C.31: Non-functional requirement "Individual attribute sets" (NFR-2).

Non-functional requirement	***Multiple granularity levels***	NFR-3

The variety of reasoning objectives, which requires support of individual attribute sets (cf. NFR-2), also calls for support of multiple simultaneous granularity levels during reasoning, e.g., considering a compute node very detailed, while other components remain at an abstract level. Increasing reasoning granularity to the desired level globally would quickly lead to an unmanageable complex model (cf. NFR-7) due to IT infrastructure scale and diversity. Hence, multiple levels of granularity within the same model are mandatory to facilitate a good trade-off between accuracy, complexity, and time to solution [337].

→ **SuperMUC** – SuperMUC is built of several specialized parts ❷, each operated and maintained by a dedicated group ❻. In this context, achieving a sufficient component coverage (cf. NFR-1) and a simple model (cf. NFR-7) mandatory prerequisites the support of multiple granularity levels.

→ **DRIHM** – Reasoning about DRIHM's European-wide distributed IT infrastructure might require covering all elements, or executing reasoning for only a single resource ㊸.

→ **Environment** – The diversity of contemporary IT infrastructures compiles a containment hierarchy of IT infrastructure types and components (cf. Figure 2.2 in Section 2.2 on page 23). In addition, contemporary HPC systems and especially supercomputers are highly scaled, what bans a global level of detail (cf. Section 2.2.2). Finally, to address the highly relative notion of IT infrastructures, as detailed in Section 2.1.1, flexible modeling and reasoning is required.

Table C.32: Non-functional requirement "Multiple granularity levels" (NFR-3).

Non-functional requirement	***Workload consideration***	NFR-4

The executed workload and especially the caused load have mostly a strong impact on IT infrastructure attributes. Hence, reasoning is required to support workload consideration [45]. Non-functional requirement NFR-4 is fulfilled, if modeling in Use Case sub system B, and reasoning execution in Use Case sub system C is able to address workload aspects. For instance, when using models selected in Use Case UC-4, or when executing reasoning Use Case UC-5. Besides, there mustn't be constraints about the support of certain workload aspects. In contrast, workload consideration must be highly flexible.

→ **Environment** – The influence of workload and load on IT infrastructure attributes is under investigation since decades. It was empirically evidenced, like the correlation between load and TTC, and it is incorporated in existing models, like the *Telecommunications Equipment Energy Efficiency Rating* (TEEER) model introduced in 2011 by Verizon [383] or the *Modeling Assertions* (MA) framework for symbolic performance model development [15, 45] (cf. Section 2.4). Flexibility is required by the variety of workload types, e.g., scientific workflows consist of manifold task types (cf. Section 2.3.1).

Table C.33: Non-functional requirement "Workload consideration" (NFR-4).

Non-functional requirement	***Job cancellation***	NFR-5

Non-functional requirement NFR-5 extends NFR-4 as it calls for job cancellation consideration, i.e., the ability to cover the cancellation of workload execution selected in Use Case UC-2.2. It is fulfilled, if in any situation, the (theoretical) workload execution can be stopped partially or completely for an arbitrary IT infrastructure component set during reasoning.

→ **SuperMUC** – About 26% of jobs executed on SuperMUC are canceled before a results is available 14.
→ **Environment** – Due to manifold reasons, workload execution might be canceled, e.g., design errors in workflows might cause a rollback during execution (cf. Section 2.3.1). This is also reflected by job statistics, demonstrating high job cancellation rates that range from 7% up to 23% [253, 97], depending on the considered user community and system. The support of job cancellation consideration is underpinned by the potential strong affect on the execution behavior and the TTC of other applications running on the same resource(s) simultaneously [253].

Table C.34: Non-functional requirement "Job cancellation" (NFR-5).

Non-functional requirement	***Development over time***	NFR-6

Especially scientific applications (cf. Section 2.3) expose a long execution time, ranging from a couple of minutes up to days. Addressing the multitude of situations within this time frame requires the consideration of development over time. This is fulfilled, if for both, attributes selected in Use Case UC-2.1, and workload selected in Use Case UC-2.2, the development over time can be analyzed using an arbitrary time interval. In particular, the former requires storing attribute values for each time step, the latter implies the ability to consume load profiles (cf. Section 2.3.3).

→ **SuperMUC** – Workload executed on the SuperMUC exposes execution times ranging from seconds up to 80 hours 13. In average, a job execution takes 5.51 hours. In addition, the electricity price varies between night and day 26.

Table C.35: Non-functional requirement "Development over time" (NFR-6).

Non-functional requirement	***Simplicity***	NFR-7

Non-functional requirement NFR-7 calls for model simplicity that can be divided in the model's *transparency*, i.e., the perceived comprehensibility, and the model's *constructive simplicity*, i.e., the characteristics of the model itself [403]. NFR-7 is fulfilled, if all involved Use Cases, especially in sub system B, and non-functional requirements are implemented in a way that pursues model simplicity and complies to "Occam's razor", a principle devised by William of Ockham, stating that among competing hypotheses, the one with the fewest assumptions should be selected. Translated to modeling, this means that it should "begin with simple models and few parameters, then add complexity only as needed to explain [...]" [84, Sec. 2]. For instance, chosen granularity levels in Use Case UC-2.3, or chosen time frames in non-functional requirement NFR-6 [419] should lead to a simple model. Generally speaking, non-functional requirement NFR-3 addresses the (classic) trade-off between accuracy and effort.

→ **Environment** – Modeling mainly aims at compiling a *valid* model, which is a model that "is sufficiently accurate for the purpose at hand" [334, p. 67]. Since a model's size or bulkiness can outweigh its benefits [334], even if it is perfectly valid, the model should be preferably simple. Besides a reduced run time, further advantages are a fast development, easy result interpretation [96], and reduced error proneness [229].

Table C.36: Non-functional requirement "Simplicity" (NFR-7).

Non-functional requirement	***Efficient use***	NFR-8

A cost-saving, easy, and fast reasoning execution requires, amongst others, efficient modeling, in particular for complex and/or highly scaled IT infrastructures. This can be achieved by fostering reusability. Especially in homogeneous IT infrastructures, component type definitions and models as well as attribute notions and realizations should be reusable. For instance, using the same IT infrastructure model but differing workloads for diverse reasoning activities.

Table C.37: Non-functional requirement "Efficient use" (NFR-8).

Provenance information model overview

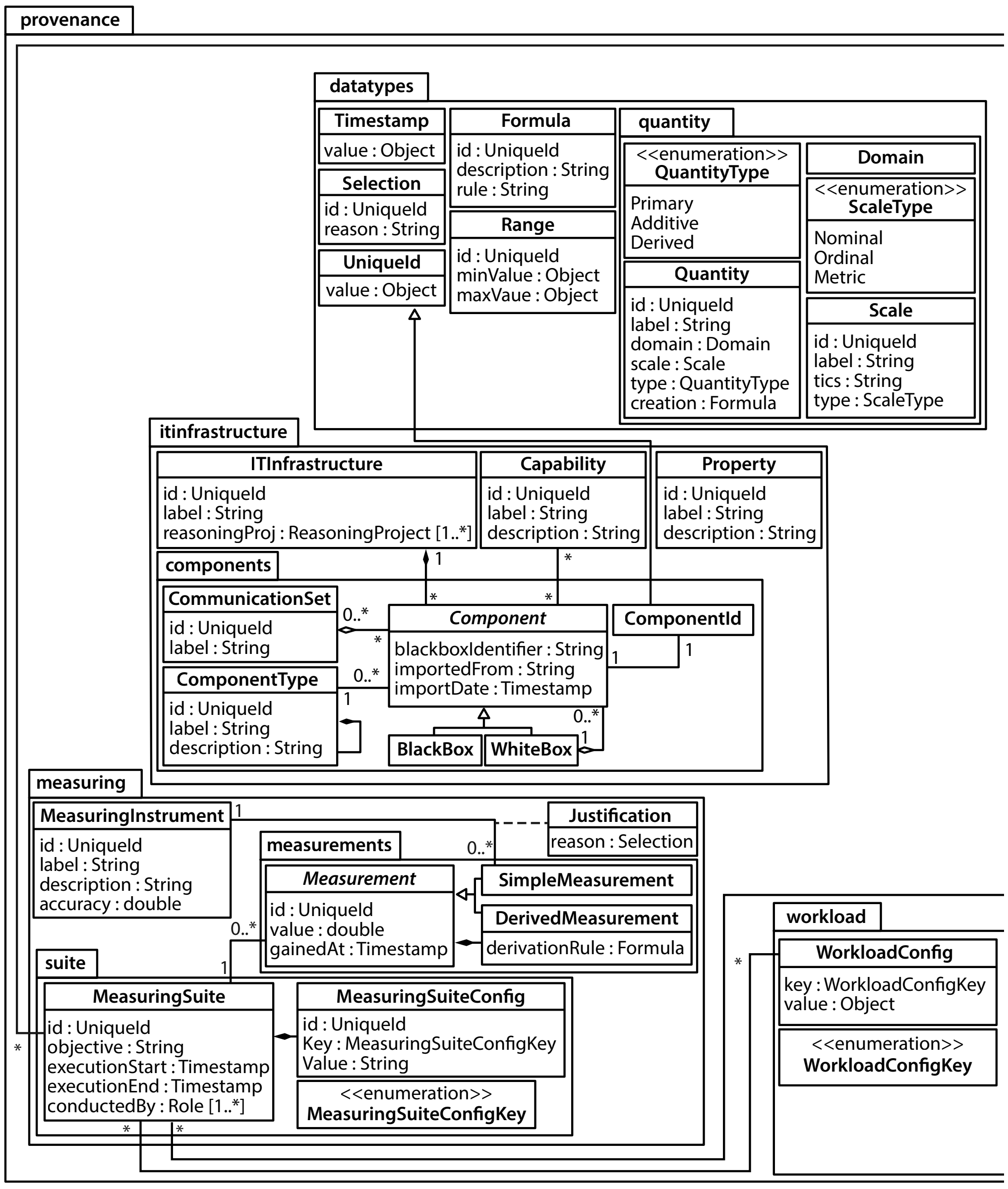


Figure C.2: The provenance information model provided by the presented process model – Part 1/2.

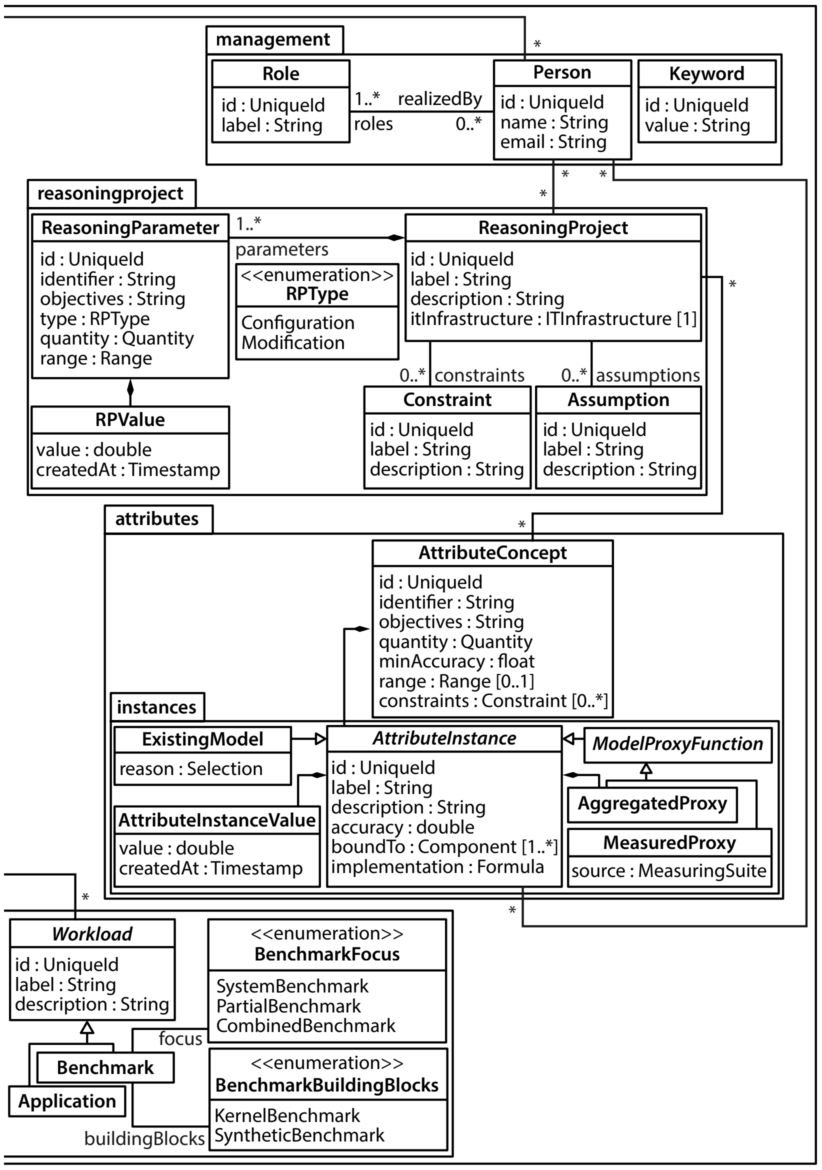
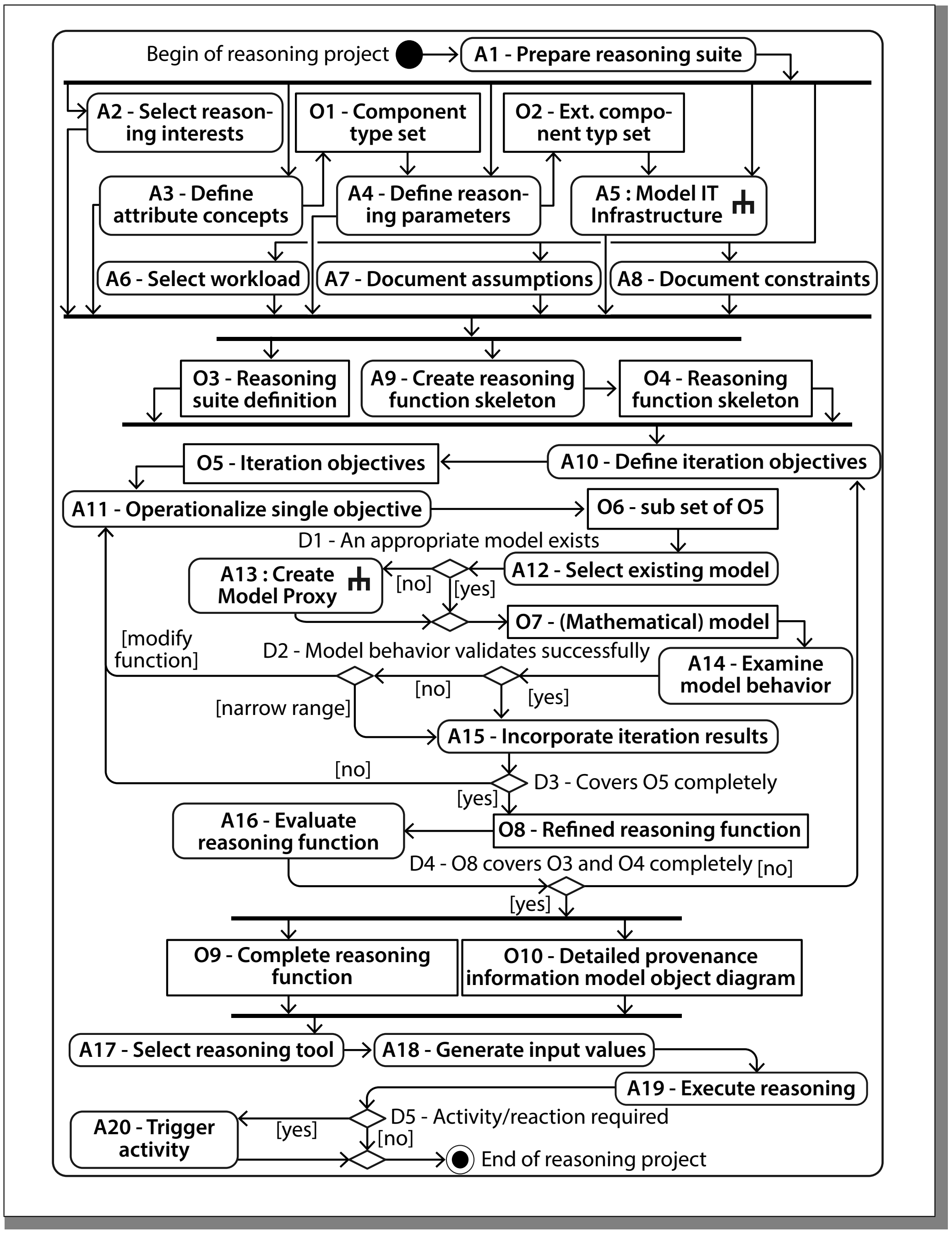


Figure C.3: The provenance information model provided by the presented process model – Part 2/2.

Action flow of activity template T-A3


Begin of reasoning project
A1 - Prepare reasoning suite
A2 - Select reasoning interests
O1 - Component type set
O2 - Ext. component typ set
A3 - Define attribute concepts
A4 - Define reasoning parameters
A5 : Model IT Infrastructure
A6 - Select workload
A7 - Document assumptions
A8 - Document constraints
O3 - Reasoning suite definition
A9 - Create reasoning function skeleton
O4 - Reasoning function skeleton
O5 - Iteration objectives
A10 - Define iteration objectives
A11 - Operationalize single objective
O6 - sub set of O5
D1 - An appropriate model exists
A13 : Create Model Proxy
[no]
[yes]
A12 - Select existing model
O7 - (Mathematical) model
[modify function]
D2 - Model behavior validates successfully
A14 - Examine model behavior
[no]
[yes]
[narrow range]
A15 - Incorporate iteration results
[no]
D3 - Covers O5 completely
[yes]
A16 - Evaluate reasoning function
O8 - Refined reasoning function
D4 - O8 covers O3 and O4 completely
[no]
[yes]
O9 - Complete reasoning function
O10 - Detailed provenance information model object diagram
A17 - Select reasoning tool
A18 - Generate input values
A19 - Execute reasoning
A20 - Trigger activity
[yes]
D5 - Activity/reaction required
[no]
End of reasoning project

Template element C.1: Action flow of T-A3 - Reasoning methodology.

List of Figures

List of Tables

List of Template elements

Bibliography

[1] C. Straube, W. Hommel, and D. Kranzlmüller. “Design Criteria and Design Concepts for an Integrated Management Platform of IT Infrastructure Metrics”. In: *Journal On Advances in Systems and Measurements* 7.1&2 (July 2014), pp. 150–167. URL: `http://www.thinkmind.org/download.php?articleid=sysmea_v7_n12_2014_14`.

[2] A. Galizia, D. D’Agostino, A. Quarati, G. Zereik, L. Roverelli, E. Danovaro, A. Clematis, E. Fiori, F. Delogu, A. Parodi, C. Straube, N. Felde, M. Schiffers, D. Kranzlmüller, Q. Harpham, B. Jagers, L. Garrote, V. Dimitrijevic, L. Dekic, O. Caumont, and E. Richard. “Towards an Interoperable and Distributed e-Infrastructure for Hydro-Meteorology: the DRIHM Project”. In: *Proceedings of the 7th International Congress on Environmental Modelling and Software (iEMSs’14)*. June 2014.

[3] A. Parodi, N. Rebora, E. Fiori, F. Delogu, F. Pintus, D. Kranzlmüller, M. Schiffers, N. Felde, C. Straube, A. Clematis, D. D’Agostino, A. Galizia, A. Quarati, E. Danovaro, O. Caumont, O. Nuissier, V. Ducrocq, É. Richard, L. Garrote, M. C. Llasat, Q. Harpham, H. R. A. Jagers, A. Tafferner, C. Forster, V. Dimitrijevic, L. Dekic, and R. Hooper. “The DRIHM Project: Building on Cutting-Edge Information and Communication Technology to Advance Hydro-Meteorological Research”. In: *Proceedings of the 7th HyMeX Workshop*. Oct. 2013.

[4] C. Straube and D. Kranzlmüller. “A Meta Model for Predictive Analysis of Modifications on HPDC Infrastructures”. In: *Proceedings of the 11th International Conference on Modeling, Simulation and Visualization Methods (MSV’14)*. July 2014.

[5] C. Straube and D. Kranzlmüller. "An Approach for System Workload Calculation". In: *Proceedings of the 12th International Conference on Parallel and Distributed Computing and Networks (PDCN'14)*. IASTED, Feb. 2014. DOI: 10.2316/P.2014.811-025.

[6] C. Straube and D. Kranzlmüller. "Model-Driven Resilience Assessment of Modifications to HPC Infrastructures". In: *Euro-Par 2013: Parallel Processing Workshops*. Vol. 8374. Springer, 2014, pp. 707–716. DOI: 10.1007/978-3-642-54420-0_69.

[7] C. Straube and D. Kranzlmüller. "An IT-Infrastructure Capability Model". In: *Proceedings of the 10th ACM Conference on Computing Frontiers (CF'13)*. ACM, May 2013. ISBN: 978-1-4503-2053-5. DOI: 10.1145/2482767.2482781.

[8] C. Straube, A. Bode, A. Hoisie, D. Kranzlmüller, and W. Nagel. "Dagstuhl Manifesto – Co-Design of Systems and Applications for Exascale". In: *Informatik Spektrum* 35.6 (Dec. 2012), pp. 464–467. DOI: 10.1007/s00287-012-0660-1.

[9] C. Straube, W. Hommel, and D. Kranzlmüller. "A Platform for the Integrated Management of IT Infrastructure Metrics (Best Paper Award)". In: *Proceedings of the 2nd International Conference on Advanced Communications and Computation (INFOCOMP'12)*. IARIA, Oct. 2012, pp. 125–129. ISBN: 978-1-61208-226-4.

[10] C. Straube, M. Schiffers, and D. Kranzlmüller. "Determining the Availability of Grid Resources using Active Probing". In: *Proceedings of the 11th International IEEE Symposium on Parallel and Distributed Computing (ISPDC'12)*. IEEE Computer Society, June 2012, pp. 95–102. DOI: 10.1109/ISPDC.2012.21.

[11] C. Straube and A. Schroeder. "Architectural Constraints for Pervasive Adaptive Applications". In: *Proceedings of the 3rd International DisCoTec Workshop on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS'10)*. EASST, June 2010, pp. 1–12. URL: http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/398.

[12] *A Guide to the Project Management Body of Knowledge: PMBOK Guide*. Vol. 5. Project Management Institute, 2013. ISBN: 978-1-93558-967-9.

[13] G. Abandah and E. Davidson. "Modeling the Communication Performance of the IBM SP2". In: *Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)*. 1996, pp. 249–257.

[14] M. Aguilera, J. Mogul, J. Wiener, P. Reynolds, and A. Muthitacharoen. “Performance Debugging for Distributed Systems of Black Boxes”. In: *ACM SIGOPS Operating Systems Review (SOSP’03)* 37.5 (2003), pp. 74–89.

[15] S. Alam and J. Vetter. “A Framework to Develop Symbolic Performance Models of Parallel Applications”. In: *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS’06)*. IEEE Computer Society, 2006.

[16] J. Alger. “On Assurance, Measures, and Metrics: Definitions and Approaches”. In: *Proceedings of the Workshop on Information-Security-System Rating and Ranking (WISSSR)*. 2002.

[17] A. Alkindi, D. Kerbyson, and G. Nudd. “Dynamic Instrumentation and Performance Prediction of Application Execution”. In: *Proceedings of the High-Performance Computing and Networking*. Ed. by B. Hertzberger, A. Hoekstra, and R. Williams. Vol. 2110. Springer, 2001, pp. 513–523.

[18] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, and I. Foster. “Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing”. In: *Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies (MSS’01)*. 2001.

[19] G. Alonso and C. Mohan. “Workflow Management Systems: The Next Generation of Distributed Processing Tools”. In: *Advanced Transaction Models and Architectures*. Vol. 1. 1. 1997, pp. 35–62.

[20] J. Ambite and D. Kapoor. “Automatically Composing Data Workflows with Relational Descriptions and Shim Services”. In: *Proceedings of the Semantic Web*. Vol. 4825. Springer, 2007, pp. 15–29.

[21] *An Introduction to the Intel™QuickPath Interconnect*. Tech. rep. 320412-001US. Intel Corporation, 2009.

[22] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J. Navarro. *GLUE Specification v. 2.0*. Tech. rep. GFD-R-P.147. Open Grid Forum, 2009.

[23] A. Anjomshoaa, M. Drescher, A. Ly, S. McGough, D. Pulsipher, and A. Savva. *Job Submission Description Language (JSDL) Specification*. Tech. rep. GFD-R.056 - 1.0. EGEE - Enabling Grids for e-Science, 2005.

[24] F. Arcieri, F. Fioravanti, E. Nardelli, and M. Talamo. “A Layered IT Infrastructure for Secure Interoperability in Personal Data Registry Digital Government Services”. In: *Proceedings of the 14th International IEEE Workshop on Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications*. 2004, pp. 95–102.

[25] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. UCB/EECS-2009-28. Electrical Engineering and Computer Sciences, University of California at Berkeley, 2009.

[26] P. Attie, M. Singh, E. Emerson, A. Sheth, and M. Rusinkiewicz. “Scheduling Workflows by Enforcing Intertask Dependencies”. In: *Distributed Systems Engineering* 3.4 (1996), pp. 222–238.

[27] A. Auweter, A. Bode, M. Brehm, H. Huber, and D. Kranzlmüller. “Principles of Energy Efficiency in High Performance Computing”. In: *Information and Communication on Technology for the Fight against Global Warming*. Ed. by D. Kranzlmüller and A. Toja. Vol. 6868. Springer, 2011, pp. 18–25.

[28] A. Avizienis, J.-C. Laprie, and B. Randell. *Fundamental Concepts of Dependability*. Tech. rep. University of California, Los Angeles (UCLA), 2001.

[29] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. “The NAS Parallel Benchmarks”. In: *Journal of High Performance Computing Applications* 5.3 (1991), pp. 63–73.

[30] D. Bailey and A. Snavely. “Performance Modeling: Understanding the Past and Predicting the Future”. In: *Proceedings of the Euro-Par 2005 Parallel Processing*. Ed. by J. Cunha and P. Medeiros. Vol. 3648. Springer, 2005, pp. 185–195.

[31] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho. “Using Performance Modeling to Design Large-Scale Systems”. In: *Computer* 42.11 (2009), pp. 42–49.

[32] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. Sancho. “Entering the Petaflop Era: The Architecture and Performance of Roadrunner”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’08)*. 2008, 1:1–1:11.

[33] J. B. Barney. “Firm Resources and Sustained Competitive Advantage”. In: *Journal of Management* 7.1 (1991), pp. 99–120.

[34] L. A. Barroso. “The Price of Performance”. In: *Queue* 3.7 (2005), pp. 48–53.

[35] *Basic Concepts in Metrology – Part 1: General concepts*. Tech. rep. DIN 1319-1 : 1995-01. Deutsches Institut für Normierung e.V. (DIN), 1995.

[36] V. Basili. “The Role of Experimentation in Software Engineering: Past, Current, and Future”. In: *Proceedings of the 18th International Conference on Software Engineering*. 1996, pp. 442–449.

[37] R. Basmadjian and H. Meer. “Evaluating and Modeling Power Consumption of Multi-Core Processors”. In: *Proceedings of the 3rd Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy’12)*. 2012, 12:1–12:10.

[38] H. Bauke and S. Mertens. *Cluster Computing: Praktische Einführung in das Hochleistungsrechnen auf Linux-Clustern*. Vol. 1. Springer, 2006. ISBN: 978-3-540-42299-0.

[39] T. Baur, N. gentschen Felde, and H. Reiser. *Konzepte Zum Monitoring im Kern D-Grid*. Tech. rep. Leibniz Rechenzentrum - Munich Network Management Team, 2007.

[40] D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawak, and C. Packer. “BEOWULF: A Parallel Workstation for Scientific Computation”. In: *Proceedings of the 24th International Conference on Parallel Processing*. 1995, pp. 11–14.

[41] S. Becker. “Performance-Related Metrics in the ISO 9126 Standard”. In: *Dependability Metrics*. Lecture Notes in Computer Science 4909 (2008). Ed. by I. Eusgeld, F. C. Freiling, and R. Reussner.

[42] C. Belady, A. Rawson, J. Pflueger, and T. Cader. *Green Grid Data Center Power Efficiency Metrics: PUE and DCIE*. Tech. rep. The Green Grid, 2008.

[43] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, and J. H. and. *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems*. Tech. rep. TR-2008-13. DARPA IPTO, 2008.

[44] R. Berlich, M. Kunze, and K. Schwarz. “Grid Computing in Europe: From Research to Deployment”. In: *Proceedings of the Australasian Workshop on Grid Computing and e-Research (ACSW Frontiers’05)*. 2005, pp. 21–27.

[45] N. Bhatia, S. Alam, and J. Vetter. “Performance Modeling of Emerging HPC Architectures”. In: *Proceedings of the HPCMP Users Group Conference.* 2006, pp. 367–373.

[46] G. D. Bhatt and A. F. Emdad. “An Empirical Examination of the Relationship between Information Technology (IT) Infrastructure, Customer Focus, and Business Advantages”. In: *Journal of Systems and Information Technology* 12.1 (2010), pp. 4–16.

[47] A. Bianzino, A. Raju, and D. Rossi. “Apples-to-Apples: a Framework Analysis for Energy-Efficiency in Networks”. In: *ACM SIGMETRICS Performance Evaluation Review* 38.3 (2010), pp. 81–85.

[48] J. Bigot, Z. Hou, C. Perez, and V. Pichon. “A Low Level Component Model Enabling Performance Portability of HPC Applications”. In: *Proceedings of the 2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC).* IEEE Computer Society, 2012, pp. 701–710.

[49] A. Birolini. *Reliability Engineering: Theory and Practice.* Springer, 2007.

[50] R. Biswas, M. J. Djomehri, R. Hood, H. Jin, C. Kiris, and S. Saini. “An Application-Based Performance Characterization of the Columbia Supercluster”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’05).* 2005, pp. 26–26.

[51] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. “An Updated Set of Basic Linear Algebra Subprograms (BLAS)”. In: *Transactions on Mathematical Software (TOMS)* 28.2 (2002), pp. 135–151.

[52] C. Blanchet, C. Combet, and G. Deleage. “Integrating Bioinformatics Resources on the EGEE Grid Platform”. In: *Proceedings of the 6th International IEEE Symposium on Cluster Computing and the Grid (CCGRID’06).* 2006.

[53] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling Computer and Manufacturing Processes.* Springer, 2001.

[54] A. Bode. “Rechnerarchitektur und Prozessoren”. In: *Informatikhandbuch.* Ed. by P. Rechenberg and G. Pomberger. Hanser, 2006, pp. 333–360.

[55] R. Böhme and F. C. Freiling. “On Metrics and Measurements”. In: *Dependability Metrics.* Ed. by I. Eusgeld, F. C. Freiling, and R. Reussner. Springer, 2008, pp. 7–13.

[56] R. Böhme and R. Reussner. “Validation of Predictions with Measurements”. In: *Dependability Metrics*. Ed. by I. Eusgeld, F. C. Freiling, and R. Reussner. Springer, 2008, pp. 14–18.

[57] J. Borrill, J. Carter, L. Oliker, and D. Skinner. “Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms”. In: *Proceedings of the International IEEE Conference on Parallel Processing (ICPP’05)*. 2005, pp. 119–128.

[58] J. Borrill. “MADCAP: The Microwave Anisotropy Dataset Computational Analysis Package”. In: *Proceedings of the 5th European SGI/Cray MPP Workshop* 5 (1999).

[59] J. Borrill, L. Oliker, J. Shalf, and H. Shan. “Investigation of Leading HPC I/O Performance Using a Scientific-Application Derived Benchmark”. In: *Proceedings of the 21th International ACM/IEEE Conference on Supercomputing (SC’07)*. 2007, pp. 1–12.

[60] J. Borrill, L. Oliker, J. Shalf, H. Shan, and A. Uselton. “HPC Global File System Performance Analysis Using a Scientific-Application Derived Benchmark”. In: *Parallel Computing* 35.6 (2009), pp. 358–373.

[61] A. Boukerche, R. Al-Shaikh, and M. Notare. “Towards Building a Highly-Available Cluster Based Model for High Performance Computing”. In: *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS’06)*. IEEE Computer Society, 2006.

[62] S. Bowers, B. Ludascher, A. Ngu, and T. Critchlow. “Enabling Scientific Workflow Reuse Through Structured Composition of Dataflow and Control-Flow”. In: *Proceedings of the 22th International Conference on Data Engineering Workshops*. 2006, pp. 70–80.

[63] E. L. Boyd, W. Azeem, H.-H. Lee, T.-P. Shih, S.-H. Hung, and E. Davidson. “A Hierarchical Approach to Modeling and Improving the Performance of Scientific Applications on the KSR1”. In: *Proceedings of the International Conference on Parallel Processing (ICPP’94)*. 1994, pp. 188–192.

[64] S. Bradner. *Key Words for Use in RFCs to Indicate Requirement Levels*. 1997.

[65] P. Bridgman. *The Logic of Modern Physics*. MacMillan, 1927.

[66] M. Broadbent and P. Weill. *Leveraging the New Infrastructure: How Market Leaders Capitalize on Information Technology*. Harvard Business Review Press, 1998.

[67] M. Broadbent and P. Weill. “Management by Maxim: How Business and IT Managers Can Create IT Infrastructures”. In: *Sloan Management Review* 38.3 (1997), pp. 77–92.

[68] M. Broadbent, P. Weill, T. Brien, and B.-S. Neo. “Firm Context and Patterns of IT Infrastructure Capability”. In: *Proceedings of the 7th International Conference on Information Systems (ICIS)*. 1996, pp. 174–194.

[69] M. Broadbent, P. Weill, and D. S. Clair. “The Implications of Information Technology Infrastructure for Business Process Redesign”. In: *MIS Quarterly* 23.2 (1999), pp. 159–182.

[70] M. Broadbent, P. Weill, and B.-S. Neo. “Strategic Context and Patterns of IT Infrastructure Capability”. In: *The Journal of Strategic Information Systems* 8.2 (1999), pp. 157–187.

[71] L. Brochard, R. Panda, and S. Vemuganti. “Optimizing Performance and Energy of HPC Applications on POWER7”. In: *Computer Science - Research and Development* 25.3 (2010), pp. 135–140.

[72] R. Brooks and A. Tobias. “Choosing the Best Model: Level of Detail, Complexity, and Model Performance”. In: *Mathematical and Computer Modelling* 24.4 (1996), pp. 1–14.

[73] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. “A Portable Programming Interface for Performance Evaluation on Modern Processors”. In: *Journal of High Performance Computing Applications* 14.3 (2000), pp. 189–204.

[74] A. A. Bush, A. Tiwana, and A. Rai. “Complementarities Between Product Design Modularity and IT Infrastructure Flexibility in IT-Enabled Supply Chains”. In: *IEEE Transactions on Engineering Management* 57.2 (2010), pp. 240–254.

[75] R. Buyya. *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall, 1998.

[76] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberga, and I. Brandicc. “Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility”. In: *Future Generation Computer Systems* 25.6 (2009), pp. 599–616.

[77] T. A. Byrd and D. E. Turner. “Measuring the Flexibility of Information Technology Infrastructure: Exploratory Analysis of a Construct”. In: *Journal of Management Information Systems* 17.1 (2000), pp. 167–208.

[78] K. L. Calvert, M. B. Doar, and E. W. Zegura. "Modeling Internet Topology". In: *IEEE Communications Magazine* 35.6 (1997), pp. 160–163.

[79] J. Cao, D. Kerbyson, and G. Nudd. "Performance Evaluation of an Agent-Based Resource Management infrastructure for Grid Computing". In: *Proceedings of the 1st International ACM/IEEE Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2001, pp. 311–318.

[80] R. Carnap. *Einführung in die Philosophie der Naturwissenschaft*. Vol. 1. Nymphenburger Verlagshandlung, 1969.

[81] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. "24/7 Characterization of Petascale I/O Workloads". In: *Proceedings of the International IEEE Conference on Cluster Computing and Workshops (CLUSTER'09)*. 2009, pp. 1–10.

[82] L. Carrington, A. Snavely, X. Gao, and N. Wolter. "A Performance Prediction Framework for Scientific Applications". In: *Proceedings of the Computational Science (ICCS'03)*. Vol. 2659. Springer, 2003, pp. 926–935.

[83] L. Carrington, A. Snavely, and N. Wolter. "A Performance Prediction Framework for Scientific Applications". In: *Future Generation Computer Systems* 22.3 (2006), pp. 336–346.

[84] L. Carrington, N. Wolter, and A. Snavely. "A Framework for Application Performance Prediction to Enable Scalability Understanding". In: *Proceedings of the Scaling to New Heights Workshop*. 2002.

[85] L. Carrington, N. Wolter, A. Snavely, and C. B. Lee. "Applying an Automated Framework to Produce Accurate Blind Performance Predictions of Full-Scale HPC Applications". In: *Proceedings of the Department of Defense Users Group Conference*. 2004.

[86] J. Carter, J. Borrill, and L. Oliker. "Performance Characteristics of a Cosmology Package On Leading HPC Architectures". In: *Proceedings of the High Performance Computing (HiPC'04)*. Ed. by L. Bougé and V. Prasanna. Vol. 3296. Springer, 2005, pp. 176–188.

[87] G. Casella and R. Berger. *Statistical Inference*. Duxbury, 2001. ISBN: 978-0-53424-312-8.

[88] C. Catlett, W. Allcock, P. Andrews, R. Aydt, R. Bair, N. Balac, B. Banister, T. Barker, M. Bartelt, and P. B. and. *Teragrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications*. Tech. rep. IOS Press, 2008.

[89] C. Chandler, N. DeBardeleben, and C. Leangsuksun. "Towards Resilient High Performance Applications Through Real Time Reliability Metric Generation and Autonomous Failure Correction". In: *Proceedings of the Workshop on Resiliency in High Performance*. 2009, pp. 1–6.

[90] A. Chanopas, D. Krairit, D. B. Khang, and K. Luang. "Managing Information Technology Infrastructure: a New Flexibility Framework". In: *Management Research News* 29.10 (2006), pp. 632–651.

[91] D. Chen, N. Eisley, P. Heidelberger, S. Kumar, A. Mamidala, F. Petrini, R. Senger, Y. Sugawara, R. Walkup, B. Steinmacher-Burow, A. Choudhury, Y. Sabharwal, S. Singhal, and J. Parker. "Looking Under the Hood of the IBM Blue Gene/Q Network". In: *Proceedings of the International ACM/IEEE Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. 2012, pp. 1–12.

[92] S. Chen, Y. Liu, I. Gorton, and A. Liu. "Performance Prediction of Component-Based Applications". In: *Journal of Systems and Software* 74.1 (2005), pp. 35–43.

[93] R. Cheveresan, M. Ramsay, C. Feucht, and I. Sharapov. "Characteristics of Workloads Used in High Performance and Technical Computing". In: *Proceedings of the 21th International Conference on Supercomputing (ICS'07)*. 2007, pp. 73–82.

[94] *China's Supercomputing Strategy Called Out*. 2014. URL: `http://www.hpcwire.com/2014/07/17/dd/` (visited on 08/20/2014).

[95] S. H. Chung, T. A. Byrd, B. R. Lewis, and F. N. Ford. "An Empirical Study of the Relationships between IT Infrastructure Flexibility, Mass Customization, and Business Performance". In: *ACM Special Interest Group on Management Information Systems (SIGMIS)* 36.3 (2005), pp. 26–44.

[96] L. Chwif, M. Barretto, and R. Paul. "On Simulation Model Complexity". In: *Proceedings of the IEEE Winter Simulation Conference*. 2000, pp. 449–455.

[97] W. Cirne and F. Berman. "A Comprehensive Model of the Supercomputer Workload". In: *Proceedings of the International IEEE Workshop on Workload Characterization*. 2001, pp. 140–148.

[98] A. Clematis, D. D'Agostino, E. Danovaro, A. Galizia, A. Quarati, A. Parodi, N. Rebora, T. Bedrina, D. Kranzlmüller, M. Schiffers, B. Jagers, Q. Harpham, and P. Cros. "DRIHM: Distributed Research Infrastructure For Hydro-Meteorology". In: *Proceedings of the 7th International Conference on System of Systems Engineering (SoSE)*. 2012, pp. 149–155.

[99] *CMMI for Systems Engineering/Software Engineering - Version 1.1*. Tech. rep. CMU/SEI-2002-TR-002. CMMI Institute, 2001.

[100] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2002.

[101] P. D. Coddington. "An Analysis of Distributed Computing Software and Hardware for Applications in Computational Physics". In: *Proceedings of the 2nd International Symposium on High Performance Distributed Computing (HPDC'93)*. 1993, pp. 179–186.

[102] L. Collatz and W. Wetterling. *Optimierungsaufgaben*. Vol. 2. Springer, 1971. ISBN: 978-3-540-05616-4.

[103] *Common Information Model*. 2014. URL: `http://www.dmtf.org/standards/cim` (visited on 07/15/2014).

[104] G. Contreras and M. Martonosi. "Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events". In: *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'05)*. 2005, pp. 221–226.

[105] S. Cox, J. Cox, R. Boardman, S. Johnston, M. Scott, and N. O'Brien. "Iridis-pi: A Low-Cost, Compact Demonstration Cluster". In: *Cluster Computing* (2013), pp. 1–10.

[106] H. J. Curnow and B. A. Wichmann. "A Synthetic Benchmark". In: *The Computer Journal* 19.1 (1976), pp. 43–49.

[107] D. D'Agostino, A. Clematis, A. Galizia, A. Quarati, E. Danovaro, L. Roverelli, G. Zereik, D. Kranzlmüller, M. Schiffers, N. gentschen Felde, C. Straube, A. Parodi, E. Fiori, F. Delogu, O. Caumont, E. Richard, L. Garrote, Q. Harpham, H. Jagers, V. Dimitrijevic, and L. Dekic. "The DRIHM Project: A Flexible Approach to Integrate HPC, Grid and Cloud Resources for Hydro-Meteorological Research". In: *Proceedings of the International ACM/IEEE Conference on High Performance Computing, Networking, Storage and Analysis (SC'14)*. Nov. 2014.

[108] J. T. Daly. "A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps". In: *Future Generation Computer Systems* 22.3 (2006), pp. 303–312.

[109] E. Danovaro, L. Roverelli, G. Zereik, A. Galizia, D. D'Agostino, G. Paschina, A. Quarati, A. Clematis, F. Delogu, E. Fiori, A. Parodi, C. Straube, N. Felde, Q. Harpham, B. Jagers, L. Garrote, L. Dekic, M. Ivkovic, O. Caumont, and E. Richard. "Setup an Hydro-Meteo Experiment in Minutes: the DRIHM e-Infrastructure for Hydro-Meteo Research". In: *Proceedings of the 10th International IEEE Conference on e-Science*. Oct. 2014.

[110] T. Davenport and J. Linder. "Information Management Infrastructure: the New Competitive Weapon?" In: *Proceedings of the 27th International IEEE Hawaii Conference on System Sciences*. 1994, pp. 885–896.

[111] S. Davidson and J. Freire. "Provenance and Scientific Workflows: Challenges and Opportunities". In: *Proceedings of the International ACM SIGMOD Conference on Management of Data*. 2008, pp. 1345–1350.

[112] K. Davis, A. Hoisie, G. Johnson, D. Kerbyson, M. Lang, S. Pakin, and F. Petrini. "A Performance and Scalability Analysis of the BlueGene/L Architecture". In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'04)*. 2004.

[113] H. Davulcu, M. Kifer, L. R. Pokorny, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. Dawson. "Modeling and Analysis of Interactions in Virtual Enterprises". In: *Proceedings of the 9th International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises (RIDE-VE'99)*. 1999, pp. 12–18.

[114] E. Deelman and A. Chervenak. "Data Management Challenges of Data-Intensive Scientific Workflows". In: *Proceedings of the 8th International ACM/IEEE Symposium on Cluster Computing and the Grid (CCGRID'08)*. 2008, pp. 687–692.

[115] E. Deelman and Y. Gil. "Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges". In: *Proceedings of the 2nd International IEEE Conference on e-Science and Grid Computing (e-Science'06)*. 2006, pp. 144–150.

[116] E. Deelman, D. Gannon, M. Shields, and I. Taylor. "Workflows and e-Science: An Overview of Workflow System Features and Capabilities". In: *Future Generation Computer Systems* 25.5 (2009), pp. 528–540.

[117] P. Denning. "A New Social Contract for Research". In: *Communications of the ACM* 40.2 (1997), pp. 132–134.

[118] W. Denzel, J. Li, P. Walker, and Y. Jin. “A Framework for End-to-End Simulation of High-Performance Computing Systems”. In: *SIMULATION* 86.5 (2010), pp. 331–350.

[119] *Description of Work – Part B - Combination of Collaborative Project and Coordination and Support Action.* Tech. rep. FP7-283568 DRIHM. CIMA Research Foundation et al., 2012.

[120] J. Devore. *Probability and Statistics for Engineering and the Sciences.* Vol. 3. Brooks/Cole, 1991. ISBN: 978-0-53414-352-7.

[121] I. Diaz, G. Fernandez, M. J. Martinm, P. Gonzalez, and J. Tourino. “Integrating the Common Information Model with MDS4”. In: *Proceedings of the 9th International ACM/IEEE Conference on Grid Computing.* IEEE Computer Society, 2008, pp. 298–303.

[122] K. M. Dixit. “Overview of the SPEC Benchmarks”. In: *Benchmark Handbook: For Database and Transaction Processing Systems.* Ed. by J. Gray. Morgan Kaufmann Publishers Inc., 1992, pp. 489–521.

[123] DMTF. *Common Information Model (CIM) Metamodel (Specification).* Tech. rep. DSP0004. Distributed Management Task Force (DMTF), 2012.

[124] J. Dongarra. “Performance of Various Computers Using Standard Linear Equations Software in a FORTRAN Environment”. In: *ACM SIGARCH Computer Architecture News* 16.1 (1988), pp. 47–69.

[125] J. J. Dongarra. “The LINPACK Benchmark: An Explanation”. In: *Supercomputing.* Ed. by E. N. Houstis, T. S. Papatheodorou, and C. D. Polychronopoulos. Vol. 297. Springer, 1988, pp. 456–474.

[126] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Y. Ishikawa, F. Johnson, Z. Jin, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. Mueller, W. Nagel, H. Nakashima, M. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. V. D. Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick. “The International Exascale Software Project Roadmap”. In: *Journal of High Performance Computing Applications* 25.1 (2011), pp. 3–60.

[127] J. Dongarra, P. Luszczek, and A. Petitet. "The LINPACK Benchmark: Past, Present And Future". In: *Concurrency and Computation: Practice and Experience* 15.9 (2003), pp. 803–820.

[128] J. Dongarra, H. Meuer, E. Strohmaier, and H. Simon. *Top 500 List of Supercomputer Sites*. 2014. URL: `http://www.top500.org/` (visited on 04/12/2014).

[129] J. Dongarra, J. Bunch, C. Moler, and G. Stewart. "LINPACK Users Guide". In: *SIAM* (1979).

[130] E. Duesterwald, J. Torrellas, and S. Dwarkadas. "Characterizing and Predicting Program Behavior and Its Variability". In: pp. 220–231.

[131] N. B. Duncan. "Capturing Flexibility of Information Technology Infrastructure: A Study of Resource Characteristics and Their Measure". In: *Journal of Management Information Systems* 12.2 (1995), pp. 37–57.

[132] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. "Full-System Power Analysis and Modeling for Server Environments". In: *Proceedings of the Workshop on Modeling, Benchmarking, and Simulation*. 2006, pp. 70–77.

[133] P. N. Edwards, S. J. Jackson, G. C. Bowker, and C. P. Knobel. *Understanding Infrastructure: Dynamics, Tensions, and Design*. Tech. rep. National Science Foundation (NSF), 2007.

[134] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. "Combining Partial Redundancy and Checkpointing for HPC". In: *Proceedings of the 32th International IEEE Conference on Distributed Computing Systems (ICDCS)*. 2012, pp. 615–626.

[135] *Energy-Efficient Data Centres – Best-Practice Examples from Europe, the USA and Asia*. Tech. rep. Federal Ministry for the Environment, Nature Conservation, Building and Nuclear Safety (BMUB), 2010.

[136] *European Grid Infrastructure (EGI) - Services for researchers*. 2014. URL: `http://www.egi.eu/services/researchers/` (visited on 07/03/2014).

[137] I. Eusgeld, B. Fechner, F. Salfner, M. Walter, P. Limbourg, and L. Zhang. "Hardware Reliability". In: *Dependability Metrics*. Ed. by I. Eusgeld, F. C. Freiling, and R. Reussner. Vol. 4909. Springer, 2008, pp. 59–103.

[138] X. Fan, W.-D. Weber, and L. A. Barroso. "Power Provisioning for a Warehouse-Sized Computer". In: *Proceedings of the 34th International Symposium on Computer Architecture (ISCA'07)*. 2007, pp. 13–23.

[139] B. Farbey, D. Targett, and F. Land. “The Great IT Benefit Hunt”. In: *European Management Journal* 12.3 (1994), pp. 270–279.

[140] X. Fei and S. Lu. “A Dataflow-Based Scientific Workflow Composition Framework”. In: *IEEE Transactions on Services Computing* 5.1 (2012), pp. 45–58.

[141] L. Fink and S. Neumann. “Exploring the Perceived Business Value of the Flexibility Enabled by Information Technology Infrastructure”. In: *Information & Management* 46.2 (2009), pp. 90–99.

[142] I. Foster. “Globus Toolkit Version 4: Software for Service-Oriented Science”. In: *Network and Parallel Computing*. Vol. 3779. Springer, 2005, pp. 2–13.

[143] I. Foster. *What is the Grid? A Three Point Checklist*. Tech. rep. Argonne National Laboratory & University of Chicago, 2002.

[144] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[145] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. “Grid Services for Distributed System Integration”. In: *Computer* 35.6 (2002), pp. 37–46.

[146] I. Foster, C. Kesselman, and S. Tuecke. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”. In: *Journal of High Performance Computing Applications* 15.3 (2001), pp. 200–222.

[147] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. “The Physiology of the Grid”. In: *Grid Computing - Making the Global Infrastructure Reality*. Ed. by F. Berman, G. Fox, and A. J. G. Hey. John Wiley & Sons, Inc., 2003, pp. 217–249.

[148] I. Foster and C. Kesselman. “The Grid in a Nutshell”. In: *Grid Resource Management - State of the Art and Future Trends*. Ed. by J. Nabrzyski, J. Schopf, and J. Weglarz. Kluwer Academic Publishers, 2004, pp. 3–13.

[149] I. Foster, Y. Zhao, I. Raicu, and S. Lu. “Cloud Computing and Grid Computing 360-Degree Compared”. In: *Proceedings of the Grid Computing Environments Workshop (GCE’08)*. 2008, pp. 1–10.

[150] D. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003. ISBN: 9-780-47146-227-9.

[151] J. Freire, D. Koop, E. Santos, and C. T. Silva. “Provenance for Computational Tasks: A Survey”. In: *Computing in Science Engineering* 10.3 (2008), pp. 11–21.

[152] T. Freund. *Software Engineering durch Modellierung wissensintensiver Entwicklungsprozesse.* GITO mbH - Verlag für Industrielle Informationstechnik und Organisation, 2007.

[153] M. Frigo and S. Johnson. "The Design and Implementation of FFTW3". In: *Proceedings of the IEEE* 93.2 (2005), pp. 216–231.

[154] *Fundamentals of Metrology - Part 3: Evaluation of Measurements of a Single Measurand, Measurement Uncertainty.* Tech. rep. DIN 1319-3:1996-05. Deutsches Institut für Normierung e.V. (DIN), 1996.

[155] *Fundamentals of Metrology - Part 4: Evaluation of Measurements; Uncertainty of Measurement.* Tech. rep. DIN 1319-4:1999-02. Deutsches Institut für Normierung e.V. (DIN), 1999.

[156] V. Fung, W. Fung, and Y. Wind. *Competing in a Flat World: Building Enterprises for a Borderless World.* Pearson Prentice Hall, 2007. ISBN: 978-0-13261-818-2.

[157] K. Fürlinger, C. Klausecker, and D. Kranzlmüller. "Towards Energy Efficient Parallel Computing on Consumer Electronic Devices". In: *Information and Communication on Technology for the Fight against Global Warming.* Ed. by D. Kranzlmüller and A. Toja. Vol. 6868. Springer, 2011, pp. 1–9.

[158] M. Gamell, I. Rodero, M. Parashar, J. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. Landge, A. Gyulassy, P. McCormick, S. Pakin, V. Pascucci, and S. Klasky. "Exploring Power Behaviors and Tradeoffs of In-Situ Data Analytics". In: *Proceedings of the International ACM/IEEE Conference on High Performance Computing, Networking, Storage and Analysis (SC'13).* 2013, 77:1–77:12.

[159] E. Gamma, R. Helm, and R. E. Johnson. *Design Patterns – Elements of Reusable Object-Oriented Software.* Vol. 1. Addison-Wesley, 1994. ISBN: 9-780-32170-069-8.

[160] A. Gara, M. A. Blumrich, D. Chen, G. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas. "Overview of the Blue Gene/L System Architecture". In: *IBM Journal of Research and Development* 49.2.3 (2005), pp. 195–212.

[161] *Gauss Centre for Supercomputing.* 2014. URL: `www.gauss-centre.eu` (visited on 01/15/2014).

[162] R. Ge, X. Feng, W.-C. Feng, and K. W. Cameron. "CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters". In: *Proceedings of the International Conference on Parallel Processing (ICPP'07)*. 2007, pp. 18–26.

[163] W. Gentzsch. "D-Grid, an e-Science Framework for German Scientists". In: *Proceedings of the 5th International Symposium on Parallel and Distributed Computing (ISPDC'06)*. 2006, pp. 12–13.

[164] *Green 500 List of Supercomputer Sites*. 2014. URL: `http://www.green500.org/` (visited on 11/30/2013).

[165] A. Gregoriades and A. Sutcliffe. "Workload Prediction for Improved Design and Reliability of Complex Systems". In: *Reliability Engineering & System Safety* 93.4 (2008), pp. 530–549.

[166] *Grid Schema Working Group (GLUE-WG)*. 2014. URL: `https://forge.ogf.org/sf/projects/glue-wg` (visited on 07/15/2014).

[167] A. Grimshaw, E. West, and W. Pearson. "No Pain and Gain! - Experiences with Mentat on a Biological Application". In: *Concurrency: Practice and Experience* 5.4 (1993), pp. 309–328.

[168] P. Gritzmann. *Grundlagen der mathematischen Optimierung: Diskrete Strukturen, Komplexitätstheorie, Konvexitätstheorie, Lineare Optimierung, Simplex-Algorithmus, Dualität*. Vol. 1. Springer, 2013. ISBN: 978-3-528-07290-2.

[169] V. Gruhn, D. Pieper, and C. Röttgers. *MDA: Effektives Software-Engineering Mit UML2 und Eclipse*. Springer, 2006. ISBN: 978-3-54028-744-5.

[170] M. Guest. *The Scientific Case for High Performance Computing in Europe 2012-2020*. Insight Publishers Ltd, 2013.

[171] L. Guijarro. "Interoperability Frameworks and Enterprise Architectures in e-Government Initiatives in Europe and the United States". In: *Government Information Quarterly* 24.1 (2007), pp. 89–101.

[172] J. Gustafson. "Reevaluating Amdahl's Law". In: *Communications of the ACM* 31.5 (1988), pp. 532–533.

[173] D. Hackenberg, R. Schöne, D. Molka, M. Müller, and A. Knüpfer. "Quantifying Power Consumption Variations of HPC Systems Using SPEC MPI Benchmarks". In: *Computer Science - Research and Development* 25.3 (2010), pp. 155–163.

[174] H. Hacker, C. Trinitis, J. Weidendorfer, and M. Brehm. "Considering GPGPU for HPC Centers: is it Worth the Effort?" In: *Facing the Multicore-Challenge*. Ed. by R. Keller, D. Kramer, and J.-P. Weiss. Vol. 6310. Springer, 2011, pp. 118–130.

[175] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. "Measuring Energy Consumption for Short Code Paths Using RAPL". In: *ACM SIGMETRICS Performance Evaluation Review* 40.3 (2012), pp. 13–17.

[176] A. Hanna and S. Rance. *ITIL™Glossary and Abbreviations - English.* Tech. rep. 1.0. Office of Government Commerce (OGC), 2011.

[177] O. Hanseth and K. Braa. "Technology as Traitor: Emergent SAP Infrastructure in a Global Organization". In: *Proceedings of the International Conference on Information systems (ICIS'98)*. 1998, pp. 188–196.

[178] J. Happe. "Analytical Performance Metrics". In: *Dependability Metrics*. Lecture Notes in Computer Science 4909 (2008). Ed. by I. Eusgeld, F. C. Freiling, and R. Reussner.

[179] R. Hatzinger, K. Hornik, and H. Nagel. *R - Einführung in die angewandte Statistik*. Vol. 1. Pearson Studium, 2011. ISBN: 978-3-86894-060-2.

[180] R. Hedges, B. Loewe, T. McLarty, and C. Morrone. "Parallel File System Testing for the Lunatic Fringe: the Care and Feeding of Restless I/O Power Users". In: *Proceedings of the 22th IEEE Conference on Mass Storage Systems and Technologies*. 2005, pp. 3–17.

[181] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures, and Their Operational Application*. Morgan Kaufmann Publishers, Inc., 1998. ISBN: 9-781-55860-571-8.

[182] H. Hellwagner. "Arbeitsspeicher- und Bussysteme". In: *Informatikhandbuch*. Ed. by P. Rechenberg and G. Pomberger. Hanser, 2006, pp. 361–379.

[183] C. Hempel. *Philosophy Of Natural Science*. 1966. ISBN: 978-0-13663-823-0.

[184] J. L. Hennessy and D. A. Patterso. *Computer Architecture, A Quantitative Approach*. 2003.

[185] R. Hennicker. *Softwaretechnik – Kapitel 3 – Objektorientierte Analyse*. 2012.

[186] A. Hevner. "A Three Cycle View Of Design Science Research". In: *Scandinavian Journal of Information Systems* 19.2 (2007), pp. 87–92.

[187] A. Hevner, S. March, J. Park, and S. Ram. "Design Science In Information Systems Research". In: *MIS Quarterly* 28.1 (2004), pp. 75–105.

[188] T. Hey and A. Trefethen. "e-Science and its Implications". In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 361.1809 (2003), pp. 1809–1825.

[189] T. Hey and A. Trefethen. "The UK e-Science Core Programme and the Grid". In: *Future Generation Computer Systems* 18.8 (2002), pp. 1017–1031.

[190] D. Hitchcock and L. Nowell. *Advanced Architectures and Critical Technologies for Exascale Computing*. Tech. rep. DE-FOA-0000255. U.S. Department of Energy (DoE), 2010.

[191] T. Hoefler, T. Mehlan, A. Lumsdaine, and W. Rehm. "Netgauge: A Network Performance Measurement Framework". In: *Proceedings of the High Performance Computing and Communications (HPCC'07)*. Ed. by R. Perrott, B. Chapman, J. Subhlok, R. F. Mello, and L. Yang. Vol. 4782. Springer, 2007, pp. 659–671.

[192] A. Hoisie, G. Johnson, D. Kerbyson, M. Lang, and S. Pakin. "A Performance Comparison Through Benchmarking and Modeling of Three Leading Supercomputers: Blue Gene/L, Red Storm, and Purple". In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'06)*. 2006, pp. 3–13.

[193] A. Hoisie, O. Lubeck, and H. Wasserman. "Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications". In: *Journal of High Performance Computing Applications* 14.4 (2000), pp. 330–346.

[194] J. Hollingsworth and B. Tierney. "Instrumentation and Monitoring". In: *The Grid 2: Blueprint for a New Computing Infrastructure*. Ed. by I. Foster and C. Kesselman. Elsevier, 2003, pp. 319–351.

[195] C.-H. Hsu and W.-C. Feng. "A Power-Aware Run-Time System for High-Performance Computing". In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'05)*. 2005, pp. 1–9.

[196] C.-H. Hsu and W.-C. Feng. “Effective Dynamic Voltage Scaling Through CPU-Boundedness Detection”. In: *Power-Aware Computer Systems*. Ed. by B. Falsafi and T. N. Vijaykumar. Vol. 3471. Springer, 2005, pp. 135–149.

[197] H.-G. Hwang, R. Yeh, H.-G. Chen, J. J. Jiang, and G. Klein. “IT Investment Strategy And IT Infrastructure Services”. In: *The Review of Business Information Systems* 6.2 (2002).

[198] *IEEE Recommended Practice for Architectural Description for Software-Intensive Systems*. Tech. rep. IEEE Std 1471-2000. Institute of Electrical and Electronics Engineerings, Inc. (IEEE), 2000.

[199] *IEEE Standard for Software Quality Assurance Plans*. Tech. rep. IEEE Std 730-1998. Institute of Electrical and Electronics Engineerings, Inc. (IEEE), 1998.

[200] *IEEE Standard Glossary of Software Engineering Terminology*. Tech. rep. IEEE Std 610.12-199. Institute of Electrical and Electronics Engineers (IEEE), 1990.

[201] J. Iivari. “A Paradigmatic Analysis of Information Systems as a Design Science”. In: *Scandinavian Journal of Information Systems* 19.2 (2007), pp. 39–64.

[202] E. Imamagic and D. Dobrenic. “Grid Infrastructure Monitoring System Based on Nagios”. In: *Proceedings of the Workshop on Grid Monitoring (GMW’07)*. ACM, 2007, pp. 23–28.

[203] C. Isci and M. Martonosi. “Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data”. In: *Proceedings of the 36th International IEEE/ACM Symposium on Microarchitecture (MICRO’36)*. 2003, pp. 93–115.

[204] *ISO/IEC 27001:2013 – Information Technology - Security Techniques - Information Security Management Systems - Requirements*. Tech. rep. ISO/IEC 27001:2013. International Organization for Standardization (ISO), 2013.

[205] I. Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach*. Pearson Education, 1992. ISBN: 9-788-13170-408-0.

[206] J. Jaffar and M. Maher. “Constraint Logic Programming: a Survey”. In: *The Journal of Logic Programming* 19/20 (1994), pp. 503–581.

[207] R. Jain. *The Art of Computer Systems Performance Analysis*. Vol. 1. John Wiley & Sons, Inc., 1991. ISBN: 978-0-47150-336-1.

[208] W. Jansen. *Directions in Security Metrics Research*. Tech. rep. NISTIR 7564. National Institute of Standards and Technology, 2009.

[209] C. Janssen, H. Adalsteinsson, and J. Kenny. "Using Simulation to Design Extremescale Applications and Architectures: Programming Model Exploration". In: *ACM SIGMETRICS Performance Evaluation Review - Special Issue on the 1st Intl. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems (PMBS'10)* 38.4 (2011), pp. 4–8.

[210] A. Jaquith. *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Vol. 1. Addison-Wesley, 2007. ISBN: 978-0-32134-998-9.

[211] D. Jensen and A. Rodrigues. "Embedded Systems and Exascale Computing". In: *Computing in Science Engineering* 12.6 (2010), pp. 20–29.

[212] S. Jha, M. Cole, D. Katz, M. Parashar, O. Rana, and J. Weissman. "Distributed Computing Practice for Large-scale Science and Engineering Applications". In: *Concurrency and Computation: Practice and Experience* 25.11 (2013), pp. 1559–1585.

[213] H. Jin, M. Frumkin, and J. Yan. *The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance*. Tech. rep. NAS-99-011. NAS System Division, NASA Ames Research Center, 1999.

[214] W. Johnston, P. Hanna, and R. Millar. "Advances in Dataflow Programming Languages". In: *ACM Computing Surveys (CSUR)* 36.1 (2004), pp. 1–34.

[215] W. Jones, J. Daly, and N. DeBardeleben. "Application Monitoring and Checkpointing in HPC: Looking Towards Exascale Systems". In: *Proceedings of the 50th Annual Southeast Regional Conference*. 2012, pp. 262–267.

[216] W. Jones, J. Daly, and N. DeBardeleben. "Impact of Sub-Optimal Checkpoint Intervals on Application Efficiency in Computational Clusters". In: *Proceedings of the 19th International ACM Symposium on High Performance Distributed Computing (HPDC'10)*. ACM, 2010, pp. 276–279.

[217] R. Joseph and M. Martonosi. "Run-time Power Estimation in High Performance Microprocessors". In: *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'01)*. 2001, pp. 135–140.

[218] A. Kaplan. *The Conduct of Inquiry*. Chandler, 1964.

[219] K. L. Karavanic and B. P. Miller. “Improving Online Performance Diagnosis by the Use of Historical Performance Data”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing*. 1999, pp. 42–42.

[220] T. R. Kayworth, D. Chatterjee, and V. Sambamurthy. “Theoretical Justification for IT Infrastructure Investments”. In: *Information Resources Management Journal (IRMJ)* 14.3 (2001), pp. 5–14.

[221] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. “Science Clouds: Early Experiences in Cloud Computing for Scientific Applications”. In: *Cloud Computing and its Applications* (2008), pp. 825–830.

[222] K. Keahey, I. Foster, T. Freeman, and X. Zhang. “Virtual Workspaces: Achieving Quality of Service and quality of Life in the Grid”. In: *Scientific Programming* 13.4 (2005), pp. 265–275.

[223] G. Kearns and A. Lederer. “A Resource-Based View of Strategic IT Alignment: How Knowledge Sharing Creates Competitive Advantage”. In: *Decision Sciences* 34.1 (2003), pp. 1–29.

[224] P. G. W. Keen. *Every Manager’s Guide to Information Technology*. Harvard Business School Press, 1995.

[225] P. G. W. Keen. *Shaping the Future: Business Design through Information Technology*. Harvard Business School Press, 1991.

[226] D. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings. “Predictive Performance and Scalability Modeling of A Large-Scale Application”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’01)*. 2001, pp. 37–49.

[227] D. Kerbyson, A. Hoisie, and H. Wasserman. “A Performance Comparison Between the Earth Simulator and Other Terascale Systems on a Characteristic ASCI Workload”. In: *Concurrency and Computation: Practice and Experience* 17.10 (2005), pp. 1219–1238.

[228] D. Kerbyson, A. Hoisie, and H. Wasserman. “Modelling the Performance of Large-Scale Systems”. In: *IEE Proceedings-Software* 150.4 (2003), pp. 214–221.

[229] O. Khalili, J. He, C. Olschanowsky, A. Snavely, and H. Casanova. “Measuring the Performance and Reliability of Production Computational Grids”. In: *Proceedings of the 7th International ACM/IEEE Conference on Grid Computing*. 2006, pp. 293–300.

[230] W. R. King and P. R. Flor. "The Development of Global IT Infrastructure". In: *Omega – Special Issue on Multiple Criteria Decision Making for Engineering* 36.3 (2008). Ed. by M. M. Wiecek, M. Ehrgott, G. Fadel, and J. Figueira, pp. 486–504.

[231] G. Kleindorfer, L. O'Neill, and R. Ganeshan. "Validation in Simulation: Various Positions in the Philosophy of Science". In: *Management Science* 44.8 (1998), pp. 1087–1099.

[232] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture - Practice And Promise.* Addison-Wesley, 2003. ISBN: 978-0-32119-442-8.

[233] S. Kleuker. *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten.* Vol. 3. Springer, 2013. ISBN: 9-783-65800-641-9.

[234] M. Klinger. "Evaluating the Feasibility and Performance of a Model Raspberry Pi Beowulf Cluster". Bachelor thesis. Ludwig-Maximilians-Universität München, 2013.

[235] S. Knittl. "Werkzeugunterstützung für interorganisationales IT-Service-Management - ein Referenzmodell für die Erstellung einer ioCMDB". PhD thesis. Technische Universität München (TUM), 2012.

[236] K. Koch, R. Baker, and R. Alcouffe. "Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on A Massively Parallel Processor". In: *Transactions of the American Nuclear Society* 65.108 (1992), pp. 198–199.

[237] N. Koch, A. Knapp, G. Zhang, and H. Baumeister. "Uml-Based Web Engineering: An Approach Based on Standards". In: *Proceedings of the Web Engineering: Modelling and Implementing Web Applications.* Ed. by G. Rossi, O. Pastor, D. Schwabe, and L. Olsina. Vol. 12. 2008, pp. 157–191.

[238] S. Kottha, K. Peter, T. Steinke, J. Bart, J. Falkner, A. Weisbecker, F. Viezens, Y. Mohammed, U. Sax, A. Hoheisel, T. Ernst, D. Sommerfeld, D. Krefting, and M. Vossberg. "Medical Image Processing in MediGRID". In: *Proceedings of the German e-Science Conference.* 2007, pp. 1–10.

[239] H. Koziolek. "Introduction to Performance Metrics". In: *Dependability Metrics.* Lecture Notes in Computer Science 4909 (2008). Ed. by I. Eusgeld, F. C. Freiling, and R. Reussner.

[240] H. Koziolek and J. Happe. “Performance Metrics for Specific Domains”. In: *Dependability Metrics*. Lecture Notes in Computer Science 4909 (2008). Ed. by I. Eusgeld, F. C. Freiling, and R. Reussner.

[241] D. Krefting, J. Bart, K. Beronov, O. Dzhimova, J. Falkner, M. Hartung, A. Hoheisel, T. Knoch, T. Lingner, Y. Mohammed, K. Peter, E. Rahm, U. Sax, D. Sommerfeld, T. Steinke, T. Tolxdorff, F. Viezens, M. Vossberg, and A. Weisbecker. “MediGRID: Towards a User Friendly Secured Grid Infrastructure”. In: *Future Generation Computer Systems* 25.3 (2009), pp. 326–336.

[242] H. G. Kruse. *Leistungsbewertung bei Computersystemen – Praktische Performance-Analyse von Rechnern und ihrer Kommunikation*. Vol. 1. Springer, 2009. ISBN: 9-783-54071-053-0.

[243] B. Kryza,
bibinitperiod Skitaã, J. Kitowski, M. Li, and T. Itagaki. “Analysis of Interoperability Issues Between EGEE and VEGA Grid Infrastructures”. In: *Proceedings of the High Performance Computing and Communications*. Ed. by M. Gerndt. Vol. 4208. Springer, 2006, pp. 793–802.

[244] S. Laan. *IT Infrastructure Architecture – Infrastructure Building Blocks and Concepts*. Lulu Press, Inc., 2011. ISBN: 9-781-44788-128-5.

[245] J. P. Lafore, J. Stein, N. Asencio, P. Bougeault, V. Ducrocq, J. Duron, C. Fischer, P. Héreil, P. Mascart, V. Masson, J. P. Pinty, J. L. Redelsperger, E. Richard, and J.-G. Arellano. “The Meso-NH Atmospheric Simulation System. Part I: Adiabatic Formulation and Control Simulations”. In: *Annales Geophysicae* 16.1 (1997), pp. 90–109.

[246] H. Langweg. “Framework for Malware Resistance Metrics”. In: *Proceedings of the 2nd Workshop on Quality of Protection (QoP’06)*. ACM, 2006, pp. 39–44.

[247] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Vol. 3. Prentice Hall, 2004. ISBN: 978-0-13148-906-6.

[248] C. C. H. Law and E. W. T. Ngai. “IT Infrastructure Capabilities and Business Process Improvements: Association with IT Governance Characteristics”. In: *Information Resources Management Journal (IRMJ)* 20.4 (2007).

[249] E. A. Lee and T. M. Parks. “Dataflow Process Networks”. In: *Proceedings of the IEEE* 83.5 (1995), pp. 773–801.

[250] S.-J. Lee, H.-K. Lee, and P.-C. Yew. “Runtime Performance Projection Model for Dynamic Power Management”. In: *Advances in Computer Systems Architecture*. Ed. by L. Choi, Y. Paek, and S. Cho. Vol. 4697. Springer, 2007, pp. 186–197.

[251] *Leitfaden zur Angabe von Unsicherheiten beim Messen*. Tech. rep. DIN V ENV 13005. Deutsche Institut für Normung e. V. (DIN), 1999.

[252] H. Levy and D. Clark. “On the Use of Benchmarks for Measuring System Performance”. In: *ACM SIGARCH Computer Architecture News* 10.6 (1982), pp. 5–8.

[253] H. Li, D. Groep, and L. Wolters. “Workload Characteristics of a Multi-Cluster Supercomputer”. In: *Proceedings of the Job Scheduling Strategies for Parallel Processing*. Ed. by D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Vol. 3277. Springer, 2005, pp. 176–193.

[254] C. Lin, S. Lu, X. Fei, D. Pai, and J. Hua. “A Task Abstraction and Mapping Approach to the Shimming Problem in Scientific Workflows”. In: *Proceedings of the International IEEE Conference on Services Computing (SCC’09)*. 2009, pp. 284–291.

[255] T. Lindinger. “Optimierung des Wirkungsgrades virtueller Infrastrukturen”. PhD thesis. Fakultät für Mathematik, Informatik und Statistik der Ludwig-Maximilians-Universität München, 2009.

[256] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda. “Performance Comparison of MPI Implementations Over InfiniBand, Myrinet and Quadrics”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’03)*. 2003, pp. 58–58.

[257] S. Liu. “A Practical Framework for Discussing IT Infrastructure”. In: *IT Professional* 4.4 (2002), pp. 14–21.

[258] D. Lojewski. “Benchmarking und Systemvergleich von Mikrocontrollern”. Diploma thesis. Hochschule Harz - Fachbereich Automatisierung und Informatik - Studiengang Informatik im Netz, 2007.

[259] C.-D. Lu. “Failure Data Analysis of HPC Systems”. In: *Journal of CoRR* abs/1302.4779 (2013).

[260] B. Ludascher, S. Bowers, T. McPhillips, and N. Podhorszki. “Scientific Workflows: More e-Science Mileage from Cyberinfrastructure”. In: *Proceedings of the 2nd International IEEE Conference on e-Science and Grid Computing (e-Science’06)*. 2006, pp. 145–153.

[261] B. Ludäscher, M. Weske, T. McPhillips, and S. Bowers. “Scientific Workflows: Business as Usual?” In: *Proceedings of the Business Process Management*. Vol. 5701. Springer, 2009, pp. 31–47.

[262] J. Luftman and T. Ben-Zvi. “Key Issues for IT Executives 2009: Difficult Economy’s Impact on IT”. In: *MIS Quarterly Executive* 9.1 (2010), pp. 203–213.

[263] J. Luftman and T. Ben-Zvi. “Key Issues for IT Executives 2010: Judicious IT Investments Continue Post-Recession”. In: *MIS Quarterly Executive* 9.4 (2010), pp. 263–273.

[264] C. Malone and C. Belady. “Metrics to Characterize Data Center & IT Equipment Energy Use”. In: *Proceedings of the Digital Power Forum*. 2006.

[265] H. Mao, L. Huang, and M. Li. “Web Resource Monitoring Based on Common information Model”. In: *Proceedings of the IEEE Services Computing (APSCC’06)*. 2006, pp. 520–525.

[266] S. March and G. Smith. “Design and Natural Science Research on Information Technology”. In: *Decision Support Systems* 15.4 (1995), pp. 251–266.

[267] G.-P. Marcu. “Architekturkonzepte für interorganisationales Fehlermanagement”. PhD thesis. Fakultät für Mathematik, Informatik und Statistik der Ludwig-Maximilians-Universität München, 2011.

[268] M. Al-Mashari and M. Zairi. “Creating a Fit Between BPR and IT Infrastructure: A Proposed Framework for Effective Implementation”. In: *Journal of Flexible Manufacturing Systems* 12.4 (2000), pp. 253–274.

[269] A. Mayer, S. McGough, N. Furmento, W. Lee, M. Gulamali, S. Newhouse, and J. Darlington. “Workflow Expression: Comparison of Spatial and Temporal Approaches”. In: *Proceedings of the Workflow in Grid Systems Workshop (GGF-10)*. 2004.

[270] P. Mayer. “MDD4SOA – Model-Driven Development for Service-Oriented Architectures”. PhD thesis. Ludwig-Maximilians-Universität München, 2010.

[271] J. McCalpin. “Memory Bandwidth and Machine Balance in Current High Performance Computers”. In: *Technical Committee on Computer Architecture (TCCA) Newsletter* (1995), pp. 19–25.

[272] J. McCalpin. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Tech. rep. Continually updated. University of Virginia, 2007.

[273] J. McCalpin. *STREAM: Sustainable Memory Bandwidth in High Performance Computers.* 2014. URL: `http://www.cs.virginia.edu/stream/` (visited on 07/31/2014).

[274] D. McKay and D. Brockway. “Building I/T infrastructure for the 1990s”. In: *Stage by Stage* 9.3 (1989), pp. 1–11.

[275] C. R. Mechoso, C. C. Ma, J. D. Farrara, J. A. Spahr, R. W. Moore, W. P. Dannevik, M. F. Wehner, P. Eltgroth, and A. A. Mirin. “Distributing A Climate Model Across Gigabit Networks”. In: *Proceedings of the 1st International Symposium on High-Performance Distributed Computing (HPDC-1).* 1992, pp. 16–25.

[276] J.-F. Méhaut and M. Martinasso. “A Contention-Aware Performance Model for HPC-Based Networks: A Case Study of the InfiniBand Network”. In: *Proceedings of the Euro-Par 2011 Parallel Processing.* Ed. by E. Jeannot. Vol. 6852. Springer, 2011, pp. 91–102.

[277] D. Meisner, B. Gold, and T. Wenisch. “PowerNap: Eliminating Server Idle Power”. In: *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems.* 2009, pp. 205–216.

[278] D. Menasce and V. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods.* Prentice Hall, 2001. ISBN: 978-0-13065-903-3.

[279] T. Mens, P. V. Gorp, and K. Czarnecki. “A Taxonomy of Model Transformations”. In: *Proceedings of the Language Engineering for Model-Driven Software Development.* Ed. by J. Bezivin and R. Heckel. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[280] *Merriam Webster's Collegiate Dictionary.* Merriam Webster, 1993.

[281] M. Meswani, M. Laurenzano, L. Carrington, and A. Snavely. “Modeling and Predicting Disk I/O Time of HPC Applications”. In: *Proceedings of the High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC).* 2010, pp. 478–486.

[282] L. Meyer, S. Rössle, P. Bisch, and M. Mattoso. “Parallelism in Bioinformatics Workflows”. In: *Proceedings of the High Performance Computing for Computational Science (VECPAR'04).* Ed. by M. Daydé, J. Dongarra, V. Hernández, and J. Palma. Vol. 3402. Springer, 2005, pp. 583–597.

[283] R. Miceli, G. Civario, A. Sikora, E. César, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin, and F. Bodin. "AutoTune: A Plugin-Driven Approach to the Automatic Tuning of Parallel Applications". In: *Applied Parallel and Scientific Computing.* Ed. by P. Manninen and P. Öster. Vol. 7782. Springer, 2013, pp. 328–342.

[284] J.-M. Milke, M. Schiffers, and W. Ziegler. "Virtuelle Organisationen in Grids: Charakterisierung und Management". In: *Praxis der Informationsverarbeitung und Kommunikation (PIK).* Ed. by O. v. Spaniol. 2006, pp. 165–170.

[285] J. Miller and J. Mukerji. *MDA Guide Version 1.0.1.* Tech. rep. omg/2003-06-01. Object Management Group (OMG), 2003.

[286] D. Molka, D. Hackenberg, T. Minartz, R. Schöne, and W. Nagel. "Flexible Workload Generation for HPC Cluster Efficiency Benchmarking". In: *Computer Science - Research and Development* 27.4 (2012), pp. 235–243.

[287] D. Montgomery and G. Runger. *Applied Statistics and Probability for Engineers.* John Wiley & Sons, Inc., 1994. ISBN: 978-0-47154-041-0.

[288] R. Murphy. "On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance". In: *Proceedings of the 10th International IEEE Symposium on Workload Characterization (IISWC'07).* 2007, pp. 35–43.

[289] S. Murugesan. "Harnessing Green IT: Principles and Practices". In: *IT Professional* 10.1 (2008), pp. 24–33.

[290] *Nagios – The Industry Standard in IT Infrastructure Monitoring.* 2014. URL: `www.nagios.com` (visited on 08/11/2014).

[291] NASA Advanced Supercomputing Division. *NAS Parallel Benchmarks.* 2014. URL: `http://www.nas.nasa.gov/publications/npb.html` (visited on 08/19/2014).

[292] National Energy Research Scientific Computing Center (NERSC). *NERSC-8 / Trinity Benchmarks.* 2014. URL: `http://www.nersc.gov/users/computational-systems/nersc-8-system-cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks` (visited on 07/31/2014).

[293] T. Necker. "Entwicklung eines objektorientierten Werkzeugs für verschiedene Verfahren der parallelen ereignisgesteuerten Simulation - 69. Bericht über verkehrstheoretische Arbeiten". PhD thesis. Universität Stuttgart, 1998.

[294] J. von Neumann. “First Draft of a Report on the EDVAC”. In: *Annals of the History of Computing* 15.4 (1993), pp. 27–75.

[295] F. Niederman, J. C. Brancheau, and J. C. Wetherbe. “Information Systems Management Issues for the 1990s”. In: *MIS Quarterly* 15.4 (1991), pp. 475–500.

[296] O. Nierstrasz, S. Gibbs, and D. Tsichritzis. “Component-Oriented Software Development”. In: *Communications of the ACM - Special Issue on Analysis and Modeling in Software Development* 35.9 (1992), pp. 160–165.

[297] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. “PACE – A Toolset for the Performance Prediction of Parallel and Distributed Systems”. In: *Journal of High Performance Computing Applications* 14.3 (2000), pp. 228–251.

[298] D. Nurmi, J. Brevik, and R. Wolski. “Modeling Machine Availability in Enterprise and Wide-Area Distributed Computing Environments”. In: *Proceedings of the Euro-Par 2005 Parallel Processing*. Ed. by J. Cunha and P. Medeiros. Vol. 3648. Springer, 2005, pp. 432–441.

[299] D. Nussbaum and A. Agarwal. “Scalability of Parallel Machines”. In: *Communications of the ACM* 34.3 (1991), pp. 57–61.

[300] M. Nyrhinen. “IT Infrastructure: Structure, Properties and Processes”. In: *Sprouts: Working Papers on Information Systems* 6 (2006).

[301] Object Management Group (OMG). *OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1*. Tech. rep. formal/2011-08-07. Object Management Group, 2011.

[302] Object Management Group (OMG). *OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1*. Tech. rep. formal/2011-08-05. Object Management Group (OMG), 2011.

[303] Object Management Group (OMG). *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1*. Tech. rep. formal/2011-08-06. Object Management Group, 2011.

[304] E. Ogasawara, D. Oliveira, F. Chirigati, C. E. Barbosa, R. Elias, V. Braganholo, A. Coutinho, and M. Mattoso. “Exploring Many Task Computing in Scientific Workflows”. In: *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS’09)*. 2009, pp. 1–10.

[305] *On the Legal Protection of Computer Programs*. Tech. rep. Council Directive 91/250/EEC. Commission Of the European Communities (CEC), 1991.

[306] P. S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, 1997. ISBN: 9-781-55860-339-4.

[307] S. Pakin. “The Design and Implementation of a Domain-Specific Language for Network Performance Testing”. In: *IEEE Transactions on Parallel and Distributed Systems* 18.10 (2007), pp. 1436–1449.

[308] V. Pallipadi. “Enhanced Intel Speedstep Technology and Demand-Based Switching on Linux”. In: *Intel Developer Service* (2009).

[309] L. Palm. “LRZ Awarded German Data Centre Prize”. In: *Inside – Innovatives Supercomuting in Deutschland* 10.1 (2012). Ed. by R. Klank.

[310] D. F. Parkhill. *The Challenge of the Computer Utility*. Addison-Wesley, 1966. ISBN: 978-0-20105-720-1.

[311] *Partnership for Advanced Computing in Europe (PRACE)*. 2013. URL: `www.prace-project.eu` (visited on 11/30/2012).

[312] S. C. Payne. *A Guide to Security Metrics*. Tech. rep. SANS Institute, 2006.

[313] G. Pedicini and J. Green. “SPOTlight on Testing: Stability, Performance and Operational Testing of LANL HPC Clusters”. In: *Proceedings of the International ACM/IEEE Conference for High Performance Computing, Networking, Storage and Analysis (SC’11)*. 2011, pp. 1–8.

[314] M. Pedram and I. Hwang. “Power and Performance Modeling in a Virtualized Server System”. In: *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW)*. 2010, pp. 520–526.

[315] R. Petrasch and O. Meimberg. *Model Driven Architecture - Eine praxisorientierte Einführung in die MDA*. Vol. 1. dpunkt.verlag, 2006. ISBN: 978-3-89864-343-6.

[316] F. Petrini, W.-C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. “The Quadrics Network: High-Performance Clustering Technology”. In: *IEEE Micro* 22.1 (2002), pp. 46–57.

[317] F. Petrini, D. Kerbyson, and S. Pakin. “The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’03)*. 2003, pp. 55–72.

[318] F. Petrini, J. Moreira, J. Nieplocha, M. Seager, C. Stunkel, G. Thorson, P. Terry, and S. Varadarajan. “What are the Future Trends in High-Performance Interconnects for Parallel Computers”. In: *Proceedings of the 12th IEEE Symposium on High Performance Interconnects*. 2004.

[319] W. Pfeiffer and N. J. Wright. “Modeling and Predicting Application Performance on Parallel Computers Using HPC Challenge Benchmarks”. In: *Proceedings of the International IEEE Symposium on Parallel and Distributed Processing (IPDPS’08)*. 2008, pp. 1–12.

[320] G. Pfister. *In Search of Clusters*. Vol. 2. Prentice Hall, 1997. ISBN: 9-780-13899-709-0.

[321] G. Pfister. *In Search of Clusters: The Coming Battle in Lowly Parallel Computing*. Prentice-Hall, Inc., 1995. ISBN: 978-0-13437-625-7.

[322] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing*. Vol. 4. Prentice Hall, 2007.

[323] T. C. Powell and A. Dent-Micallef. “Information Technology as Competitive Advantage: the Role of Human, Business, and Technology Resources”. In: *Strategic Management Journal* 18.5 (1997), pp. 375–405.

[324] J. Pras and A. Schoenwaelder. *On the Difference between Information Models and Data Models*. Tech. rep. RfC 3444. The Internet Society, Network Working Group, 2003.

[325] J. Qin, K. Y. Chan, and P. Manneback. “Performance Analysis in Parallel Triangular Solver”. In: *Proceedings of the 2nd International IEEE Conference on Algorithms & Architectures for Parallel Processing (ICAPP’96)*. 1996, pp. 405–412.

[326] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie. *Model Driven Architecture With Executable UML*. Cambridge University Press, 2004. ISBN: 978-0-52153-771-1.

[327] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi. “Scheduling Data-Intensive Workflows onto Storage-Constrained Distributed Resources”. In: *Proceedings of the 7th International IEEE Symposium on Cluster Computing and the Grid (CCGRID’07)*. 2007, pp. 401–409.

[328] L. Ramakrishnan, K. Jackson, S. Canon, S. Cholia, and J. Shalf. “Defining Future Platform Requirements for e-Science Clouds”. In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. ACM, 2010, pp. 101–106.

[329] S. Ravelomanana, S. C. S. Bianchi, C. Joumaa, and M. Sibilla. "A Contextual Grid Monitoring by a Model Driven Approach". In: *Proceedings of the International Conference on Internet and Web Applications and Services (AICT-ICIW'06)*. 2006, pp. 37–37.

[330] D. Reed. *High-End Computing: the Challenge of Scale*. Tech. rep. Los Alamos National Laboratory (LANL), 2004.

[331] W. Reisig. *Petrinetze – Modellierungstechnik, Analysemethoden, Fallstudien*. Vieweg+Teubner Verlag, 2010. ISBN: 9-783-83489-708-4.

[332] M. Riedel, A. Streit, F. Wolf, T. Lippert, and D. Kranzlmüller. "Classification of Different Approaches for e-Science Applications in Next Generation Computing Infrastructures". In: *Proceedings of the 4th International IEEE Conference on eScience (eScience'08)*. 2008, pp. 198–205.

[333] M. Riedel, F. Wolf, D. Kranzlmüller, A. Streit, and T. Lippert. "Research Advances by Using Interoperable e-Science Infrastructures". In: *Cluster Computing* 12.4 (2009), pp. 357–372.

[334] S. Robinson. *Simulation: The Practice of Model Development and Use*. John Wiley & Sons, Ltd., 2004. ISBN: 9-780-47009-278-1.

[335] J. F. Rockart, M. J. Earl, and J. W. Ross. "Eight Imperatives for the New IT Organization". In: *Sloan Management Review* 38.1 (1996), pp. 43–54.

[336] I. Rodero, S. Chandra, M. Parashar, R. Muralidhar, H. Seshadri, and S. Poole. "Investigating the Potential of Application-Centric Aggressive Power Management for HPC Workloads". In: *Proceedings of the International Conference on High Performance Computing (HiPC)*. 2010, pp. 1–10.

[337] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. "The Structural Simulation Toolkit". In: *ACM SIGMETRICS Performance Evaluation Review - Special Issue on the 1st Intl. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems (PMBS'10)* 38.4 (2011), pp. 37–42.

[338] A. Rodrigues, R. Murphy, P. Kogge, and K. Underwood. "Characterizing a New Class of Threads in Scientific Applications for High End Supercomputers". In: *Proceedings of the 18th International Conference on Supercomputing (ICS'04)*. 2004, pp. 164–174.

[339] B. Rood and M. Lewis. “Multi-state Grid Resource Availability Characterization”. In: *Proceedings of the 8th International ACM/IEEE Conference on Grid Computing (GRID’07)*. IEEE Computer Society, 2007, pp. 42–49.

[340] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. “DRAMSim2: A Cycle Accurate Memory System Simulator”. In: *Computer Architecture Letters* 10.1 (2011), pp. 16–19.

[341] J. Ross, C. Beath, and D. Goodhu. *Reinventing the IT Organization: Final Report to the Advanced Practices Council of SIM International.* Tech. rep. Sloan School of Management, MIT, 1995.

[342] C. Rupp. *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis.* Vol. 5. München: Hanser, 2009. ISBN: 978-3-44641-841-7.

[343] C. Rupp and K. Pohl. *Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level.* Vol. 2. dpunkt.verlag, 2009. ISBN: 978-3-89864-708-3.

[344] C. Rupp, S. Queins, and B. Zengler. *UML 2 glasklar: Praxiswissen für die UML-Modellierung.* Vol. 3. Carl Hanser Verlag GmbH & CO. KG, 2007. ISBN: 9-783-44641-118-0.

[345] K. Rycerz, E. Ciepiela, G. Dyk, D. Groen, T. Gubala, D. Harezlak, M. Pawlik, J. Suter, S. Zasada, P. Coveney, and M. Bubak. “Support for Multiscale Simulations with Molecular Dynamics”. In: *Procedia Computer Science* 18 (2013), pp. 1116–1125.

[346] M. Sääksjärvi. “The Roles of Corporate IT Infrastructure and Their Impact on IS Effeciveness”. In: *Proceedings of the 8th European Conference on Information Systems (ECIS’2000).* 2000.

[347] A. Sabetta and H. Koziolek. “Measuring Performance Metrics: Techniques and Tools”. In: *Dependability Metrics.* Ed. by I. Eusgeld, F. C. Freiling, and R. Reussner. Vol. 4909. Springer, 2008, pp. 226–232.

[348] R. Sabherwal and Y. E. Chan. “Alignment Between Business and IS Strategies: A Study of Prospectors, Analyzers, and Defenders”. In: *Information Systems Research* 12.1 (2001), pp. 11–33.

[349] S. Saini, D. Talcott, D. Jespersen, J. Djomehri, H. Jin, and R. Biswas. “Scientific Application-based Performance Comparison of SGI Altix 4700, IBM POWER5+, and SGI ICE 8200 Supercomputers”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’08).* 2008, pp. 1–12.

[350] T. Sato. “Can the Earth Simulator Change the Way Humans Think?” In: *Proceedings of the 16th International Conference on Supercomputing (ICS’02)*. 2002, pp. 1–1.

[351] T. Sato, S. Kitawaki, and M. Yokokawa. “Earth Simulator Running”. In: *Proceedings of the Proceedings of ISC*. 2002, pp. 20–22.

[352] R. Savola. “Towards a Security Metrics Taxonomy for the Information and Communication Technology Industry”. In: *Proceedings of the International IEEE Conference on Software Engineering Advances (ICSEA’07)*. 2007.

[353] M. Schiffers. “Management dynamischer Virtueller Organisationen in Grids”. PhD thesis. Fakultät für Mathematik, Informatik und Statistik der Ludwig-Maximilians-Universität München, 2007.

[354] R. Schöne and D. Hackenberg. “On-Line Analysis of Hardware Performance Events for Workload Characterization and Processor Frequency Scaling Decisions”. In: *Proceedings of the 2nd International ACM/SPEC Conference on Performance Engineering (ICPE’11)*. 2011, pp. 481–486.

[355] U. Schöning. *Theoretische Informatik - kurzgefasst*. Vol. 4. Spektrum Akademischer Verlag, 2008. ISBN: 978-3-82741-824-1.

[356] M. Schulte-Zurhausen. *Organisation*. Vol. 4. Vahlen Franz GmbH, 2005. ISBN: 978-3-80063-205-3.

[357] B. Selic. “The Pragmatics of Model-Driven Development”. In: *IEEE Software* 20.5 (2003), pp. 19–25.

[358] P. Senkul, M. Kifer, and I. Toroslu. “A Logical Framework for Scheduling Workflows Under Resource Allocation Constraints”. In: *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB’02)*. 2002, pp. 694–705.

[359] P. Senkul and I. Toroslu. “An Architecture for Workflow Scheduling under Resource Allocation Constraints”. In: *Information Systems* 30.5 (2005), pp. 399–422.

[360] H. Shan, K. Antypas, and J. Shalf. “Characterizing and Predicting the I/O Performance of HPC Applications Using a Parameterized Synthetic Benchmark”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’08)*. 2008, pp. 1–12.

[361] S. Shang and P. B. Seddon. “Assessing and Managing the Benefits of Enterprise Systems: the Business Manager’s Perspective”. In: *Information Systems Journal* 12.4 (2002), pp. 271–299.

[362] I. Sharapov, R. Kroeger, G. Delamarter, R. Cheveresan, and M. Ramsay. “A Case Study in Top-Down Performance Estimation for a Large-Scale Parallel Application”. In: *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP’06).* ACM, 2006, pp. 81–89. ISBN: 1-59593-189-9.

[363] S. Sharma, C.-H. Hsu, and W.-C. Feng. “Making a Case for a Green500 List”. In: *Proceedings of the Parallel and Distributed Processing Symposium (IPDPS’06).* 2006, pp. 1–8.

[364] S. K. Sia, C. Soh, and P. Weill. “Global IT Management: Structuring for Scale, Responsiveness, and Innovation”. In: *Proceedings of the 3rd ACM Communications of the ACM* 53.3 (2010), pp. 59–64.

[365] M. Silver, L. Markus, and C. M. Beath. “The Information Technology Interaction Model: A Foundation for the MBA Core Course”. In: *MIS Quarterly* 19.3 (1995), pp. 361–390.

[366] Y. Simmhan and L. Ramakrishnan. “Comparison of Resource Platform Selection Approaches for Scientific Workflows”. In: *Proceedings of the 19th International ACM Symposium on High Performance Distributed Computing.* ACM, 2010, pp. 445–450.

[367] H. A. Simon. *The Sciences Of The Artificial.* Vol. 2. 1981. ISBN: 978-0-26226-449-5.

[368] H. Simon. *The Sciences Of The Artificial.* Vol. 3. MIT Press, 1996. ISBN: 978-0-26226-449-5.

[369] M. Singh. “Synthesizing Distributed Constrained Events from Transactional Workflow Specifications”. In: *Proceedings of the 12th International Conference on Data Engineering.* 1996, pp. 616–623.

[370] M. Singh. “Semantical Considerations on Workflows: An Algebra for Intertask Dependencies”. In: *Proceedings of the International Workshop on Database Programming Languages,* 1995, pp. 1–17.

[371] A. Sinha and A. P. Chandrakasan. “Dynamic Voltage Scheduling using Adaptive Filtering of Workload Traces”. In: *Proceedings of the 14th International Conference on VLSI Design.* 2001, pp. 221–226.

[372] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. “A Framework for Performance Modeling and Prediction”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’02).* 2002, pp. 21–21.

[373] A. Snavely, N. Wolter, and L. Carrington. "Modeling Application Performance by Convolving Machine Signatures with Application Profiles". In: *Proceedings of the International IEEE Workshop on Workload Characterization*. 2001, pp. 149–156.

[374] Software Engineering Standards Committee (SESC). *IEEE Recommended Practice for Software Requirements Specifications*. Tech. rep. IEEE Std 830-1998. Institute of Electrical and Electronics Engineerings, Inc. (IEEE), 1998.

[375] K. Specht, R. Bulander, and W. Gohout. *Statistik für Wirtschaft und Technik*. Oldenbourg Verlag, 2012. ISBN: 978-3-48671-356-5.

[376] T. Stahl and M. Völter. *Model-Driven Software Development – Technology, Engineering, Management*. John Wiley & Sons, Inc., 2006. ISBN: 978-0-470-02570-3.

[377] C. Stiller. "Grundbegriffe der Messtechnik". In: *Grundlagen der Mess- und Regelungstechnik*. Ed. by C. Stiller. Shaker Verlag, 2006, pp. 113–128.

[378] P. A. Strassmann. *The Business Value of Computers*. Vol. 1. The Infomation Economic Press, 1990. ISBN: 9-780-96204-132-7.

[379] C. Straube. "DAGA – Active Probing zur Bestimmung der Verfügbarkeit von Grid-Ressourcen". Diploma thesis. Ludwig-Maximilians-Universität München, 2011.

[380] C. Straube and D. Kranzlmüller. "A Generic Capability Model for Analyzing Modification Effects in HPC Infrastructures". Poster presentation at the 22nd International ACM Symposium on High Performance Parallel and Distributed Computing (HPDC'13). 2013.

[381] *System Configuration Details*. 2014. URL: `http://www.lrz.de/services/compute/supermuc/systemdescription/` (visited on 08/15/2014).

[382] *System Security Engineering - Capability Maturity Model – Model Description Document*. Tech. rep. 3.0. Carnegie Mellon University, 2003.

[383] T. Talbot and H. Davis. *Verizon NEBS TM Compliance: Energy Efficiency Requirements for Telecommunications Equipment*. Tech. rep. VZ.TPR.9205. Verizon, 2011.

[384] A. S. Tanenbaum. *Computerarchitektur - Strukturen - Konzepte - Grundlagen*. Vol. 5. Addison-Wesley, 2005. ISBN: 978-3-82737-151-5.

[385] A. Tanenbaum and M. V. Steen. *Verteilte Systeme - Prinzipien Und Paradigmen*. Vol. 2. Addison-Wesley, 2007. ISBN: 978-3-82737-293-2.

[386] J. Taylor. *Enhanced-Science (e-Science) Definition*. 2013. URL: `www.e-science.clrc.ac.uk` (visited on 03/20/2013).

[387] *The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range*. Tech. rep. UCRL-5375. Lawrence Livermore National Laboratory, 1986.

[388] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya. "DCSim: A Data Centre Simulation tool for Evaluating Dynamic Virtualized Resource Management". In: *Proceedings of the 8th International Conference and Workshop on Systems Virtualization Management (SVM), Network and Service Management (CNSM)*. 2012, pp. 385–392.

[389] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely. "Modeling Power and Energy Usage of HPC Kernels". In: *Proceedings of the 26th International IEEE Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW'12)*. 2012, pp. 990–998.

[390] V. Tosic and S. Dordevic-Kajan. "The Common Information Model (CIM) Standard – An Analysis of Features and Open Issues". In: *Proceedings of the 4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services (TELSIKS'99)*. 1999, pp. 677–680.

[391] K. Trivedi. *Probability and Statistics with Reliability Queuing and Computer Science Applications*. Vol. 2. John Wiley & Sons, Inc., 2001. ISBN: 978-0-47133-341-8.

[392] D. Tsichritzis. "Beyond Calculation". In: *Beyond Calculation: The Next Fifty Years of Computing*. Ed. by P. Denning and R. Metcalfe. 1997, pp. 259–265.

[393] P. Turnbull. "Effective Investment in Information Infrastructures". In: *Information and Software Technology* 33.3 (1991), pp. 191–199.

[394] *Twenty-One Experts Define Cloud Computing*. 2014. URL: `http://cloudcomputing.sys-con.com/node/612375/print` (visited on 08/11/2014).

[395] A. Umar. "IT Infrastructure to Enable Next Generation Enterprises". In: *Information Systems Frontiers* 7.3 (2005), pp. 217–256.

[396] E. Upton and G. Halfacree. *Raspberry Pi User Guide*. Wiley, 2013. ISBN: 978-1-11879-546-0.

[397] U. Valentini, R. Weißbach, R. Fahney, T. Gartung, J. Glunde, A. Herrmann, A. Hoffmann, and E. Knauss. *Requirements Engineering und Projektmanagement.* Springer, 2013. ISBN: 9-783-64229-432-7.

[398] A. Varga. "The OMNeT++ Discrete Event Simulation System". In: *Proceedings of the European Simulation Multiconference (ESM'01).* 2001, pp. 185–192.

[399] N. Venkatraman. "IT-Enabled Business Transformation from Automation to Business Scope Redefinition". In: *Sloan Management Review* 35.2 (1994).

[400] S. Verbrugge, D. Colle, P. Demeester, R. Huelsermann, and M. Jaeger. "General Availability Model for Multilayer Transport Networks". In: *Proceedings of the 5th International Workshop on Design of Reliable Communication Networks (DRCN'05).* IEEE Computer Society, 2005, pp. 1–8.

[401] A. Voss, M. Mascord, M. Fraser, M. Jirotka, R. Procter, P. Halfpenny, D. Fergusson, M. Atkinson, S. Dunn, and T. B. and. "e-Research Infrastructure Development and Community Engagement". In: *Proceedings of the UK e-Science All Hands Meeting.* 2007.

[402] L. Wang and S. U. Khan. "Review of Performance Metrics for Green Data Centers: a Taxonomy Study". In: *The Journal of Supercomputing* 63.3 (2011), pp. 639–656.

[403] S. C. Ward. "Arguments for Constructively Simple Models". In: *Journal of the Operational Research Society* 40.2 (1989), pp. 141–153.

[404] T. Warner. "Information Technology as Competitive Burden". In: *Sloan Management Review* 29.1 (1987), pp. 55–61.

[405] G. Wasson and M. Humphrey. "Policy And Enforcement In Virtual Organizations". In: *Proceedings of the 4th International Workshop on Grid Computing (GRID'03).* 2003, pp. 125–132.

[406] J. Watt, O. Ajayi, J. Jiang, J. Koetsier, and R. O. Sinnott. "A Shibboleth-Protected Privilege Management Infrastructure for e-Science Education". In: *Proceedings of the 6th International IEEE Symposium on Cluster Computing and the Grid (CCGRID'06).* 2006, pp. 356–364.

[407] R. P. Weicker. "An Overview Of Common Benchmarks". In: *Computer* 23.12 (1990), pp. 65–75.

[408] R. Weicker. "A Detailed Look At Some Popular Benchmarks". In: *Parallel Computing* 17.10 (1991), pp. 1153–1172.

[409] R. Weicker. “Dhrystone: A Synthetic Systems Programming Benchmark”. In: *Communications of the ACM* 27.10 (1984), pp. 1013–1030.

[410] P. Weill. *The Role and Value of Information Technology Infrastructure: Some Empirical Observations.* Tech. rep. Sloan WP No. 3433-92. Massachusetts Institute of Technology (MIT), Sloan School of Management, 1992.

[411] P. Weill, M. Subramani, and M. Broadbent. *IT Infrastructure for Strategic Agility.* Tech. rep. Massachusetts Institute of Technology (MIT), Sloan School of Management, 2002.

[412] J. Weinberg, M. McCracken, E. Strohmaier, and A. Snavely. “Quantifying Locality In The Memory Access Patterns of HPC Applications”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC’05).* 2005, pp. 50–60.

[413] M. Weiser, B. Welch, A. Demers, and S. Shenker. “Scheduling for Reduced CPU Energy”. In: *Mobile Computing.* Ed. by T. Imielinski and H. Korth. Vol. 353. Springer, 1996, pp. 449–471.

[414] B. Wichmann. *Validation Code for the Whetstone Benchmark.* Tech. rep. NPL-DITC 107/8. National Physical Laboratory, 1988.

[415] M. Wieczorek, A. Hoheisel, and R. Prodan. “Taxonomies of the Multi-Criteria Grid Workflow Scheduling Problem”. In: *Proceedings of the Grid Middleware and Services.* Springer, 2008, pp. 237–264.

[416] M. Wieczorek, R. Prodan, and T. Fahringer. “Comparison of Workflow Scheduling Strategies on the Grid”. In: *Proceedings of the Parallel Processing and Applied Mathematics.* Vol. 3911. Springer, 2006, pp. 792–800.

[417] T. Wilde, A. Auweter, and H. Shoukourian. “The 4 Pillar Framework for Energy Efficient HPC Data Centers”. In: *Computer Science - Research and Development* (2013), pp. 1–11.

[418] B. Wilkinson. *Grid Computing - Techniques and Applications.* Chapman & Hall/CRC Computational Science, 2009. ISBN: 978-1-42006-954-9.

[419] T. Willemain. “Insights on Modeling from a Dozen Experts”. In: *Operations Research* 42.2 (1994), pp. 213–222.

[420] D. Woollard, N. Medvidovic, Y. Gil, and C. A. Mattmann. “Scientific Software as Workflows: From Discovery to Distribution”. In: *IEEE Software* 25.4 (2008), pp. 37–43.

[421] W. Xia and W. R. King. *Determinants of Organizational IT Infrastructure Capabilities: An Empirical Study*. Tech. rep. Carlson School of Management, University of Minnesota, 2002.

[422] J. Xu, Z. Kalbarczyk, and R. K. Iyer. "Networked Windows NT System Field Failure Data Analysis". In: *Proceedings of the International Pacific Rim Symposium on Dependable Computing*. 1999, pp. 178–185.

[423] L. Xue, G. Ray, and B. Gu. "Environmental Uncertainty and IT Infrastructure Governance: A Curvilinear Relationship". In: *Information Systems Research (INFORMS)* 22.2 (2011), pp. 389–399.

[424] T. Yang, X. Ma, and F. Mueller. "Predicting Parallel Applications' Performance Across Platforms Using Partial Execution". In: *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'02)*. 2002.

[425] Y. Yang, X. Zhang, and Y. Song. "An Effective and Practical Performance Prediction Model for Parallel Computing on Nondedicated Heterogeneous NOW". In: *Journal of Parallel and Distributed Computing* 38.1 (1996), pp. 63–80.

[426] R. K. Yin. *Case Study Research: Design and Methods*. Vol. 4. SAGE Publications, 2008.

[427] M. Yokokawa. "Present Status of Development of the Earth Simulator". In: *Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems*. 2001, pp. 93–99.

[428] J. Yu, R. Buyya, and C. K. Tham. "Cost-Based Scheduling of Scientific Workflow Applications on Utility Grids". In: *Proceedings of the 1st International Conference on e-Science and Grid Computing*. 2005, pp. 139–147.

[429] J. Yu and R. Buyya. "A Taxonomy of Scientific Workflow Management Systems for Grid Computing". In: *Journal of Grid Computing* 3.3-4 (2005), pp. 171–200.

[430] J. Yu, R. Buyya, and K. Ramamohanarao. "Workflow Scheduling Algorithms for Grid Computing". In: *Proceedings of the Metaheuristics for Scheduling in Distributed Computing Environments*. Ed. by F. Xhafa and A. Abraham. Vol. 146. Springer, 2008, pp. 173–214.

[431] S. Zanikolas and R. Sakellariou. "A Taxonomy of Grid Monitoring Systems". In: *Future Generation Computer Systems* 21.1 (2005), pp. 163–188.

[432] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. “Modeling Hard-Disk Power Consumption”. In: *Proceedings of the 2nd Conference on File and Storage Technologies (FAST’03)*. 2003, pp. 217–230.

[433] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation – Integrating Discrete Event and Continous Complex Dynamics Systems*. Vol. 2. Academic Press, 2000. ISBN: 9-780-12778-455-7.

[434] M. V. Zelkowitz and D. R. Wallace. “Experimental Models for Validating Technology”. In: *IEEE Computer* 31.5 (1998), pp. 23–31.

[435] M. Zelkowitz. “An Update to Experimental Models for Validating Computer Technology”. In: *Journal of Systems and Software* 82.3 (2009), pp. 373–376.

[436] Q. Zhang, L. Cheng, and R. Boutaba. “Cloud Computing: State-of-the-Art and Research Challenges”. In: *Journal of Internet Services and Applications* 1.1 (2010), pp. 7–18.

[437] H. Zhao and R. Sakellariou. “Advance Reservation Policies for Workflows”. In: *Proceedings of the Job Scheduling Strategies for Parallel Processing*. Ed. by E. Frachtenberg. Vol. 4376. Springer, 2007, pp. 47–67.

[438] Y. Zhao, I. Raicu, and I. Foster. “Scientific Workflow Systems for 21st Century, New Bottle or New Wine?” In: *Proceedings of the IEEE Congress on Services - Part I*. 2008, pp. 467–471.

[439] G. Zheng, G. Gupta, E. Bohm, I. Dooley, and L. V. Kale. “Simulating Large Scale Parallel Applications Using Statistical Models for Sequential Execution Blocks”. In: *Proceedings of the 16th International IEEE Conference on Parallel and Distributed Systems (ICPADS’10)*. 2010, pp. 221–228.

[440] G. Zheng, A. K. Singla, J. M. Unger, and L. Kalé. “A Parallel-Object Programming Model for Petaflops Machines and Blue Gene/cyclops”. In: *Proceedings of the International IEEE Parallel and Distributed Processing Symposium*. 2002, pp. 8–16.

[441] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. Kalé. “Simulation-Based Performance Prediction for Large Parallel Machines”. In: *Journal of Parallel Programming* 33.2-3 (2005), pp. 183–207.

Abbreviations and index

ASCI	Advanced Simulation and Computing
BLAS	Basic Linear Algebra Subroutines
CDN	Content Delivery Networks
CFD	Computational Fluid Dynamics
CIM	Common Information Model
CMB	Cosmic Microwave Background
CMDB	Configuration Management Database
COTS	Commodity-off-the-shelf
CPU	Central Processing Unit
CUE	Carbon Usage Effectiveness
DMTF	Distributed Management Task Force
DOE	Department of Energy
DRIHM	Distributed Research Infrastructure for Hydro Meteorology
DVFS	Dynamic Voltage and Frequency Scheduling
EBNF	Extended Backus Naur Form
EGEE	Enabling Grids for e-Science
FLOP	Floating Point Operation
GLUE	Grid Laboratory Uniform Environment
GPFS	General Parallel File System
GPU	Graphics Processing Unit
HM	Hydro Meteorology
HPC	High Performance Computing

HPL	High Performance LINPACK
IT	Information Technology
JSDL	Job Submission Description Language
KPI	Key Performance Indicators
LANL	Los Alamos National Laboratory
LRZ	Leibniz Supercomputing Center
MADCAP	Microwave Anisotropy Dataset Computational Analysis Package
MDA	Model Driven Architecture
MDD	Model Driven Development
MOF	Meta Object Facility
MPI	Message Passing Interface
NAS	Network Attached Storage
NASA	National Aeronautics and Space Administration
NGI	National Grid Initiatives
OGF	Open Grid Forum
ORNL	Oak Ridge National Laboratory
PAPI	Performance Application Programming Interface
PERCS	Productive, Easy-to-use, Reliable Computing System
PIM	Platform Independent Model
PRACE	Partnership for Advanced Computing in Europe
	Platform Specific Implementation
PSM	Platform Specific Model
PUE	Power Usage Efficiency
QPI	Quick Path Interconnect
RAPL	Running Average Power Limit
RDL	Resource Description Language
RPi	Raspberry Pi
RS	Requirements Specification
SNMP	Simple Network Management Model
SLA	Service Level Agreement
SGI	Silicon Graphics
TOC	Total Cost of Ownership
UML	Unified Modeling Language