Dissertation

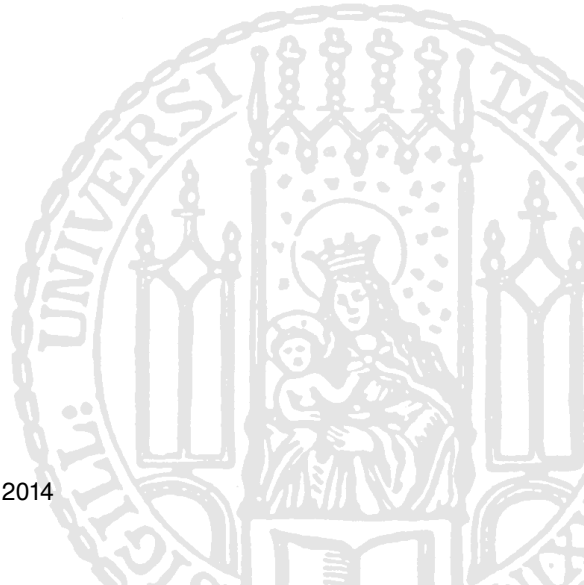
an der

FAKULTÄT FÜR MATHEMATIK,
INFORMATIK UND STATISTIK

DER LUDWIG--MAXIMILIANS--UNIVERSITÄT MÜNCHEN

Martin Gerhard Metzker

A network QoS management architecture for virtualization environments



Eingereicht am: 28. Oktober 2014

Dissertation

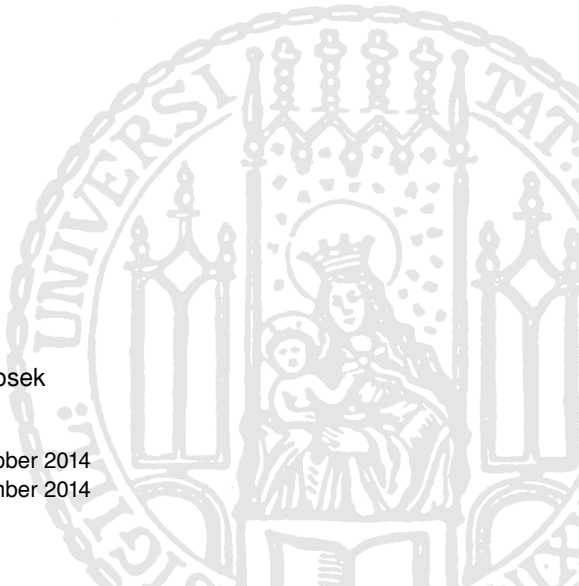
an der

FAKULTÄT FÜR MATHEMATIK,
INFORMATIK UND STATISTIK

DER LUDWIG--MAXIMILIANS--UNIVERSITÄT MÜNCHEN

Martin Gerhard Metzker

A network QoS management architecture for virtualization environments



1. Berichterstatter:
Prof. Dr. Dieter Kranzlmüller

2. Berichterstatter:
Prof. Dr. Gabrijela Dreo Rodosek

Eingereicht am: 28. Oktober 2014
Tag der Disputation: 8. Dezember 2014

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

Network quality of service (QoS) and its management are concerned with providing, guaranteeing and reporting properties of data flows within computer networks. For the past two decades, virtualization has been becoming a very popular tool in data centres, yet, without network QoS management capabilities.

With virtualization, the management focus shifts from physical components and topologies, towards virtual infrastructures (VI) and their purposes. VIs are designed and managed as independent isolated entities. Without network QoS management capabilities, VIs cannot offer the same services and service levels as physical infrastructures can, leaving VIs at a disadvantage with respect to applicability and efficiency.

This thesis closes this gap and develops a management architecture, enabling network QoS management in virtulization environments. First, requirements are dervied, based on real world scenarios, yielding a validation reference for the proposed architecture. After that, a life cycle for VIs and a taxonomy for network links and virtual components are introduced, to arrange the network QoS management task with the general management of virtualization environments and enabling the creation of technology specific adaptors for integrating the technologies and sub-services used in virtualization environments. The core aspect, shaping the proposed management architecture, is a management loop and its corresponding strategy for identifying and ordering sub-tasks.

Finally, a prototypical implementation showcases that the presented management approach is suited for network QoS management and enforcement in virtualization environments. The architecture fulfils its purpose, fulfilling all identified requirements. Ultimately, network QoS management is one amongst many aspects to management in virtualization environments and the herin presented architecture shows interfaces to other management areas, where integration is left as future work.

Kurzreferat

Die Verwaltungsaufgaben für Netzdienstgüte umfassen das Bereitstellen, Sichern und Berichten von Flusseigenschaften in Rechnernetzen. Während der letzen zwei Jahrzehnte entwickelte sich Virtualisierung zu einer Schlüsseltechnologie für Rechenzentren, bisher ohne Möglichkeiten zum Verwalten der Netzdienstgüte.

Der Einsatz von Virtualisierung verschiebt den Fokus beim Betrieb von Rechenzentren weg von physischen Komponenten und Netzen, hin zu virtuellen Infrastrukturen (VI) und ihren Einsatzzwecken. VIs werden als unabhängige, voneinander isolierte Einheiten entwickelt und verwaltet. Ohne Netzdienstgüte, sind VIs nicht so vielseitig und effizient einsetzbar wie physische Aufbauten. Diese Arbeit schließt diese Lücke mit der Entwicklung einer Managementarchitektur zur Verwaltung der Netzdienstgüte in Virtualisierungsumgebungen.

Zunächst werden Anforderungen aus realen Szenarios abgeleitet, mit denen Architekturen bewertet werden können. Zur Abgrenzung der speziellen Aufgabe Netzdienstgüteverwaltung innerhalb des allgemeinen Managementproblems, wird anschließend ein Lebenszyklusmodell für VIs vorgestellt. Die Entwicklung einer Taxonomie für Kopplungen und Komponenten ermöglicht technologiespezifische Adaptoren zur Integration von in Virtualisierungsumgebungen eingesetzten Technologien. Kerngedanke hinter der entwickelten Architektur ist eine Rückkopplungsschleife und ihre einhergehende Methode zur Strukturierung und Anordnung von Teilproblemen.

Abschließend zeigt eine prototypische Implementierung, dass dieser Ansatz für Verwaltung und Durchsetzung von Netzdienstgüte in Virtualisierungsumgebungen geeignet ist. Die Architektur kann ihren Zweck sowie die gestellten Anforderungen erfüllen. Schlussendlich ist Netzdienstgüte ein Bereich von vielen beim Betrieb von Virtualisierungsumgebungen. Die Architektur zeigt Schnittstellen zu anderen Bereichen auf, deren Integration zukünftigen Arbeiten überlassen bleibt.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Network quality of service (QoS) and its management are concerned with providing, guaranteeing and reporting properties of data flows within computer networks. Network QoS directly affects the performance of software applications, that depend on communication. Through this relation, service consumers and operators have a direct interest defining and achieving of network QoS. Its visibility and importance has been growing, as software architectures grow increasingly distributed [FHTG 10]. In order to guarantee performance aspects of applications, performing network QoS management is substantial, especially in environments where resources for achieving network QoS are sparse and contested. Data centres employing virtualization often create such environments.

Virtualization environments (VE) are typically used to provide isolated interconnected servers for individual, customer managed service instances. With virtualization, virtual machines (VM) are used instead of physical computers, to provision service instances and service components. One of the most eye-catching advantages of virtualization over traditional dedicated computers are multiple concurrently active VMs, sharing the resources of few physical hosts. Exploiting this advantage means multiple service components share the same network uplinks, leading to an environment where network resources are sparse and contested.

To facilitate network traffic forwarding, there are virtual network components, sharing the physical resources of hosts with VMs. To achieve network QoS, virtual network components require specific types and amounts of resources, which must be provided by virtualization hosts. Models describing network QoS are available and today's virtualization technologies are capable of performing suited resource allocations. The missing pieces to network QoS management in VEs are

- concepts for mapping network QoS to resource requirements of virtual components,
- coordination of management efforts concerning virtual components, virtual topologies and the physical infrastructure, and
- a management approach for network QoS accounting for unique properties of VEs.

This work develops an architecture for network QoS management in VEs, by providing the missing pieces and combining them with existing approaches. New and challenges specific to VEs are analysed for their effects on network QoS and implications on its management. The findings shape the resulting management architecture, so that corresponding management systems can provide network QoS for networks including virtual components and links.

1.1. Virtualization is indirection

Virtualization technologies often create abstracted resource pools from the physical infrastructure. VMs and virtual networks (VN) are allocated shares from this pool as needed. In a model view of virtualization, the physical hardware is consolidated into a single black box of resources, accessed only through a virtualization layer. This layer's task is mapping virtual components to physical resources.

Virtual components are the hardware provided by the virtualization layer and occupied by operating systems (OS) of virtual systems, for instance

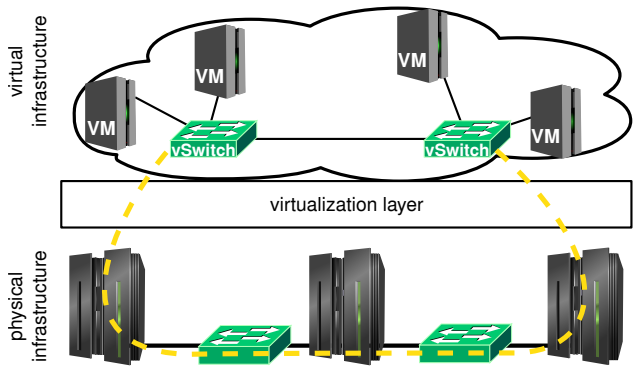


Figure 1.1.: A VI realised on top of a physical infrastructures

VMs. The manner of how the virtualization layer provides virtual components and the concrete choice of physical resources for the provisioning can be changes any time, by configuring the virtualization layer. Through this, the relation between an OS and its physical resources is dynamic, while it is static in traditional infrastructures. Usually the mapping from virtual components to physical resources is transparent for the OS and it cannot influence the mapping either.

Figure 1.1 shows a virtual infrastructure (VI), a set of networked VMs and virtual network components that must be implemented isolated from other VIs. The VI in Figure 1.1 consists of two virtual switches (labelled vSwitch), interconnecting four VMs. The VI is active, which means all its components are active and a network facilitating communication paths as specified by VI links has been realised. When a virtual system is activated, it is *placed* on a physical host by the virtualization layer. This means the VM is using the host's resources in the form of virtual components to power its OS. Placing a VI means placing all VMs and network components and creating an isolated network.

When exchanging data frames, the two vSwitches inside the virtual infras-

tructure are peer entities, but the effective data flow depends on their placement. The dashed line in Figure 1.1 depicts a possible data flow, where the data frames cross through the entire physical infrastructure. This example illustrates the main challenge to network QoS through virtualization: The introduction of the virtualization layer is an indirection, creating a gap between physical resources and a services. The gap results in more flexibility for assigning and using physical resources on the one hand and uncertainty concerning effective data flows and placement of service instances on the other hand.

Resource allocations are a central aspect of achieving network QoS. While virtualization aims to be completely transparent to components and services, it may not be transparent for QoS considerations. For robust QoS management in virtualization environments virtual systems, the virtualization layer, the physical hosts and their interconnect must be managed as part of the same management task. Current network QoS management approaches do not include the specifics of VEs, such as the presence of a virtualization layer. Therefore, QoS in a set-up including virtual and physical components cannot be guaranteed.

1.2. Challenging example

Consider a twisted pair network cable as it is used in today's Ethernet infrastructures. In a physical infrastructure, when used to connect two computers, the resulting infrastructure is trivially described as: two endpoints directly connected by a cable. In this set-up, properties of network connections depend on the capacity of the cable and the endpoints. All three elements are static, thus their dependencies can be gauged once and considered known. The endpoints' capacities can be divided up into shares and assigned to various connections. Quality of service considerations in this example concern endpoints only, as the cable is a passive component and does not perform actions that may have side effects on network QoS.

The virtual counterpart of a twisted pair Ethernet cable is a piece of software provided by the virtualization layer. It's concrete implementation

depends on the VMs' placement. If both endpoints are placed on the same host, QoS considerations boil down to moving data from one software container to another. If the endpoints are placed on different hosts, the virtual cable is a complex component, where data is processed, sent over the physical network to another host and reprocessed to be delivered to the other endpoint. Determining the capacity of this virtual cable is much more complex than determining the capacity of a physical Ethernet cable and QoS considerations concern many virtual and physical components.

Additionally, there are dynamic aspects to consider:

- VMs can be paused and placed on a completely different host.
- Other virtual components competing for resources are created or withdrawn on demand.
- The VI is changed and the cable must become a switch.

An advantage of virtualization over traditional infrastructures is that it allows hardware maintenance and load balancing without service interruptions. This advantage is gained from the ability to move VMs between physical hosts at runtime. In this example, when a VM is *migrated* between physical hosts, the software container is moved and the virtual cable must be implemented differently, so that still connects the virtual endpoints.

Through migration and on demand placement of VIs in VEs, resource allocations and usage changes constantly. Given an initial configuration realising network QoS was found, it is possible for later changes to other VIs, to have negative side effects on network QoS.

A change at runtime to the example VI may also include adding a third VM to the infrastructure, thus turning the virtual cable into a virtual switch. Turning an originally passive ISO OSI layer 1 component, a cable, into an active ISO OSI layer 2 component, a switch, is a noticeable change, also affecting management and especially for QoS considerations. Flexible topologies are another advantage of virtualization over traditional infrastructure and are frequently encountered use cases in management scenarios where virtualization is employed.

Furthermore, access to virtual infrastructures is mostly gained through IP routing, resulting in many virtual layer 3 components and virtual layer 3 endpoints in the internet. These use cases tie dynamic changes in virtual component placement to internet routing and QoS properties. Moving endpoints and changing network topologies require highly refined management capabilities to sustain QoS levels before, during and after these activities.

The implementation of virtual components, the dynamic changes in VEs and the translation of QoS requirements and configurations between different management views are the three main challenges to a management architecture for network QoS management.

1.3. Problem analysis

May it be VMs, VNs or other virtual resources, virtualization always provides abstraction from physical set-ups. Ideally, virtualized systems are not aware of this and operating systems using virtual resource need not be adapted to this new environment. Even though VMs and components are intended to offer the same capabilities and act in the same manner as their physical counterparts, their management is different, as there the virtualization layer and its capabilities must always be considered.

Virtualization follows the idea of a multiplex: offering multiple logical instances of a resource, sharing limited physical resources and capacities. For example, a VM is a logical instance of a computer. Its characterising components, CPU, memory (RAM), and storage (HDD) and network access, usually in the form of an Ethernet network interface card (NIC). These components are mapped to resource shares of the underlying physical computer (host). To provide VMs, a *virtual machine monitor* (VMM) or *hypervisor* takes the place of the host's operating system. The VMM implements the virtualization layer and provides virtual hardware, the VM, on which guest operating systems are run.

Figure 1.2 illustrates a virtualization multiplex with two VMs and a virtualization host. Each VM has an individual logical instance of a CPU and the

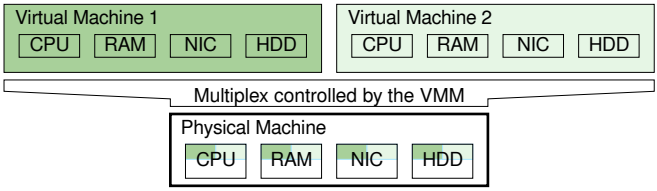


Figure 1.2.: A VMM multiplexing physical resources

multiplex performed by the virtualization layer is a time division multiplex, where a VM is given time intervals in which the physical CPU will execute actual architecture specific instructions from the logical CPUs instruction stream. Beginning and ending these execution intervals and switching between VMs is all part of the VMM's tasks. Access to RAM, HDD and NIC is similarly coordinated by the VMM, but with different techniques.

For network access, VMMs usually implement a switch, in order to bridge virtual NICs and physical networks. This exceeds the basic concept of a multiplex, but the general notion of all virtual components sharing limited physical resources remains and must always be accounted for.

Communication processes are structured and described by the *system*, *service* and *protocol* interfaces [HAN 99]. The interfaces characterise the interaction of entities in a vendor and implementation independent manner. The virtualization layer multiplex describes a provider consumer relationship between virtual components and providing infrastructure, best described as a service interface. Yet, the behaviour, functionality and purpose of virtual entities, such as VMs and virtual switches, indicates the system interface must be used to correctly describe the interaction between virtual and physical components. Which one is more adequate, depends on the current placement and implementation of components and is therefore subject to change. This makes VEs problematic for network management and especially QoS management.

In order to perform network QoS management, the interface and corresponding communication determines how network QoS is implemented:

`System interface`: All depicted computers and network components are seen as discrete systems. Applying this interface, virtual and physical systems are peer entities on the same subnet and treated equally by network QoS management. Realising network QoS concerns *horizontal communication* for which many QoS approaches exist and have been implemented.

`Service interface`: Virtual components use the underlying network transparently, regardless of its topology and attributes. For such *vertical communication* QoS can only be realised if corresponding service primitives exist, which is not the case in today's VEs.

Assuming the virtual infrastructure in Figure 1.1 is entirely placed on exactly one of the three physical hosts, communication between the virtual systems is entirely implemented by the VMM and data frames never leave the physical host. VMM and host can be seen as a monolithic platform and the service interface best describes the relation between VI and providing platform. If a VI is spread across all physical hosts, the system interface is more accurate, as network traffic crosses many discrete systems. A complete VI placement description consists of two components, one based for each interface type.

In addition to the management problems of strictly physical set-ups, virtualization employs multiple abstracted views on infrastructures, showing more or less details about component realisation and placement. The amount of included information depends on the operators' roles, i.e., goals and tasks. When performing management tasks, the customers' concerns are their services and the VIs implementing them. The concrete implementations of virtual components, especially physical hosts, the virtualization layer and the supporting physical infrastructure are not directly managed by customers, thus excluded from their management views.

Figure 1.3 shows a VE with two coexisting VI. Each VI belongs to a different customer, resulting in two customer views, limited to one or the other VI. Through the virtualization multiplex, management actions by different

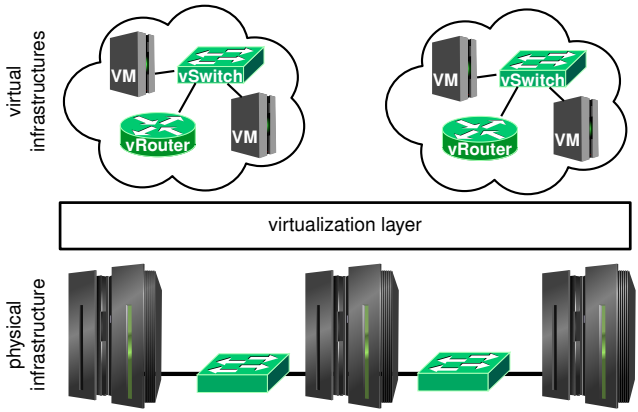


Figure 1.3.: Two coexisting virtual infrastructures realised using the same physical resources.

customers may interfere with each other. For network QoS management, all performed management must be coordinated and translated to tasks and configurations for the actual physical infrastructure.

Virtualization has many legitimate techniques to implement virtual components and infrastructures. Section 1.2 discusses the implementation of a single virtual Ethernet cable which turns out distinctly more complex, compared to the simple nature of its physical role model. With an available pool of implementation approaches, developing a network QoS architecture for virtualization environments largely consists of orchestrating virtualization technologies and providing technology agnostic management.

1.4. Problem statement

When providing networks and connections with a requested QoS, resources have to be allocated to meet all demands. An accurate model of all vir-

tual and physical components and their relations allows to identify which resources have to be allocated. By controlling and coordinating the dynamics in VEs, quality assurances can be maintained by reacting to changes and adopting resource allocations accordingly. With the high level of management views offered to operators of VIs, QoS configurations and requirements must be translated into requirements and allocations that can be implemented by the hosting components. The main thesis is that network quality of service in VEs can be realised if resource allocations in VEs guarantee QoS capacities as is the case in strictly physical infrastructures. The main question behind all new challenges to network QoS management introduced by virtualization is: *Which resources are concerned?*

This generic question motivates a set of virtualization specific questions, requiring solutions in order to develop a management architecture for network quality of service in VEs. The driving questions for this thesis are:

- **How to manage network QoS in virtualization environments?** The virtualization layer hides the underlying physical infrastructure, thus actively impedes network QoS management, because precise knowledge of topology and resources are required for effective network QoS management. How can effective QoS management be performed in the presence of this opposing element? Further, VEs mostly incorporate many host computers. This demands integrated management of the physical network and the virtualization systems, resulting in the question: what is integrated management of physical networks and virtualization with respect to network QoS?
- **How to cope with virtualization dynamics?** Besides automatic adjustments that today's virtualization facilities can perform, the VE is constantly managed by multiple users, creating, modifying or decommissioning VIs. As all physical resources are shared, each management operation may affect achievable network QoS. Coordination of management operations and the configuration of the overall system is a big challenge. To cope with this it is necessary to know how changes affect network QoS. Furthermore, to generally solve this problem it has to be determined if dynamic effects can be classified and how to handle classes of dynamics.

- **What is accurate modelling of a virtual component?** In general, QoS requirements on network connections and posing QoS demands are technology agnostic. Consequently, a model of virtual components suitable for network QoS management is generic where QoS requirements are defined. At the same time such a model is very specific to represent, select and employ available techniques and technologies capable of provisioning a virtual component. What are the requirements on a model of virtual components suited for network QoS management and what could such a model look like?

- **How to enforce network QoS in virtual infrastructures?** As described in Section 1.2, hosting platforms are shared in VEs. Enforcing network QoS requires a new set of strategies, to coordinate all providing resources into fulfilling QoS requirements. The requirements were originally posed at some abstract isolated VI and must be adapted to a component sharing resources. What are the constraints of virtual systems and what are coordination efforts are required?

- **What is a prototypical life cycle of a virtual infrastructure?** In a VE, creation and withdrawal of VIs happen frequently while other VIs are operational. Creation and withdrawal must be managed correctly to not impede network QoS of operational VIs. However, life cycles of VIs are understood merely intuitively. A sophisticated description of the VI life cycle does not exists. What are the concrete procedures for creating and withdrawing VIs? What changes can be performed on VIs and what are their procedures?

The main contributions in developing a management architecture are threefold, corresponding to the main challenges introduced by virtualization: a) the gap between abstract network paths with QoS requirements and specific configurations of components and technologies is bridged, b) dynamic changes are controlled and reflected in the VE's configuration and c) management operations on VIs are translated into atomic operations on the providing infrastructure and globally coordinated.

1.5. Approach and Outline

Current management of VEs focuses on VMs and groups of VMs. Having a network QoS architecture for VEs is a different way of performing management, by focusing on the interconnect instead of the endpoints. When depicting a network as a graph consisting of vertices and edges, current management mostly targets the vertices, while this approach attributes more value to the edges.

Network QoS management is a well understood problem and new challenges arise around virtualization only. The basic idea is to integrate management of virtual and physical components, to allow the application of available network QoS management.

Core to this approach is the unified software development process as laid down by Jacobson, Booch and Rumbaugh in [JBR 99]. Said process is structured into phases, separating design from implementation thoroughly. The unified software development process encompasses developing an architecture following a requirements analysis before the actual refinement and implementation. The architecture is the goal of this work. This is a top-down apporach, suiting the problem at hand, where the staring point for QoS management are abstract isolated VIs, tailored precisely to the services' needs, which are refined and arranged to be implemented by a physical infrastructure.

While the host of possibilities to implement virtual components constantly increases with the development of new technologies, the intentions of managers remain fairly static. To develop a sustainable QoS management architecture that can integrate multiple technologies, requirements are developed by performing a scenario analysis.

The resulting requirements and architecture are focused on network QoS and are not limited to the technologies currently available. Through the top-down approach, new use cases and management scenarios can be added and analysed as performed in Chapter 3. New technologies can be added at the “bottom”, implementing the architecture's models.

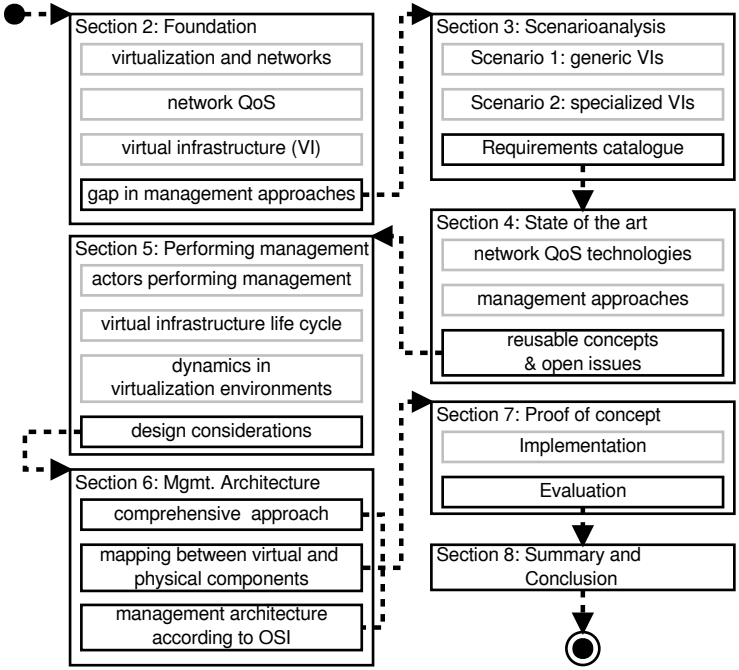


Figure 1.4.: Structure of this document

Following a top-down approach also serves identifying where established models can be reused. This allows focusing more on virtualization and its effects on network QoS management.

Outline of this work

The structure of this document is depicted in Figure 1.4. Chapter 2 begins with the generic make up of VEs and the common concepts that yield to a multi-layered view on networks built in VEs. After that network QoS

and network QoS management are introduced. The result of this chapter is a more detailed problem description what network QoS in VEs is to accomplish.

With this knowledge, Chapter 3 performs a requirements analysis and derives a requirements catalogue for a network QoS management architecture that is suited to perform effective network QoS management in virtualized environments.

With the refined view through the detailed problem description and the requirements catalogue, Chapter 4 arranges thesis with other work. Afterwards, techniques and technologies for network QoS and its management are discussed. There are many approaches that allow the provisioning of certain QoS features of individual links, but with a lot of requirements and assumptions on the links' endpoints. Consequently their management is scoped so narrowly towards specific link and endpoint types, that management approaches for networks and network paths have become a separate field of research. Using several examples, Section 4.1 illustrates that QoS technologies are available on varying abstraction layers within VEs. Section 4.3 then analyses management approaches that are intended to perform network QoS management simultaneously on all abstraction layers. This chapter shows that the available management approaches are not suited to fulfil all requirements identified and a new approach is required to realise network QoS management in VEs.

The following Chapter 5 goes beyond networks and QoS management, to create a comprehensive understanding of VEs and the management performed therein. A life cycle for VIs is developed as the basis for structuring management tasks. This enables situating network QoS management within management in VEs in general, contributing to this architecture's sustainability and towards integration with other management tasks and approaches. The chapter's results are prerequisites and requirements on a management system's behaviour and the identification of management roles. This guides the development of the management architecture in the following Chapter 6.

Chapter 7 documents a prototypical implementation and the architecture

is assessed, using the requirements catalogue developed in Chapter 3. This work is concluded in Chapter 8 with a summary and an outlook of further research possibilities and applications of the developed approach.

Previous publications

The author of this work is also responsible for previous relevant and cited publications. These publications either lead up to and shape the herein presented work, or elaborate on isolated aspects to trigger scientific dialogue and refine ideas. While the respective publications are cited where appropriate, this section explicitly states their relations to this work and author.

[Metz 09] **On I/O virtualization management**
This is this author's first publication and based on this author's diploma thesis "Design and analysis of techniques for virtualizing I/O-Channels". The paper focuses on the modelling of dynamic and static aspects of the management of virtual I/O devices. To grasp dynamic aspects, a life cycle for VMs was introduced. The need to recognise dynamic aspects is a result from this author's diploma thesis, while the life cycle and ensuing identification of management operations and procedures were new results, developed by both authors. The resulting life cycle definition and identified management operations constitute Section 5.2.1.

[Metz 10] **Towards end-to-end management of network QoS in virtualized infrastructures** (main author)
This publication states this author's intention to address network QoS management in virtualization environments. As part of elaborating on the overall vision and problem, this paper presents the classification of virtual components presented in Section 6.2.1. The corresponding classification of links had not been performed at that point.

[Metz 11] **Bottom--up harmonisation of management attributes describing hypervisors and virtual machines**
This publication is the result of an in depth analysis of available VMMs, with respect to the monitoring data they offer. In this work, [Metz 11] is cited in Section 6.2.3 as an example for illustrating technology specific metrics, and a second time in Section 6.3.3.4, as resource, providing information on how the herein specified models and procedures could be refined and extended to match specific technologies.

[Metz 12] **Link Repair in Managed Multi-domain Connections with End-to-end Quality Guarantees**
This paper introduces the concept of propagation rules to communicate and coordinate the realisation of QoS attributes. The class *PropagationRule* introduced on page 224 is a specialisation of the concept introduced in [Metz 12]. The specialisation performed for this work, uses a schema which is not part of [Metz 12]. The concrete repair and monitoring strategies introduced in [Metz 12] could be realised in a management system implementing the architecture introduced in Section 6, but this is beyond this work's scope.

[Metz 14a] **Dienstgütemanagement für Netze in Virtualisierungsumgebungen** (main author)
This publication offered this work's management approach for peer review. It virtually is this work's Section 6.1, motivated using a scenario including an early version of the classification of link types, as introduced in Section 6.2.1.

[Metz 14b] **Mapping virtual paths in virtualization environments** (main author)
This publication offered this work's finalised classification of link types from Section 6.2.1 and the refinement procedure from Section 6.2.3 for peer review.

1.6. Expected results

In VEs, VIs are designed and implemented according to customer needs. There is considerable abstraction between the VIs' effective implementations and the VIs as demanded and managed by the customer. Sensibly implementing and maintaining VIs involves much interpreting and mapping of management information and operations. This thesis introduces a management architecture based on an approach, designed specifically for management in VEs.

When requirements such as management capabilities and quality of service are taken into account, it may be required to introduce additional virtual components to extend the capabilities of a virtualization technology or as another layer of indirection. For example an additional VM can act as a virtual switch to perform traffic prioritisation, if the virtual switches provided by the virtualization layer do not offer such functionality. Implementation details like this are completely transparent for the customer and the system may remove the indirection implicitly (and automatically) when adequate resources become available. The introduced architecture foresees an automated mapping of QoS requirements to allow for such constructs to implement network QoS on the one hand. On the other hand the manager in charge of network QoS management is not concerned with such implementation details. This enables network QoS management in VEs.

The management task separated into management of abstract models, automated mapping, implementation and monitoring. This requires a generic topology model, capable of describing the relations between virtual links and paths and their implementing links and paths. Using this model, high level QoS management is refined to management operations that match the VEs current state, i.e. the placement of components, and to match the technology of the implementing component. This last step translates the refined generic management operations into management operations for the specific component. With this separation and ordering of activities the management architecture deals with the dynamics in VEs. Delaying the translation into component and technology specific management oper-

ations promotes implementations where adaptors are used to specialize the introduced interfaces to the actual employed technologies.

It shows throughout the analysis and architecture design, that network QoS management overlaps into many other management areas. The conceptual core of the developed approach is a management loop arranging management tasks in a manner that allows for interpretation and mapping where needed. With the overlaps into many other areas, this approach might be extended in future work to integrate network management in VEs.

Chapter 2

Virtual infrastructures in virtualization environments

Virtualization environments are a combination of many concepts with the overall goal of providing virtual infrastructures, tailored to the specific needs of individual services of customers. The key enabling technologies all perform some kind of virtualization. Characteristic for VEs is, that many strains of such technologies are used and that the services offered to data centre customers are realised using virtual components.

The basis to any VE is the physical infrastructure, a cluster of interconnected virtualization hosts. With virtual network components, networks extend into physical hosts, which used to be a singular non-forwarding entity at the edge of a network. This is an important new aspect for network QoS management. Another key feature of virtualization is a very loose coupling of virtual components to the physical set-up, enabling migrating components between hosts. This allows for great flexibilities in constructing VIs, and also creates a gap between VIs as managed entities on the one side and their current implementations on the other side. Closing this gap, to ensure VIs operate as expected by customers is the main concern of management performed in VEs. Performing network QoS management on virtual networks, spanning across multiple virtualization hosts, is the motivating goal for this thesis.

To establish basic terminology, this chapter briefly introduces the main branches of virtualization technologies, host virtualization in Section 2.1.1 and network virtualization in Section 2.1.2. After that, Sections 2.2 and 2.3 elaborate on the goal of developing a management architecture for network QoS in VEs, by introducing network QoS and commonly associated management activities and the concept of a management architecture and its intentions. Section 2.4 summarises key observations of this chapter.

2.1. Virtualization

The term virtualization is frequently employed when a multiplex is applied to provide multiple isolated instances of a single component. A complete definition can be found in [MNM Lind 10]. Most discussions on virtualization are specifically targeted at host virtualization, possibly because it is the most visible form of virtualization, providing VMs. This is also the main application described in Section 1.3.

The two major branches of virtualization introduced in this chapter are:

- host virtualization and
- network virtualization.

Host virtualization subsumes all technologies leading up to virtual network components and VMs. Network virtualization incorporates all technologies for providing isolated communication topologies.

2.1.1. Host virtualization

Host virtualization is employed on a single computer, in the form of a VMM, to provide many VMs. For any computing application, there are three defining aspects to a computer:

- its hardware architecture,
- the storage facility and
- the network uplink.

VMMs often implement all three, in order to be able to provide VMs.

Regarding performance capacities of VMs, the computer architecture is described as a combination of CPU time and available RAM. Therefore, in most literature, a VM is described with respect to the four aspects, CPU, RAM, storage and network. Figure 2.1 illustrates a physical computer, that has three virtual siblings.

To facilitate network access, the virtualization host implements a VN, so that the three VMs can share one physical uplink. There are various strains of virtualization technologies, focusing on single aspects or are even more specialised towards individual hardware components.

Most notably, NICs often employ single root I/O virtualization (SR-IOV), to provide multiple virtual NICs to the host computer. In effect, the VMM can assign each VM its individual NIC and need not worry about implementing a VN, thus offloading some tasks to the NIC. For QoS management the virtualization method matters: When implementing a switch, the VMM plays an integral part of effective network QoS management. When employing SR-IOV, the NIC becomes more critical than the VMM, as resources and the multiplex are handled by the NIC.

A complete overview techniques is given in [MNM Lind 10]. The important aspect for the problem at hand is, that ultimately it is the VMM that creates the virtual components.

Usually, a computer's OS provides abstraction from the underlying hardware, providing an uniform hardware independent platform. Host virtualization can be considered a next logical step towards more abstraction. Whereas an OS provides a platform for applications, host virtualization provides a platform for multiple OS. The general notion is depicted in Figure 2.2.

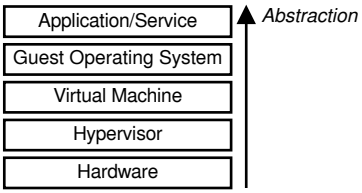


Figure 2.2.: Running applications with host virtualization

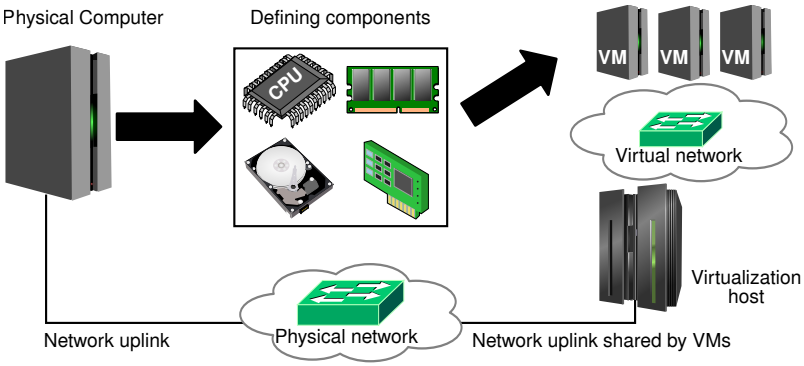


Figure 2.1.: Physical and virtual computers and networks

There are many motivations for employing host virtualization: Consolidating services onto fewer computers may free up valuable rack space, or be more energy efficient. While motivations may vary, the persistent concept in host virtualization is to break the tie between a computer's hardware and OS, move the OS into a container and have the OS work inside this container. The contained OS becomes a *guest* which is hosted by a VMM.

2.1.2. Network virtualization

In [TaWe 10], computer networks are defined as a *"collection of autonomous computers interconnected by a single technology"*. In a model view, there are two kinds of participants in any computer network: the autonomous computers as *data terminal equipment* (DTE) and intermediary *data communications equipment* (DCE), passing on data packets towards endpoints. Together, DTEs, DCEs and the links connecting them, constitute a network topology or *infrastructure*, thus the foundation for distributed systems. A stream of packets from one component to another is a *flow* [Zhan 87].

Typical DCEs are switches and routers, where switches operate on ISO OSI

Component	Used name	Participant role	Type
switch	switch	DCE	physical
router	router	DCE	physical
computer with VMM	host	DTE	physical
computer without VMM	server	DTE	physical
virtual switch	vswitch	DCE	virtual
virtual router	vrouter	DCE	virtual
virtual machine	VM	DTE	virtual

Table 2.1.: Components in virtualization environments

layer 2 while routers operate on ISO OSI layer 3. As depicted in Figure 2.1 and more explicitly in Figure 1.3 on page 9, there are also virtual versions of network components. Network components implementing virtualization are still emerging. Mostly, virtual network components are provided by the same hosts that provide VMs.

While similar in their perceived function, virtual network components must be managed differently than their physical role models. As the focus of this work is network QoS management, virtual DCEs must be seen as an additional independent kind of network component. Table 2.1 is a comprehensive list of components that may participate in virtual networks and are therefore subject to management for network QoS management in VEs. Notice that this list does not explicitly contain physical or virtual application layer gateways. Those require application specific knowledge and therefore they are assumed to always be implemented as VM.

The same technologies that are used for virtual endpoints are also used to provide virtual network components, e.g. routers, which can be as migrant as endpoints [WKB+ 08]. Any component can be virtual and migrating through the physical infrastructure over time. This is a new aspect in managing VN, compared to the traditional *"logical structures providing abstract and selective views of the physical network resources allocated to VN customers"* [NJC+ 99]. While the view on the VN remains unchanged

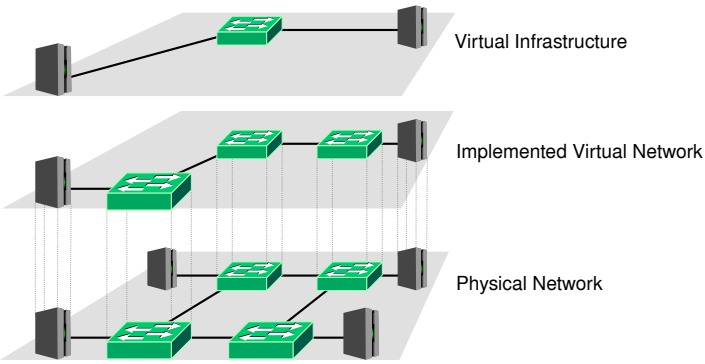


Figure 2.3.: An abstract virtual network, mapped onto a physical infrastructure

by migrating components, the actual links, the providing components and their allocated resources may change entirely.

VIs comprise subsets of DTEs, DCEs and links of a physical topology. Participating components may individually handle flows of different VNs, which forms the basis for QoS considerations. Additionally, the concept of a *virtual link* describes a *path* through a network which is abstracted to a single link of a VN.

Figure 2.3 illustrates the mapping of a VI onto a physical network. From a users perspective, the value of an infrastructure lies with the DTEs where applications and services are operated. Hence, a user is likely to ask for a network like the one titled *Virtual Infrastructure* in Figure 2.3. In such a simple example the switch merely represents the wish for the servers belong to the same network. In more complex infrastructures, network components may be explicitly requested, for instance a virtual routers, to enable access control management at the network layer for VIs. The *Implemented Virtual Network* are the components implementing the VI. Due to the placement of the two servers, the network must incorporate three physical switches.

Similarly to VMs detaching an OS from the underlying hardware, VNs

detach topologies from the underlying network. According to [TaWe 10] the *"usual advantage of virtualization […] is that it provides flexible reuse of a resource"*. For networks this means to expose or hide certain aspects of links and network nodes, for example other network participants, or details about a networks inner structure.

The abstraction of a network path to a single virtual link is a tool employed in network realization and network management alike. Figure 2.3 could employ virtual links to hide two of the three involved switches from the manager. Which switches should be hidden depends on the manager's goals.

With the intention of performing network QoS management, a virtual network link represents a subtopology. Performing management operations on a virtual link implies performing management on all components of the hidden subtopology.

2.2. QoS management

The previous sections describe network virtualization in VEs as a collection of technologies to implement network components and network topologies. Because of this shared use of a common environment, coordinating measures must be taken, to avoid the customers' applications interacting and affecting each other in unwanted manners.

One such aspect where VIs interfere with each other is resource usage. The computation, storage and especially network capacities of any host are limited, as are the capacities of (virtual) network components and links. It is possible for a greedy, resource intensive, single entity to starve the network data transfer of adjacent VIs. To avoid such situations and guarantee certain attributes to networks and the communication service they provide, providers perform network QoS management. With network QoS management a provider aims to implement, enforce and monitor assertions concerning characteristics of network data transfer.

2.2.1. Network QoS properties

Once connected to a network, DTEs may communicate with each other. For the users of communicating services and applications it is not relevant whether their communication traverses virtual or physical networks. In VEs, flows compete for resources within the physical network and virtualization hosts. The amount of available resources influences the characteristics, the quality, of a network flow. The classical metrics for network quality of service apply to virtual and physical networks alike. According to [TaWe 10] these are:

`Data rate` Data volume transmitted/received within a time frame.[1]
`Delay` Time passed between transmitting and receiving a message.
`Loss rate` Proportion of sent messages that were not received.
`Jitter` Expected or allowed standard deviation for measuring delay.

Next to QoS parameters that directly concern flows there are parameters concerning infrastructures as such, which also affect how well a network "works", for instance maintenance windows, which ought to be addressed as well [MNM Yamp 09]. During maintenance, problems are to be expected or the service may even be interrupted for a short period of time. While these parameters do apply to VEs, they must be addressed on a management level. For actual network flows the above four metrics are the most relevant.

Especially where VMs are involved, the network is merely a supporting service, which influences how well the actual service the customers want can operate. Consequently requirements on network QoS are associated or even directly derived from the "main" service. For effective network QoS management QoS requirements are formalised and agreed upon by customer and provider as part of the service level agreement (SLA). Within a SLA quality of service parameters, their measurement and reporting are laid down, among other specifications [MNM Scha 08].

[1] The term used by Tanenbaum and Wetherall is bandwidth, but especially when discussing networks this term is ambiguous.

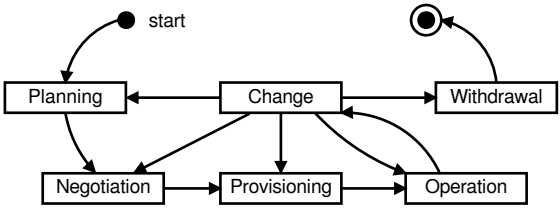


Figure 2.4.: The IT service life cycle [MNM Dreo 02]

After a VI has been deployed, the provider performs management to fulfil and enforce all quality of service parameters.

2.2.2. Performance management activities

VIs provided by a data centre to a customer are instances of an IT service. Service instances follow a life cycle, from the customers initial motivation to procure a VI, to its final decommissioning, when the user has no need for this service any more. For IT services, a generic life cycle, depicted in Figure 2.4, is described in [MNM Dreo 02]. Performing network QoS management is an ongoing task throughout the entire life cycle [MNM Gars 04].

During the planning and negotiation phases, network QoS requirements are defined with respect to a VI's purpose. In the provisioning phase, the infrastructure specification is put into action: components and networks are created and the network QoS requirements are turned into configurations for the providing infrastructure. During the operation phase, the provider ensures that the network performs within the specified parameters (the customer, may participate here as well!).

There can be direct changes to network QoS requirements, e.g. demanding a higher data rate, or network QoS requirements may change implicitly, with changes to the VI and the agreed usage/purpose in the SLA. As part of the provided VI, network QoS management plays a role during the withdrawal

phase in the sense that requirements must also be removed from the host infrastructure, to avoid future side effects.

Reviewing the IT life cycle with VIs in mind yields two important activities of network QoS management: specifying network QoS parameters and making them part of a VEs configuration. While network QoS metrics themselves are mostly those described in Section 2.1.2, the activities include:

- defining corresponding values for QoS properties,
- defining matters of measuring and reporting, and
- defining the flows for which QoS requirements apply.

According to [HAN 99], the specification of QoS parameters is one activity within the area of *performance management* of the functional model of the ISO Open Systems Interaction management architecture. Managing configurations of systems is its own functional area *configuration management*.

Continuing along the structure of the OSI functional areas in [HAN 99], *"performance management can be seen as a systematic continuation of fault management. Whereas fault management is responsible for ensuring that a communications net […] just operates, this is not enough to satisfy the objectives of performance management, which wants the overall system to perform well."*. The stated intention of network QoS management, to implement, enforce and monitor assertions concerning characteristics of network data transfer, map to the goal of *"performing well"*: the goal is guaranteeing characteristics of flows, not bare connectivity and ability to transmit data.

The full set of performance management activities according to [HAN 99] are:

- Establishing quality of service parameters and metrics[2]
- Monitoring all resources for performance bottlenecks and threshold crossings
- Processing measurement data and compiling performance reports

[2]This activity is restated for the completeness of this list.

- Evaluating history logs (i.e., records on system activity, error files)
- Carrying out measurements and trend analysis to predict failure before it occurs
- Carrying out performance and capacity planning. This entails providing analytical or simulative prediction models that are used to check the results of new applications, tuning measures, and configuration changes

As a combination of configuration and performance management, the immediate and constantly occurring activities in network QoS management during an actively used service are configuration and monitoring alike. Configuration is performed to set up VIs that are capable of fulfilling network QoS requirements. Monitoring serves the purpose of watching services operating, so that adequate configuration can be performed to end up with VIs capable of fulfilling network QoS requirements.

There are two main aspects to consider: determining effective implementations of intended VIs (cf. Section 2.1.2) and attributing QoS requirements a demand for resources that the VE can provide, for instance CPU time on a host or data rate within a switch. Continuous monitoring ensures that configurations are effective and the guaranteed network QoS is provided.

2.3. ISO OSI management architecture structure

Architectures are templates with concepts and concerns that can be implemented in ready to use systems [JBR 99]. To perform network management, management architectures consist of four sub models, which according to [HAN 99] describe:

- objects that are subject to management and their interrelations (information model),
- functions to perform management on managed objects (function model),

- roles and cooperation forms between managing entity and managed entity (organisation model)
- communication of management information between entities (communication model).

In a management architecture for network QoS management in VEs the central objects that are subject to management (*managed objects*) are physical and virtual DTEs, DCEs, network links and their QoS attributes, as well as the VIs they constitute.

The core management functions to facilitate network QoS perform resource allocation to components, links and flows. Further, in VEs, components and links can be created, modified, moved between hosts and destroyed as needed to construct networks that can meet user demands. As a last class of management functions, mappings of intended virtual networks onto the VE (cf. Section 2.1.2) are created, modified and removed.

The organisation model identifies different roles of managers by their intentions when using the management system. For VEs this mostly pertains to VIs vs. physical components and subnets.

The communication model describes all interaction patterns between managing and managed systems and the communicated information. A prominent aspect of the communication model developed in this thesis is the communication of available and used resources as well as monitoring data required for performance management.

2.4. Virtualization specific observations

In combination, the introduced aspects explain problems and new behaviour, that can be observed in VEs. For instance, without virtualization, resources, i.e. servers, had to be selected carefully when implementing services, while with virtualization, DCEs and DTEs are created and linked as needed. The problem of actually placing a service is transformed into placing (virtual) components.

Components can be regarded as generalised building blocks, which are used for all services. Few generalised building blocks are easier to manage as arbitrarily specified functional components of user defined services. This is an advantage when housing multiple services.

On the other hand, the technologies enabling these generalised building blocks add abstraction layers, resulting in multiple views on user faced services (by functional components) and their implementations, as depicted in Figure 2.3. Managing and coordinating multiple views with varying levels of abstraction is additional work, effectively making management of such services and their implementations more difficult. For instance, customer managers are usually restricted to higher abstraction levels, but may still be in charge of network QoS management of their infrastructures. Without any knowledge on component placement or the implementing physical hardware this is difficult to realise, as much network QoS related management tasks require data from hardware components or configure hardware components.

In VEs, there is at least one such additional layer, in the form of VMMs. Performing network QoS management in VEs is a task spread out across all abstraction layers. The task must be coordinated to be effective, because bottlenecks and shortages can occur on every abstraction layer, but ultimately every component requires physical resources to perform.

VIs are created and withdrawn with the services they implement. Coupling the beginning and end of a VI's life cycle this tightly to a service results in frequent changes to the number and placement of VIs running within a VE. Number and placement are two additional dimensions with dynamics for managing resource allocations and consequently network QoS. The corresponding management tasks are another new challenge to network QoS introduced through virtualization.

The intended architecture closes this gap created through additional abstraction and implementation layers to enable network QoS management in VEs.

Chapter 3

Scenario analysis

While the general problems of network QoS management are understood very well, the new aspects introduced by virtual components, their interaction and their effects on network QoS have seldom been subject to research efforts.

Network QoS is affected through performing management on network links and paths, and on individual components. All three happen regularly in VEs, in the form of high order management use cases where users interact with a management system to fulfil a certain task. The effective management operations are either an explicit step of a management use case, or an implicit configuration required to implement a high order management task.

This chapter starts out with introducing two real world scenarios for VEs, where different services are provided to customers. To show diversity in the introduced scenarios and applicability of the presented approach beyond these two scenarios, Section 3.1 introduces a morphology for scenarios, emphasising generic characteristics of VEs. Section 3.2 then introduces the scenarios and the therein provided services. As a next step, an abstracted scenario is developed in Section 3.3, where a single VE is used to provide all services described in both real world scenarios, as a basis for the requirements analysis.

The set of requirements can be used to evaluate any given management system or architecture for its capabilities for network QoS management in VEs. These requirements are also rationale for the management architecture for network QoS management in VEs specified in Chapter 6 and will be used as a means to evaluate the management software developed in Chapter 7.

3.1. Morphology of scenarios

The scenarios are chosen based on their differences in providing network access for, and management of, VMs and VIs. The intention is to capture a wide variety of use cases and in turn derive a comprehensive list of requirements for network QoS management in VEs. This section introduces aspects for discerning the later analysed management scenarios, to ensure that the developed architecture is applicable to multiple scenarios and is not casuistic for a specific problem.

Section 1.3 illustrates that the main problem introduced through virtualization that network traffic through virtual components has aspects of horizontal and vertical communication at the same time, with some information hidden. Based on this observation, the morphology chosen for the scenario analysis performed herein is also concerned with the availability of information. Information may be:

- available,
- not available,
- explicitly hidden under certain conditions,
- implicitly hidden through the employed techniques and technologies,
- static and always available, or
- subject to change and must be extracted and refined.

The diversity of services provided in the scenarios will reflect in varying use cases and supports the applicability of the developed architecture.

Table 3.1 shows the morphological field with all characteristics and their possible occurrences. In later sections, service specific instances highlight relevant characteristics of the services provided within the scenarios. This

Characteristic	Occurrence	
access network	local net	Internet
physical LAN usage	peer entity	tunnelled
VM placement	static	dynamic
VM LAN access	bridged	routed
VM access	direct	proxy | gateway
VM manager	customer	provider
QoS manager	customer	provider
QoS requirements	strict	flexible

Table 3.1.: Morphological field for managing network paths including virtual components

serves as a quick reference and as a tool to identify where variations in network QoS management are to be expected and must be supported by management systems.

`access network` … the service consumers origin network. From a management systems point of view, this is either a local network or the Internet. In local networks it can be assumed. that enough information is available, to identify all components along an end-to-end path. Where network paths span across the Internet, QoS management can only be performed within the own administrative domain. However, for the eventuality that Internet service providers enable and allow QoS connections across their networks, methods for coupling QoS domains are already available [MNM Roel 05].

`physical LAN usage` … the manner the LAN is used by virtualization. Mostly, VMMs implement a switching fabric and as a result network traffic needs or does not need to pass the physical LAN, depending on the placement of the communicating virtual components. A cluster of VMMs has the option to either bridge the virtual components into the physical LAN or implement tunnels in-between physical hosts, to keep VNs isolated from one another and the physical network.

These two different functions also affect how network QoS needs to be performed.

`VM placement` … the possibility to migrate VMs. Dynamic VM placement results in changing topologies and network QoS paths, imposing additional management problems and tasks, compared to static, unchanging VM placement.

`VM LAN access` … the manner a VM's network traffic leaves the VI for the Internet or other hosted VIs. Traffic can either be bridged or routed into the LAN. Both cases are relevant to network QoS management.

`VM access` … the manner customers access a VM. Customers may access services on the VM directly over the network or there are application layer gateways that facilitate access to services provided by VMs. An application layer gateway may greatly influence network QoS management: they are mandatory parts of network QoS paths and often perform protocol conversion, as the example presented in Section 3.2.2 shows.

`VM manager, QoS manager` … the customers possibility to immediately perform management. If customers perform management their users' views are restricted to their respective VI(s). If it is the provider performing management, those restrictions are unnecessary. Dealing with views and separations thereof is a concern of any management architecture.

`QoS requirements` … the possibility to alter QoS requirements after deployment. Strict QoS requirements can not be altered, flexible QoS requirements can be altered. Considering the use case analysis that is performed starting in Section 3.4, the impact on QoS management are management tasks persisting to the modification of already deployed network QoS paths.

3.2. Real world scenarios

Section 3.1 introduced a morphological field. This section employs it to characterise services provided as part of application scenarios, according to how and by whom network QoS is to be provided and managed.

The scenario in Section 3.2.1 is implemented by the Leibniz Supercomputing Centre (LRZ). It features varying types and sizes of VIs and several managers with different tasks, concerns and insight into the providing infrastructure. This scenario is very similar to the common infrastructure as a service (IaaS) and platform as a service (PaaS) scenarios known from cloud computing. This scenario is suited to represent large scale VEs in general.

The second scenario, in Section 3.2.2, is situated at the Technische Universität München (TUM), where virtualization is employed to provide and manage desktop operating systems for employees and students. This scenario features varying access methods and integration with the physical networks, which result in different and changing network QoS requirements.

For the use case analysis, an abstracted scenario combining the characteristics of both real world scenarios is created in Section 3.3. This scenario is suited to implement all management use cases for performing network QoS management in VEs.

Besides real world examples for services provided using VEs, the scenarios provide organisational structures in which responsibilities for performing management on VEs have been recognised and assigned. In both scenarios, services, management roles and organisational structure can be expected to remain unchanged and unchallenged when introducing network QoS management to these environments. Therefore the abstracted scenario provides a reasonable setting for elaborating use cases and deriving requirements.

3.2.1. Scenario I: LRZ hosted infrastructures

The Leibniz Supercomputing Centre (Leibniz-Rechenzentrum, LRZ) of the Bavarian Academy of Sciences and Humanities is an IT service provider for both universities in Munich and other publicly funded research facilities. Its services range from general customer faced IT services, over managing the network infrastructure connecting its customers in the Munich area (Münchner Wissenschaftsnetz, MWN), to operating one of the world's fastest supercomputers [LRZ 12].

The three interesting LRZ services for this work are:

`Svc.#1` Attended hosting of virtual servers
`Svc.#2` Unattended hosting of virtual servers
`Svc.#3` Project infrastructures

To facilitate their services, the LRZ operates a dedicated physical infrastructure, consisting of 80 blade-servers housed in 16 enclosures. All blades have the same VMM, VMware ESXi, installed and VMs gain network access strictly through the VMMs provided virtual switches. Every server has the same virtual switches configured and each switch is assigned a VLAN ID which is valid in the physical network. Consequently each VN is accessible on every server and every VN maps to exactly one VLAN ID in the physical network.

Figure 3.1 illustrates the topological set-up within the LRZ scenario. The boxes at the top depict the three identified services. Below, there are four virtual topologies which are the networks to which virtual servers can be connected. The networks are implemented to distinguish between MWN-accessible vs. Internet-accessible and attended vs. unattended servers. Virtual servers are not allocated their own VLAN(s), hence there is a relation between network configuration and potential infrastructure managers. The router depicts the edge node through which network traffic enters and leaves the VE. Dedicated VIs share the same Internet and MWN uplink, but are not directly connected to the VNs of virtual servers.

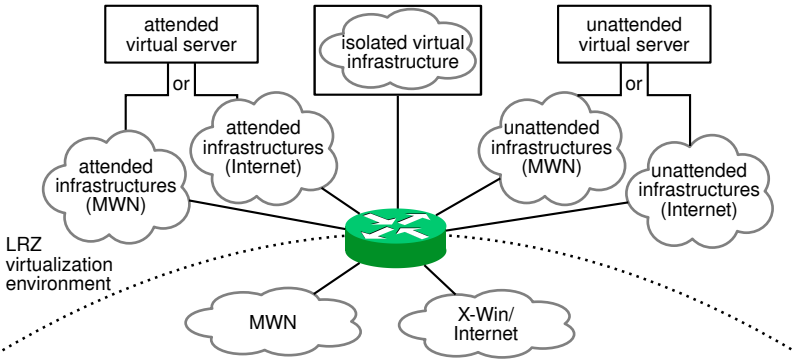


Figure 3.1.: Special purpose topologies in the LRZ virtualization environment

Svc.#1 Attended hosting of virtual servers The attended variant of hosted virtual servers gives the customer an infrastructure where the LRZ takes care of most management tasks. This even includes the operating system and all its services, for instance an entire webserver, including the HTTP daemon and database system. As the LRZ provides the entire platform, i.e. the webserver, this service can be considered as PaaS. The customers are only concerned with providing their content. If the customer has specific requests pertaining to network and system management, these requests are directed towards the LRZ service desk as customers have not got any means of directly performing management themselves.

Initially the customer requests an infrastructure which is to be provided and operated by the LRZ. The topology and properties of that infrastructure, especially network QoS requirements, are formalised in a service level agreement and hence considered strict. After the infrastructure has been provisioned, the LRZ completely takes care of operating the infrastructure within the specified parameters.

Table 3.2 summarises the characterisation of this service flavour according to the morphological field introduced in Section 3.1. The defining aspects

Characteristic	Occurrence		
access network	local net	Internet	
physical LAN usage	peer entity	tunnelled	
VM placement	static	dynamic	
VM LAN access	bridged	routed	
VM access	direct	proxy	gateway
VM manager	customer	provider	
QoS manager	customer	provider	
QoS requirements	strict	flexible	

Table 3.2.: Morphological field for attended hosting of virtual servers at the LRZ

Characteristic	Occurrence		
access network	local net	Internet	
physical LAN usage	peer entity	tunnelled	
VM placement	static	dynamic	
VM LAN access	bridged	routed	
VM access	direct	proxy	gateway
VM manager	customer	provider	
QoS manager	customer	provider	
QoS requirements	strict	flexible	

Table 3.3.: Morphological field for unattended hosting of virtual servers at the LRZ

for this instance are `VM manager`, `QoS manager` and `QoS requirements`. The other aspects follow the initial scenario description.

`Svc.#2` UNATTENDED HOSTING OF VIRTUAL SERVERS With unattended hosting, customers manage their VIs themselves, rather than delegating management tasks to the LRZ. This means, that operating and maintaining the OS of VMs is not of the LRZ's concern. As a first consequence, the customer is free to assign the VMs' resources to programs and processes arbitrarily and any time. Hence, the QoS requirements can be changed more

Characteristic	Occurrence		
access network	local net	Internet	
physical LAN usage	peer entity	tunnelled	
VM placement	static	dynamic	
VM LAN access	bridged	routed	
VM access	direct	proxy	gateway
VM manager	customer	provider	
QoS manager	customer	provider	
QoS requirements	strict	flexible	

Table 3.4.: Morphological field for project infrastructures at the LRZ

flexibly compared to the attended variant of the service. The involvement of the LRZ Service Desk is less frequently required as a mediator to communicate new or changed requirements. The customers perform management on the VM directly, but requests pertaining to the network still go though the Service Desk.

Similar to the attended variant, the defining aspects of this service are `VM manager`, `QoS manager` and `QoS requirements`. This service's characterisation is summarised in Table 3.3.

`Svc.#3` PROJECT INFRASTRUCTURES Dedicated infrastructures for projects include many VMs and at least one VN, which is not shared with other customers. In general, the VMs are unattended virtual servers, not managed by the LRZ, but connected to an isolated VN, instead of one of the four provided networks shown in Figure 3.1.

The most notable difference to the other services is the amount of involved virtual components. While the creation and withdrawal of VMs has to be requested via the LRZ Service Desk, customers are given control over a resource pool and may arbitrarily assign those resources to their VMs.

To enable this kind of customer performed management, customers are granted access to a management system, through which resource allocations

to VMs can be performed and tasks can be further delegated. To support this, customers are free to create multiple accounts within the VM resource management system and delegate shares of their initial resource pool.

This service's morphology is summarised in Table 3.4. The main property setting this service apart is `VM LAN access`. However, the degrees of freedom for `VM manager`, `QoS manager` and `QoS requirements` are greater compared to the unattended virtual server service, as there are more aspects about the components that may be changed.

3.2.2. Scenario II: TUM FMI virtual desktop infrastructure

The Technische Universität München (TUM) is one of Munich's universities. The Center for Mathematics and the Department of Informatics (FMI) are located within the same building at the campus Garching. IT Services are provided centrally for both departments by the same IT group, and the LRZ. One of these services is providing virtual desktops for researchers, students and faculty. The desktops are accessible from anywhere within the FMI building and the Internet.

For this service, the FMI IT group operates a VE, a virtual desktop infrastructure (VDI). Figure 3.2 illustrates the key contributing systems. The service provided to users is the box at the top left position. Networks are depicted as clouds and the subsystems realizing remote access to VMs are ellipses in the "FMI VDI remote access" quadrant. Users are given three distinct ways to access their workstations: physical SunRay terminals within the FMI facilities, a VNC based Java web client and directly, as every VM is given a public IP address. The physical terminals are located at various places in the FMI building, while the Java web client is suited to access one's desktop from anywhere on the Internet.

The VMs are implemented by an Oracle, virtual desktop infrastructure, based on the VM monitor Openbox. The usual mode of operation is one unified LAN ("access network for workstations" in Figure 3.2) and traffic separation is implemented using IP subnets. There are two kinds of traffic: user data, for instance surfing the Internet, and virtual input/output (I/O)

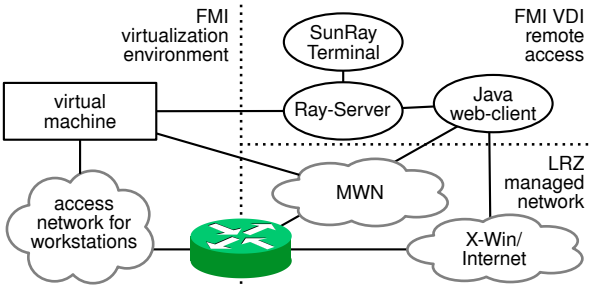


Figure 3.2.: TUM FMI virtual desktop infrastructure

channels. Openbox exports several I/O channels for each VM through an extension to Microsoft's remote desktop protocol (RDP) called VRDP. Most notably these channels are the display and sound output, as well as mouse and keyboard input. In combination with the VMM, the Ray Server interacts with VMs through VRDP to provide remote access to VMs through the VMM, rather than through a VMs network connection.

Especially for faculty members, VNs may be implemented to integrate virtual desktops into the private networks of their chairs. Figure 3.2 depicts this as the VM's direct connection to the MWN, to which the chairs belong. As described in Section 3.2.1, the MWN is managed by the LRZ. This means for a VM's integration into a chair's network the FMI IT group and the LRZ work together to facilitate the network path.

Svc.#4 Personal desktop The main feature of this service is presenting users their desktop and working environment independent from their locations or methods of access. The users are in full control over their operating systems and configure it as they see fit, while VM integration into the local network and especially network QoS are of the sole concern of the service provider. The QoS requirements are considered dynamic, varying with the user applications, desktop resolution and origin, from where user access their desktops.

Characteristic	Occurrence		
access network	local net	Internet	
physical LAN usage	peer entity	tunnelled	
VM placement	static	dynamic	
VM LAN access	bridged	routed	
VM access	direct	proxy	gateway
VM manager	customer	provider	
QoS manager	customer	provider	
QoS requirements	strict	flexible	

Table 3.5.: Morphological field for TUM FMI virtual desktop infrastructure

The VMs are accessible over the Internet, as they are given public IP-addresses that are routed announced properly. Even though direct access, for example via SSH, is possible, the virtual workstations are primarily accessed via the SunRay-Server. Further, the VMs are all meant to fully participate in their local LAN and allowed to interact with each other as peer entities. To interact with non VMs, network traffic is routed.

VM placement is static, which means once a VM is deployed it will remain on that host throughout its lifetime. However, after a few idle hours a VMs state is saved and the VM destroyed. The system always keeps 300 unused "template" VMs around and when a user connects to a desktop the VM state is quickly restored and pushed onto such a template VM. This is how quick access for users and sensible resource regaining are implemented at the same time. Consequently, while VM placement is static, a specific OS can switch hosts without being rebooted.

The characteristics of this service are summarised in Table 3.5.

3.3. Abstracted scenario

Combining the settings and services from the introduced scenarios yields an abstracted scenario where a single service provider operates a versatile

VE. Every service can be delivered multiple times to different customers and the resource consumption of all instances must be coordinated. The services have very distinct features, so that this abstracted scenario can be used to identify all requirements, instead of analysing all other scenarios individually.

The MWN from the real world scenarios is abstracted to an *internal network*. This may still be a routed network of interconnected LANs, but not the Internet. In other instances the internal network could be understood as "company internal". As introduced in Section 3.2.1 there are four predefined networks to which virtual servers can be connected. The criteria for which network is selected are attended vs. unattended and Internet vs. internal network. The attended Internet network used for the services `Svc.#1` and `Svc.#2` is used for the same purpose as the access network for workstations for the service `Svc.#4`. Hence, these two networks can be consolidated into one Internet subnet for the abstracted scenario.

In addition, virtual desktops can also be integrated into LANs of the internal network. Implementations thereof are an isolated network, similar to that of project infrastructures. While project infrastructures are routed into the Internet, the networks for virtual desktops are bridged into the LANs.

Figure 3.3 shows a VE, offering the services introduced in the four services introduced in the previous Section 3.2. The Figure shows the remote access infrastructure for virtual desktops as a single component within the VE. The specific MWN of the real life scenarios has been replaced with a generic "internal network", which is managed by the provider, but outside the VE. To match the described service, there is only one VDI access infrastructure.

Virtual network endpoints

The abstraction from the real world scenarios into a VE offering all introduced services yields four different types of VN endpoints, i.e. VMs, that are offered to customers as part of a service:

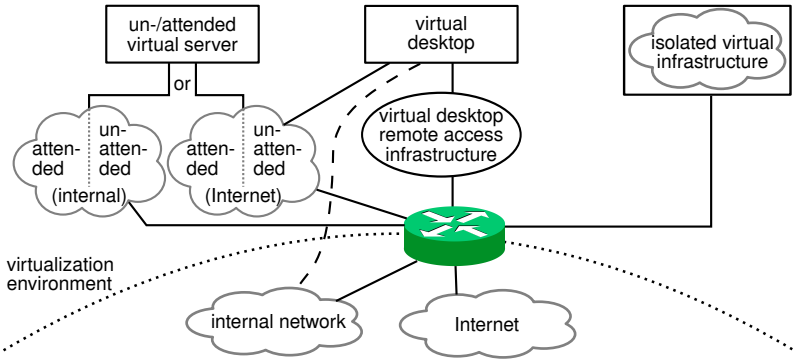


Figure 3.3.: Abstracted scenario, uniting different services

- A virtual workstation (VWS), mainly providing a desktop to which an user from outside the VE connect via an application layer gateway, similar to the SunRay-Server.
- The remote access server (RAS), an application layer gateway used to access virtual workstations. The gateway does not need to be virtual, but the network between the gateway and the virtual workstation is always transparent to the user.
- A virtual server (VSi), providing services which are used by other virtual computers that are part of the same virtual topology. All communication with this virtual server is VI internal, indicated through this component type's suffix `i`.
- A virtual server (VSe), providing services employed from not further specified clients. This virtual server can communicate with components that are not part of its VI. The capability for external communication is indicated through this component type's suffix `e`.

The simplest virtual topology is a single VSe, connected to the Internet, corresponding to an un-/attended server, as described in `Svc.#1` and `Svc.#2`. Virtual topologies for project infrastructures are comprised of many servers,

where some are VSe VMs and others are VSi VMs. VWS VMs are special in the regard that they may be accessed via the Internet, as well as a RAS, which is also part of the VE. Not necessarily as a VM, but as a component managed by the provider that interacts with VMs of customers.

In the presented scenario the main communication partner of VWS VMs is the user of the virtual desktop. Such specific knowledge about the application already allows for conclusions on its expected QoS requirements. As the users are interacting with remote programs as they do with locally run programs, they are likely to expect a similar responsiveness and therefore the network must deliver a low delay connection. Similar considerations are valid for RAS, in addition to a requirement to handle many simultaneous connections, as they are central access points. Performing network QoS management for these VMs is an example for network QoS motivated by dynamic application requirements, depending on service consumption, that must be mapped onto the virtual components and network.

VSi VMs provide supporting services, that are indirectly used through VSe or VWS VMs. Their communication partners are always within the provider's domain, which allows for specific QoS requirements with respect to the consumers of their services. On the other hand, VSe VMs are mostly accessed from outside the provider's domain, which often results in generalised QoS requirements focused on the individual VM, rather than the end-to-end network path. Performing network QoS management for these VMs is representative for constructing a topology with rather static properties from a limited set of resources.

3.4. Network QoS management requirements

VN components share physical resources with all other virtual components, especially VMs. Consequently, all management affecting resource allocation and availability may have side effects on the achievable network QoS.

This section's goal is to capture side effects on network QoS through the existing real world use cases. To guarantee network QoS the side effects

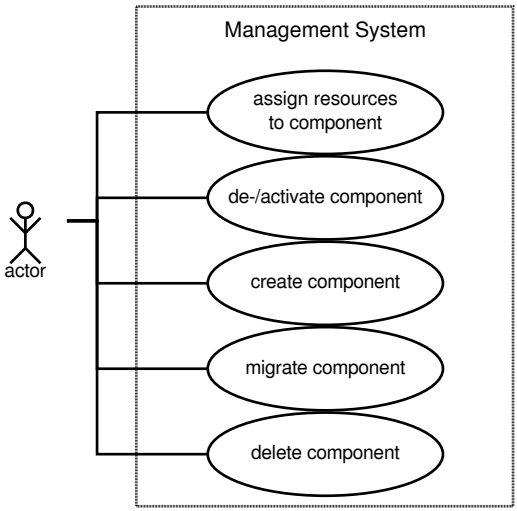


Figure 3.4.: Use cases relating to virtual components

from resource allocations must be handled correctly or even avoided. This rationale is the basis for this requirements analysis.

This requirements analysis follows the method described in [JBR 99]. Each use case is described and detailed out, so that side effects on network QoS can be heuristically identified and requirements on network QoS management formulated.

For the requirements on network QoS management in VEs, management of virtual components must be analysed. The identified use cases are shown in Figure 3.4. The summary of the performed analysis can be found in Appendix A, use cases 1 through 6.

The use case descriptions refer to a collection of managed VMs as VI, even though the network aspect of VIs is not represented in the currently

employed management systems. The term is still used in this section to avoid the need to introduce and discuss implementation specific concepts.

The full list of unique observations, identified as potential for side effects on network QoS, is:

- A network uplink is provisioned on the target host.
- Component placement defines how the VNs spans across the physical topology.
- Network components cannot be allocated resources from resource pools.
- Resources allocated to VMs cannot be used by VN components.
- Resources on the host can be released and in turn allocated to network components.
- The components network uplink on the origin host are not required after the migration completes.
- There is a time interval where one component blocks resources on two hosts.
- When a component is activated it binds and blocks resources on a host.

The analysis shows, that management operations on single virtual components almost always affect multiple components which are very often network components. This is considered strong reason for requiring automation. Table 3.6 provides an overview of the requirements derived from the potential side effects on network QoS resulting from management operations on virtual components. The full description of the identified requirements can be found in Appendix B.

3.5. Summary

This chapter analyses management performed in current VEs with the aid of two real world scenarios. Its focus is on resource management in the context of network QoS management. The use case analysis identifies potential for side effects between resource management on VMs and network QoS

Req.-ID	Title
Req.#1	Resource allocation to VMs
Req.#2	Resource allocation to VN components
Req.#3	QoS links can be specified with virtual endpoints
Req.#4	Support for different types of QoS paths
Req.#5	VI semantics for performing management
Req.#6	Monitoring structures for VIs
Req.#7	Management users are restricted to their associated VIs
Req.#8	Multiple concurrent and customer managers
Req.#9	Automated adaptation of links
Req.#10	Automated enforcement of network QoS requirements

Table 3.6.: Summary of network QoS management requirements

management. This leads to requirements on a management system that is capable of providing network QoS in a VE.

The performed requirements analysis looks at VE as a single entity that provides VIs. The derived requirements are on a management system that allows network QoS management, as intended.

Chapter 4

Related Work

This chapter provides an overview of available technologies and approaches relevant for developing a network QoS management architecture for VEs and arranges them with this work and the identified requirements. The conceptual aspects of network QoS management and management architectures is understood very well and introduced in Section 2.2.

Figure 4.1 illustrates the partitioning of the network QoS management problem, for marshalling related work:

`Technology` … methods for creating network links and components with guaranteed properties.
`Management` … approaches for management automation and VI semantics.
`Integration` … architectures for network management, including virtualization.

In developing a management architecture, this work aims to orchestrate available technologies to build VIs. These technologies constitute the first area. Management and integration are both aspects belonging to the operational specifics of VEs. The management area means specific approaches relating to single problems. The orchestration of individual solutions to a comprehensive network QoS management architecture constitutes the integration area.

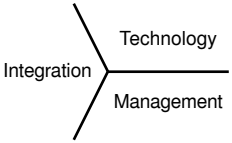


Figure 4.1.: Key aspects to network QoS management in VEs

Currently, most research and development is focused on the technology aspect. Where QoS is also subject to research, the focus is on specific combination of scenario and technologies, neglecting heterogeneity and integration. The architectures introduced within this chapter are located around Cloud Computing and Future Internet, where the management aspect is more the centre of attention and the make ups of data centres take more of a subsidiary role.

Requirement `Req.#4` lists varying types of links with unique properties due to their endpoints and can be used as an indicator for how focused or generic an approach is. Regarding the analysed architectures in this chapter, the conclusion of this chapter is that there are available approaches for some of the posed requirements and problems, but network QoS management in VEs in its entirety remains unsolved.

4.1. Network QoS Technologies

The technologies at hand are the basis for any consideration of a network QoS management architecture. To effectively perform QoS management, every employed component must have some capacities for guaranteeing QoS relevant aspects. For virtual components this means the entire provisioning stack, from the abstracted component down to the contributing hardware resources, must have such capacities. A network QoS technology, in this context, is an implementation of a specific component, virtual or physical,

with QoS capacities, so that a management system can use it to realise network QoS in VEs.

The following is a selection of available previous work, illustrating for each layer in the provisioning stack at least one implementation with QoS capacities can be found. The main discussed contributions are:

- Project Crossbow
- A survey on QoS aware resource management
- Currently popular VMM implementations
- Currently popular implementation of network components

The most sophisticated approach addressing this problem was project Crossbow [TDSB 09, SUN 09, TDS+ 09] by SUN Microsystems. It developed a specialised virtualization facility and corresponding hardware, that allows for fine grained resource allocation to virtual network interfaces and links.

Approaches, such as Crossbow, where software and hardware are very closely coupled, are not the regular case and require homogeneous hardware. VEs are scale-out clusters, as those presented in the real world scenarios in Section 3.2. Here, the important aspect of scale-out clusters is that they are not built once, but grow over time, by adding additional servers [SMEVH 10]. Even if a cluster starts out with homogeneous hardware, it must be expected that there will be heterogeneous hardware. This is discussed in [CaJi 09] and [PGP+ 10], and the need for adequate management is recognised.

A survey [RyKo 13] on network virtualization for QoS aware resource management names incompatibility of available QoS approaches in implementation and description as two of the most pressing research challenges in the area of network virtualization. It becomes clear, that the problem is orchestrating subsystems and components into providing VNs with guaranteed QoS properties. On the other hand, individual subsystems and components that implement mechanisms that can be used to achieve network QoS are available.

According to [BGRS 11], in 2009 VMware had a 80% market share with its brand of VMM. One of it's biggest challengers is the Xen hypervisor, which is used by Citrix and Oracle for their virtualization products, as well as for some of the biggest (perceived) Cloud Computing providers, such as Amazon [Shan 09, WaNg 10]. Both VMM implementations support resource allocations to VMs as per requirement `Req.#1`.

There is not any equivalent support for resource allocations to VN components as required by `Req.#2`. However, there are available technologies that can contribute it if managed and integrated correctly. For instance, the Open vSwitch [PPK+ 09] that is often used together with the Xen hypervisor [RyKo 13], has the capability to limit data rates and perform further scheduling in combination with OpenFlow [MAB+ 08]. This could be modified to serve as access gateway to established QoS approaches, for example [AAC+ 03].

There are many approaches that realise network QoS even when there are virtual components involved. Project Crossbow and the just drawn combination around Open vSwitch are two representatives. Crossbow illustrates how extensible the concepts of virtualization are, so that one can add and apply highly specialised technologies to enforce network QoS. On the other hand, Open vSwitch and OpenFlow are highly evolved concepts with open source implementations, which enable many architectural approaches. The list of use case specific implementations could be extended, but does not yield any further merit. The key point is that network QoS can be implemented with available technologies and this thesis may focus on the management thereof.

4.2. Internet focused approaches

With goal of network QoS management in VEs in mind, the architectures introduced in Section 4.3 best represent current approaches. There is a plethora of other management approaches, which employ VNs to remedy problems of today's Internet architecture. The most recent of these approaches pick up on "*How to lease the Internet in your spare time*" [FGR 07]

Project	Ref.	Architectural Domain
VNRMS	[NJC+ 99]	VN management
Tempest	[VdMR+98]	Enabling alternate control architectures
NetScript	[dSYF 01]	Dynamic composition of services
Genesis	[KCC+ 01]	Spawning VN architectures
VNET	[XCL+ 12]	Virtual machine Grid computing
VIOLIN	[RJXG 05]	Deploying on demand value added services on IP overlays
X-Bone	[Touc 01]	Automating deployment of IP overlays
PlanetLab	[PACR 03]	Deployment and management of overlay-based testbeds
UCLP	[WCS+ 03]	Dynamic provisioning and reconfiguration of lightpaths
AGAVE	[BLG+ 07]	End-to-end QoS aware service provisioning
GENI	[BCL+ 14]	Creating customized VN testbeds
VINI	[BFH+ 06]	Evaluating protocols and services in a realistic environment
CABO	[FGR 07]	Deploying end-to-end services on shared infrastructure

Table 4.1.: Architectures compared in [ChBo 09]

or "*Overcoming the Internet impasse through virtualization*" [APST 05a]. This section arranges these approaches with this thesis and the architectures analysed in Section 4.3.

The general notion of that research is to tailor VNs, to fit the varying needs of services. Virtualization, especially VN components, merely serve as vehicle to implement multiple instances of routing software with different fine tuning. A comparison of 13 such architectures, listed in Table 4.1, is performed in [ChBo 09]. CABO is the original architecture employing VN components, introduced in [FGR 07]. From the presented list, VNET is the most applicable to network management in VEs.

VNs are core to these approaches and QoS is an aspect in many, but solving the challenges through virtualization are not their business. Instead, it is assumed VEs can be managed so that resource allocations to VN components and links has a predictable or even deterministic effect on VN capacities.

4.3. Comprehensive management approaches

As per requirement Req.#5 network QoS management is performed on abstract VIs. Derived from analysing real world scenarios, the idea behind this requirement is to separate the management of VIs and the providing infrastructure. This allows VI managers to perform their tasks and need not about the underlying infrastructure and the provider need not disclose internal information about the VEs.

The comprehensive approaches subject to this section fulfil two requirements:

- VNs are subject to management.
- There is an abstract representation of each managed topology, that does not carry information about other managed topologies.

The first requirement is chosen to find relevant approaches. The second requirement is a (much) weaker version of requirement Req.#5. It is implied, that management approaches which work with isolated abstract representations, are more like to address the issues at hand. The second requirement is to filter out highly specialised use case specific approaches, that do not target VEs.

The mapping of VI components onto the physical infrastructure is not guaranteed to be static or even injective, as illustrated in Section 2.1.2. In [PGP+ 10] the authors discuss the diversity and heterogeneity of VN components and their provisioning. Amongst others, [MVKK 12, BKFS 07, TaYa 10], [PGP+ 10] recognises VM migration and the changes to the network as management problem that needs to be addressed. This concurs with this thesis' identification of frequent migration and changes to the size and topology of VIs as the main drivers for the dynamics in VEs, in Section 1.4.

A study [SoAn 10] performed in 2010 analysed management operations performed on VEs. It is determined, that for average sized deployments of VEs, in 2010, the typical management operations include 90 start ups of VMs and 50 automatic VM migrations, per day; peak values measure over

1000 daily occurrences of a single management operation. These numbers illustrate the constant dynamics in VEs, to support the requirements for automated management.

The architectures introduced in the following aim at QoS management with the involvement of virtualization. Each approach is relevant, because some aspects of the requirements formulated in Section 3.4 are covered. However, neither management architecture is aimed at all the requirements identified in this thesis. The work discussed in the following is representative for the current situation in the area of network QoS management in VEs.

The analysed architectures are:

- Autonomic architecture for virtual network piloting
- Dynamically Adaptive Virtual Networks for a Customized Internet
- Intelligent Service Oriented Network Infrastructure
- VNET

4.3.1. Autonomic Architecture for Virtual network Piloting

The Autonomic Architecture for Virtual network Piloting (AAVP) [FAB+ 11] is the most recent development of continued research efforts towards automated management. The researchers focus their work around future Internet technologies [Pujo 08, FBA+ 11]. Their subjects to management are foremost virtual routers, which offer most merit to networks and services in future Internet, or next generation Internet, scenarios [APST 05b, WKB+ 08, SWP+ 09].

Figure 4.2 shows a deployment of the AAVP architecture, situated entirely on a single virtualization host. Its three active components are the *Virtualization Context Collector* (VCC), the *Virtualization Piloting Decision Maker* (VPDM) and the *Virtualization Manager* (Virtualization Resource Manager, VRM in [FBA+ 11]). The key idea is that every network component has an autonomic piloting following the proposed architecture and does the right thing to implement virtual topologies and even QoS with virtual endpoints.

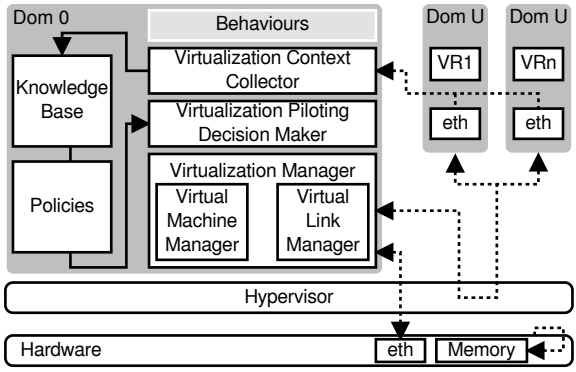


Figure 4.2.: AAVP architecture, adapted from [FAB$^+$ 11]

The AAVP architecture recognises that users/customers use an abstract description of the topology to be implemented. However, this does not quite satisfy the requirement for VI semantics, as their specification schema [FAP 10] foresees a strict 1:1 mapping for links and components. Section 1.2 illustrates the complexity of managing virtual links using the example of a virtual twisted pair cable. The cable is implemented either as a relatively simple forwarding between two VMs on the same host, or as a path between two VMs on separate hosts, hiding multiple links and components from the user. Through its 1:1 mapping the AAVP has no means of hiding paths or path segments. Consequently its monitoring approach, realised through the VCC, does not need to map and convert monitoring structures.

A system following this architecture can automatically enforce QoS locally on each host and can realises QoS paths specifically for virtual endpoints. It exchanges data between knowledge bases to coordinate all hosts into providing network QoS paths for VIs. The concrete interaction schemes with managers and the interactions between autonomous piloting systems have not been laid out, yet. Similarly different types of QoS paths are beyond the current scope of AAVP.

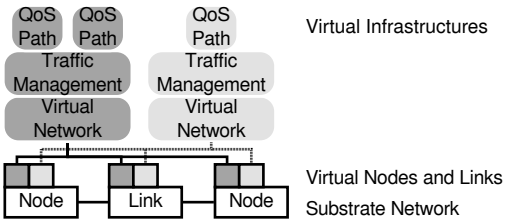


Figure 4.3.: Example DaVinci set up

4.3.2. Dynamically Adaptive Virtual Networks for a Customized Internet

The architecture for Dynamically Adaptive Virtual Networks for a Customized Internet (DaVinci) [HZSL+ 08], provides network paths with guaranteed QoS properties. VNs are used to concurrently have different methods of traffic management and traffic engineering on the same physical infrastructure. Through this, VNs implement different traffic classes as well as different methods for measurement and enforcement, while virtual components and links are used for resource allocations. Requested network paths are implemented by the VN whose methods are best suited to realise the QoS requirements of the requested path.

Figure 4.3 shows an example for realising network paths with guaranteed QoS properties using the DaVinci approach. Each node and link of the physical infrastructure (*substrate network* in [HZSL+ 08]) is virtualized, so that every VN topology is a subset of the physical topology. For each VN customised traffic management is performed, resulting is different QoS properties for which a VN can make assertions.

In the DaVinci architecture, individual QoS paths are what is requested and delivered to customers, similar to the VIs introduced in Chapter 3. While this architecture has a very narrow scope of intended use, its requirements and main concerns compare to this thesis. Its abstracted description and

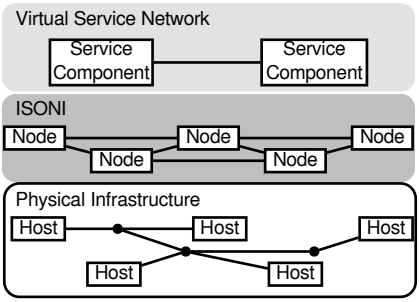


Figure 4.4.: Realising virtual infrastructures with ISONI

handling of QoS paths corresponds to this thesis' requirement for VI semantics. Still, this changes the required monitoring task so that it is not comparable to monitoring in VEs. The need for automated adaptation of paths and QoS enforcement are central aspects of DaVinci.

As a purely network focused architecture, DaVinci is not concerned with endpoints at application level and does not recognise varying types of QoS path. It is an architecture targeted at implementing links. Consequently the managers of VEs do not have corresponding counterparts in the DaVinci architecture.

4.3.3. Intelligent Service Oriented Network Infrastructure

With its *service components* (SC) and *virtual service networks* (VSN) the Intelligent Service Oriented Network Infrastructure (ISONI) [VKO+ 09] lays its focus on VIs, designed for specific services. Its main concerns are deployment and monitoring. Managers are restricted to their VSNs and may access monitoring data.

Figure 4.4 illustrates the ISONI approach. Host are managed using management agents called *ISONI exchange box* (IXB). Each host with an IXB

is an ISONI node and realises SCs as VMs. To build VNs, VSNs in ISONI terms, the IBX create OSI layer 3 overlay networks and perform access control to avoid crosstalk between the VSNs. For its high level goal of service provisioning, ISONI requires an interface description and parametrisation for each SC, which allow configuration of different types of QoS paths.

ISONI is strictly targeted at virtual endpoints, which fits a portion this thesis' requirements pertaining to endpoints and types of QoS paths. The architecture implements the concept of VIs and aims to provide adequate monitoring structures while restricting its managers to their VIs.

Developed specifically for real time multimedia services, ISONI is not designed to handle the high dynamics in VEs. On the one hand, its restriction to OSI layer 3 leaves out most of network management relevant to network QoS in VEs. On the other hand it does not perform automated management.

4.3.4. VNET

The VNET architecture has been developed to deal with migrating VMs, which may be distributed over multiple sites [SuDi 04, SGD 04]. It is constantly developed to be efficient and "lightweight" enough to be employed in a HPC environment [CXB+ 12]. While the handling of dynamics through migration in VEs is a key motivator, network QoS is not an established concern of VNET. However, it has been interfaced with VRESERVE, a network reservation system for optical networks [LSD 05].

The VNET approach at management of VNs foresees a VNET proxy on each host and all network traffic to and from VMs passes through this proxy. Proxies decide on a per flow basis how to forward data frames. To handle migration and VMs separated across multiple sites, the proxies form an overlay network and may encapsulate frames to forward them to the correct destination host; or hosts in the case of multi and broadcast.

Figure 4.5 shows an application of the VNET architecture as presented in [XCL+ 12]. The basic components are *VNET/P Core*, *VNET/P Control*

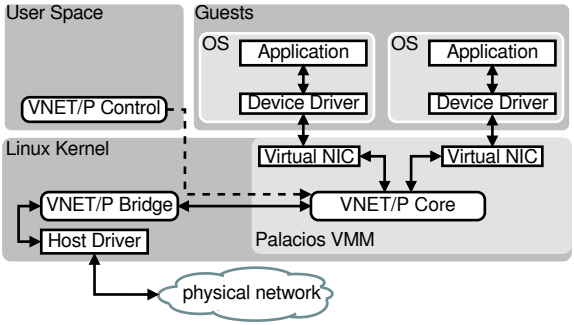


Figure 4.5.: VNET architecture [XCL+ 12]

and *VNET/P Bridge*. The suffix "/P" is specific to the implementation. VNET/P identifies an implementation as part of the VMM, while VNET/U is a user space implementation [XCL+ 12]. The Core and Bridge constitute the proxy. The Core is the service access point through which network traffic enters and leaves the VNET overlay. Instead of encapsulating and forwarding, the Core can also decide traffic is to be bridged into the physical network. The VNET Bridge is the component that forwards network traffic from/to the host. The Control component is the managing component through which the behaviour of the proxy is controlled.

In VNET, VIs are described with a domain specific language. It recognises two types of network paths, which are implemented differently. VNET can implement and monitor these topologies and correctly handle migrating endpoints. It does not recognise network QoS and is not concerned with managers of the VE. Still, VNET appears to be the most sophisticated approach to perform network management in VEs.

4.4. Discussion

The overview of related work presented in this chapter includes technologies for resource management of single subsystems and components, and a host of architectures for employing VNs to achieve certain goals. As for how to leverage the available technologies to build and manage the VNs, there appears to be a gap in the developments of network QoS management, the *integration* aspect in Figure 4.1.

From the surveyed architectures, four were developed with a wide enough scope to warrant a closer analysis and comparison to the requirements derived in Chapter 3. Table 4.2 summarises the findings. Each requirement is marked either met completely, to a certain extent or not at all.

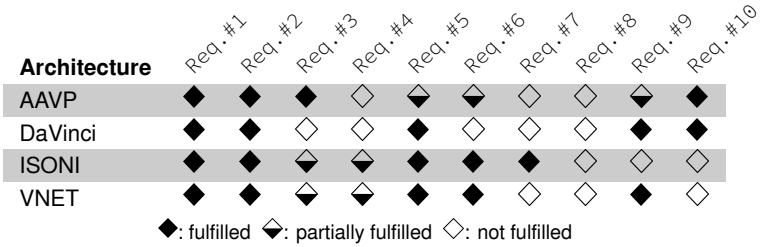


Table 4.2.: Summary of fulfilled requirements

Requirements `Req.#1` and `Req.#2` are very basic requirements, often fulfilled through the underlying technologies and prerequisites for any management architecture. Therefore all management architectures can be said to fulfil these requirements.

Requirements `Req.#3` and `Req.#4` aim at the management of virtual network paths. This is a more complex problem and also very often (the only) subject of the "Future Internet" architectures. The requirements can only be fulfilled to the extent where network path management through the Internet and through the data centre are similar. This is mostly OSI layer 3 and above. The combination with endpoints are out of scope of the Future Internet research. Requirements `Req.#3` and `Req.#4` can be interpreted to

gauge how reliable the QoS properties of paths created with the underlying technologies are.

Requirements `Req.#5`, `Req.#6`, `Req.#7` and `Req.#8` pertain to the management of VIs and their monitoring. For this comparison Requirement `Req.#5` is said to be fulfilled when there is a managed object for individual VIs. Requirement `Req.#6` must always be seen in context of the intention of the architecture is to get a reasonable statement. For instance VNET has a very sophisticated and partly automated monitoring approach to provide relevant information about its VIs. Hence the requirement is marked fulfilled, even though the entire VNET approach does not implement or monitor network QoS. Requirements `Req.#7` and `Req.#8` can be interpreted to gauge how well an architecture can be used with customer managers in dynamic environments. The fact that these requirements often remain unfulfilled correlates with Requirements `Req.#3` and `Req.#4`: as network QoS for VIs can hardly be managed, coordination efforts are neither particularly useful nor required.

Requirements `Req.#9` and `Req.#10` pertain to automated management. The fulfilment of these requirements must also be evaluated in context. Only intended and implemented management can be expected automated. Therefore the analysed architectures can fulfil `Req.#9` and `Req.#10` even though they completely lack other aspects, for instance coordination of concurrently performed management.

Having applied the correlation between fulfilment statements, Table 4.2 shows that VIs cannot be managed for all use cases identified in Chapter 3 and that multiple managers performing management is hardly addressed at all. Building an architecture without these short comings has to accomplish correspondingly more, in order to fulfil the automation requirements as well.

A comprehensive approach at network QoS management in VEs does not exist, even though many management approaches based on a fully controlled VE exist. The remainder of this work develops a management architecture for network QoS in VEs and close this gap between capabilities of available technologies and intended applications.

Chapter 5

Performing management in virtualization environments

The goal of this chapter is identifying prerequisites, and requirements on the behaviour of a management system. Prerequisites must be fulfilled by subsystems not developed as part of this work. Requirements on the behaviour carry design decisions, rather than defining necessities for achieving network QoS management.

The resulting requirements shape the approach for developing the management architecture in Chapter 6. This arranges network QoS management with other management performed in VEs, to realise the herein developed architecture's sustainability.

To gain a complete view, a life cycle for VIs is developed first. It allows to derive management use cases and, in a further step, functional and non-functional requirements for performing management in VEs. This method is also used in [MNM Dreo 02]. The resulting requirements are then classified as prerequisite, or behavioural requirement.

First, Section 5.1 takes another look at the services introduced in Section 3.2, to obtain a better understanding of the intentions of managers whose actions affect network QoS. This serves threefold:

1. Identifying intentions enables the identification of operations on VMs and virtual topologies, performed in Section 5.2, leading up to the life cycle in Section 5.2.3.
2. Identifying managers enables structuring the identified use cases in Section 5.3 by the actors involved.
3. Identifying managers is part of developing the organisation model in Section 6.3.2.

Section 5.3 derives requirements using the same method as applied in Section 3.4, before Section 5.4 summarises this chapter.

5.1. Managing virtualized services

Management is performed by different users with different goals. This section analyses the services `Svc.#1`, `Svc.#2`, `Svc.#3` and `Svc.#4`, introduced in Section 3.2, to refine their management into tasks which help to identify management roles and use cases. In contrast to Chapter 3, all management operations that affect the VMs' and network's capacities to provide and deliver services are relevant. These are management operations performed on of the following:

- the services themselves, or
- VMs and networks, or
- virtualization facilities, or
- physical hardware.

There are three generic roles involved with service management: the *customer*, the *provider* and the *user* [GHHK 01, GHKR 01]. The customers' overall goals are maintaining services, while the providers work to enabling services and service management. The users is the intended audience for provided services, but perform management solely through operating services, if at all. The following identifies these generic roles and their specific goals in each service. This creates specialised roles for management tasks in VEs.

Services are always requested by customers. It is assumed that all planning and negotiating is finished, before entering the provisioning phase. The provisioning phase is the first phase where concrete management is performed on the VE and relevant management operations occur. This is the starting point for each service description. The discussed services are:

- `Svc.#1` Attended hosting of virtual servers
- `Svc.#2` Unattended hosting of virtual servers
- `Svc.#3` Project infrastructures
- `Svc.#4` Personal desktop

`Svc.#1` Attended hosting of virtual servers The provider creates and maintains the server, while the customer uses the software installed on the server. During the provisioning phase, the provider creates the server as a single VM, connects it to one of two available access networks and installs the required software on the server.

After that, the customer uses the server as intended. For performance management as described in Section 2.2.2, the customer uses specific management information and monitoring data. The server, with its OS and installed services, provides monitoring data about the current load and activity. Data obtained about the network uplink provides information about interactions with other systems. Operating and managing services implicitly controls the amount of resources services would use, however, the customer does not control the amount of resources the VE provides. This task is delegated to the provider through an explicit request.

`Svc.#2` Unattended hosting of virtual servers As a variant of the hosting service, the tasks are very similar to the attended hosting service `Svc.#1`. The distinct feature of the unattended variant is that the customer has full control over the operating system. This enables the customer to use OS tools to manage resource allocations to processes. In this form, resource allocation is managed by the provider for the virtualization facility and by the customer for the VM. Through this, the customer is responsible for the monitoring data obtainable from the operating system.

Svc.#3 PROJECT INFRASTRUCTURES This service also has the role of a provider that controls the VE and reacts to customer requests. The customers are granted bulk amounts of resources, which they may arbitrarily allocate to their VMs. To perform resource allocation, customers are granted access to manage the VE directly. Moreover, on the customers side, primary and secondary users are discernible. According to the service description in Section Svc.#3, there are main users that can define subordinate users. The main users control the entire virtual topology and assign resource shares to subordinate users. Also, only main users interact with the provider to attain additional components and resources. The subordinate users are in charge of a subset of a virtual topology and work with resources shares that are fragments of what the provider allocated to the customer. Their management domains are created within a virtual topology, therefore they are labelled secondary users.

In contrast to the services Svc.#1, Svc.#2 and Svc.#4, withdrawing a VM does not mark the withdrawal of the entire service. In this case, the provider must reorganise the (virtual) network and make the released resources available to the customer again.

Svc.#4 PERSONAL DESKTOP With a direct uplink to the Internet and integration into a local network and the remote access server, this service has the broadest variety of attached networks and available access methods. Setting up a VM for a personal desktop is very similar to setting up a VM for an (un-)attended server. On the other hand, setting up the networks is more complex. Access over the remote access server requires a corresponding configuration of the VMM. The customer is involved during the provisioning phase a VM may be connected to the local network, which is within the customer's management domain. Once the virtual desktop has been provisioned, the customer has full control over the OS, analogous to unattended virtual servers. During the operating phase management of the LAN remains a task where provider and customer interact, as each is responsible for parts of the network. This can be regarded as a new role on the customer side of the service.

Actors performing network QoS management

The descriptions of services and the identified tasks allow a refinement of customer and provider roles, focused on certain aspects of performance management and certain areas of the VE. As network QoS management is not actually performed in the real world scenarios, it is assumed, that every role performing QoS management and resource allocation on VMs, also fulfils similar tasks on VNs and network components. Further, it is assumed that performance management of services requires management information from the corresponding VI's components and therefore every management role involved with performance management needs to interact with the anticipated management system for network QoS management in VEs, at least to obtain monitoring data. The identified roles described in the following are:

- User
- Primary customer
- Secondary customer
- Local admin
- Virtualization provider
- Hardware provider

`User`: Interacts with the services provided on VMs. A user performing management only controls the actual service, not the VM's OS nor any other component. A service may be adaptable to the available resources. In this case the Users need information about resource utilisation of the VM and general load on its host within the VE. This information is for example used to early detect bottlenecks and threshold crossings.

`Primary customer`: Interacts with the management system and the provider. The primary customer knows about the entire VI, its DTEs, DCEs, topology, as well as the associated resources and actively performs performance management. Once resources have been assigned to a customer's VI by the provider, primary customers may allocate and reallocate them to existing virtual components directly, through the management system. Especially after a VI's initial provisioning, primary customers may request additional resources and virtual

components from the provider. Also, primary customers manage secondary customers, to delegate resource shares and responsibilities for virtual components or subtopologies.

`Secondary customer`: Interacts with the management system to perform performance management analogous to the primary customer. However, secondary customers do not interact with the provider and usually, their management domain is a subset of what was provided by the provider to the primary customer.

`Local admin`: Virtual desktops are an example for services that are integrated into a customers domain, as opposed to externally provided services, that exist separated from the customers domain. Integrating a virtual desktop into the customers LAN, is an isolated supporting task and not directly related to using the service. Therefore this task can be delegated to a separate role, the local admin, who is in charge of ensuring that the virtual desktop can interact with any other machine on the LAN in the same way, as local physical desktops can. As the provider has no means of performing management in the customers domain, the local admin plays an integral part during the provisioning phase, to initially connect the virtual desktop with the local LAN. After that the local admin ensures the connection between LAN and virtual desktop performs within the specified parameters.

`Virtualization provider`: The virtualization provider role performs management on the VE to provide VIs and guarantee QoS. It has full control over VMs, VMMs and the network components interconnecting the virtualization hosts. VMs, network components and topologies are created according to the customers' requests, initially during the provisioning phase and later on as well. With global knowledge of all hosts and virtual components the virtualization provider controls virtual component placement and resource allocation. While the main goal is providing infrastructures according to the customers' requests, the virtualization provider also has full responsibility for the hosts and VMMs, which especially comprises planning and performing maintenance work.

`Hardware provider`: In large scale environments, management of the IT infrastructure is separated into many distinct roles. Especially network management and physical network components are responsibilities of dedicated roles. To match the VEs of the chosen real world scenarios, having two separate provider roles is sufficient. The responsibilities of the hardware provider can be summarised as everything related to networks and physical hardware, that is not covered by the virtualization provider. As a consequence, virtualization provider and hardware provider both manage the physical network, to create and manage virtual topologies. While the virtualization provider's view is limited to network components that constitute part of the VE, the hardware provider is responsible for all physical network components.

5.2. Management operations performed during life cycles

Section 2.2.2 introduces life cycles conceptually, to identify the goals and tasks pertaining to network QoS management. This section develops a specialised life cycle for VIs in VEs, to derive management use cases in Section 5.3.

When performing management on network paths, a single management operation often implies many management operations on different components. By first analysing management operations on virtual components and infrastructures, it becomes possible to determine implicit operations and dependencies between operations when analysing management use cases. This yields more refined and complete requirements. Services employed by users are always assumed capable of fulfilling QoS requirements, if the VE provides sufficient resources.

As life cycle descriptions of VIs are not available, Section 5.2.1 starts out with an individual VM's life cycle, then Section 5.2.2 advances to a life cycle for virtual topologies, before Section 5.2.3 presents a life cycle for VIs.

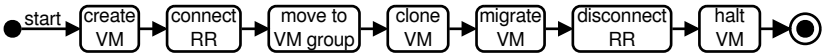


Figure 5.1.: Example life cycle of VMs, introduced in [Metz 09]

5.2.1. Operations on individual virtual machines

Figure 5.1 shows an exemplary life cycle for a VM, already published in [Metz 09]. This life cycle is focused on management operations that affect a VMs I/O capabilities. In the Figure, RR stands for remote resource, a piece of a VM that is not local to the hosting computer, e.g. storage. A VN with allocated resources to fulfil certain QoS requirements and spanning across multiple hosts can also be considered a RR, as it is not confined to a VM's host. Figure 5.1 shows an exemplary life cycle for a VM, first introduced in [Metz 09]. This life cycle is focused on management operations that affect a VMs I/O capabilities. In the Figure, RR stands for remote resource, a piece of a VM that is not local to the hosting computer, e.g. storage. A VN with allocated resources to fulfil certain QoS requirements and spanning across multiple hosts can also be considered a RR, as it is not confined to a VM's host.

The clone and migrate operations are intuitive examples that the depicted management operations do not have to be performed in the shown order and that some operations may be performed several times during the same VM's life cycle. Yet, it is a complete list of management operations on VMs, directly affecting their I/O subsystems. Networking is part of the I/O subsystem and networks can be considered remote resources. Therefore, the illustrated management operations in Figure 5.1 have network QoS relevant aspects. The individual operations described in the following are:

- create VM
- connect RR
- move to VM group
- clone VM
- migrate VM
- disconnect RR
- halt VM

`create VM`: Regarding the individual VM instance, *create VM* is performed only once. Creating a VM means selecting a host and using shares of its resources to create and activate a VM according to its specifications. Once created the VM is an active component within the VE, which in most cases will proceed with booting an OS. The boot process may be stalled until required RRs are connected.

`connect RR`: The VM is connected to a RR within the scope of the VE. This includes the creation of data paths within a VMM so that the VM can access its RRs. Depending on the current state of the virtualization infrastructure, this also requires the creation of network paths. For the scope of an individual VM, a RR is provided by the VE so that it can be used after a VM's creation.

`move to VM group`: A VM instance is arranged with its environment and is given to a customer from a management perspective. This includes the application of customer or service specific rules and policies, especially network QoS attributes and requirements. At the end of this operation the VM's configuration is such, that it matches the service's description and requirements. This operation is always performed when changes to the VM are necessary, to ensure its contribution to the service and to avoid unwanted side effects of changes. During this procedure it may be concluded that it is not possible to provide the VM according to the specification of the service and the customer's requirements. In this case a suitable host must be found and the VM be migrated, before all rules and policies can be applied. Eventually the VM is either ready to provide the service for which it is intended, or it is concluded, that the service cannot be provided as has been specified. In this case action is required at the scope of service provisioning and service management.

`clone VM`: The VE creates an identical copy of a VM, its current state and its RRs. While there is no persistent effect on the original VM, the procedure itself accesses a VM's assets which has the potential to influence its QoS.

`migrate VM`: A VM, in its current state, is transferred to another host.

This implies a change to every connection to RRs and the VM must be arranged with its environment again. Depending on the VM's displacement within the physical set-up there are more or less implications on existing network paths. After a migration, completely different physical resources are used to realise the VM and the original resource, used before the migration must be freed. During the procedure there are severe effects on the QoS a VM can achieve.

`disconnect RR`: A VM stops using a RR, the connection is severed and its resources freed.

`halt VM`: A VM is deactivated. This usually includes an OS shutdown and remaining connected RRs are disconnected. This VM's instance is removed from the VE, so that it does not block any resources any longer. For *halt VM* many implementations can be found, each with different properties regarding which and how many resources are freed, which has implications on the following *create VM* that may be performed.

The main concern of a VM's planning phase is preselecting suitable hosts for *create VM*. As the internals of VEs are usually hidden from customers, VMs have no explicit negotiation phase.

The provisioning phase includes all necessary operations until a VM can provide its service relevant to the customer. Initial provisioning comprises creating a VM, connecting RRs and moving a VM to a group. A VM also enters a provisioning phase when migrating to a new host, as there are new resources to be allocated to that VM. In this case the VM has already been created, but RRs may need reconnecting and the VM must be arranged with its environment.

Direct changes to VMs are connecting and disconnecting RRs, as well as reconfigurations, which are part of the operation *move VM to group*. Further, a VMs capabilities to provide services may be impeded by its cloning and migration. These operations are also part of a VM's change phase, even though both operations neither change the VM itself, nor how it may be used as part of a service. Consequently the operation phase is characterized by a VM being put to use by customers and cloning and migration

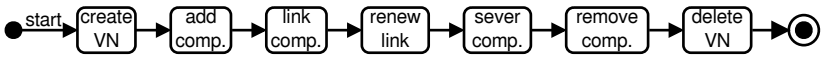


Figure 5.2.: Exemplary life cycle of a virtual network

are not performed. Withdrawing a VM from the VE is disconnecting all RRs and halting the VM.

A management system must expose certain functions to enable performing the introduced management operations on individual VMs.

5.2.2. Operations on virtual topologies

The previous Section 5.2.1 analyses management operations on individual VMs, which are the endpoints within VIs. This section is concerned with management operations provisioning and withdrawing VNs in VEs.

Figure 5.2 shows an exemplary life cycle for VNs, analogous to the life cycle presented for VMs in Figure 5.1. The management operations need not be performed in the presented order and most operations will be performed multiple times throughout a VN's life cycle. These operations control a components integration into the topology. The individual operations described in the following are:

- create VN
- add component
- link components
- renew link
- sever components
- remove component
- delete VN

`create VN`: This first operation creates the VN, so that components can be added and links between components can be created. At this time no components are associated with the VN. This operation is concerned with attributes all components and links within a VN share. Prominent examples for information is prepared before links and components are be added, are:

- address space
- responsible management roles
- management access
- monitoring strategies
- generic QoS requirements

`add component`: This operation performs membership management as a prerequisite for linking two components. VNs are isolated subtopologies and there can only be links between components belonging to the same VN.

`link components`: This operation connects two components directly, so that they can exchange data, within the scope of the VN. The resulting link is an abstract managed object and management views focusing on the VI/VN will always show the direct connection between two components. The link's actual implementation and placement, however, depends on its endpoints' placements and the QoS requirements associated with the endpoints and link. Also, a links implementation may change over time due to migration.

`renew link`: This operation is meant to handle changes to the VI and VN, especially concerning migration. In VEs, most links are terminated by virtual components, which means an endpoint's network access is implemented by a VMM or other kind of virtualization facility. Especially VMMs, pieces of software, often have many approaches to implement network access. Which method is chosen may depend on the host and its equipment, the current load, and its position in the physical network topology. All three aspects may change over time and a link's implementation must be renewed, to ensure it still satisfies QoS requirements.

`sever components`: This operations removes a previously created link. This severs two components within the scope of a VI/VN and also releases the resources currently occupied by a link's implementation.

`remove component`: This operation is the complementary group membership management operation to *add component*. A component to be

removed from a VN may not terminate links belonging to that VN. Existing links must be severed first.

`delete VN`: This operation completely withdraws a VN from the VE. All existing links are severed and the components removed from the VN. Finally, abstract resources, such as address space, which were previously allocated to a VN are released.

Analogous to VMs in Section 5.2.1, VNs are part of a service and therefore its planning phase mostly consists of preselecting resources and there is not an explicit negotiation phase.

A VN's provisioning phase, can be crafted arbitrarily complex: Every host has a limited capacity to host virtual components. For any metric that may be chosen to measure a host's capacity, the complexity of determining an optimal set of resources to implement and place a virtual component is NP-hard, known as the knapsack problem [MaTo 06]. With further restrictions on placement, e.g. grouping of VMs, the problem becomes NP-complete (NPC).

Common use cases do not constantly saturate network links. Providers exploit this through *over provisioning*, allocating more resources to VNs than the physical network is capable of providing. The benefit is more VMs and VIs can be realised without deploying more resources, i.e. hosts, at the expense of increased planning costs. This leaves ample opportunities for high dimensional planning and decision making, during VN provisioning phases.

Changes to VNs are the addition or removal of links or components. Smaller changes merely add links and components to the VN, without affecting already placed entities, so that the VN immediately return into its operation phase. Larger changes my trigger or even require a new placement of some or even all VN components, in order to meet QoS requirements. In such cases, the VN enters a new provisioning phase to newly place (and replace) the entire VN within the VE. When a VN is withdrawn, all links are severed and the VN is deleted.

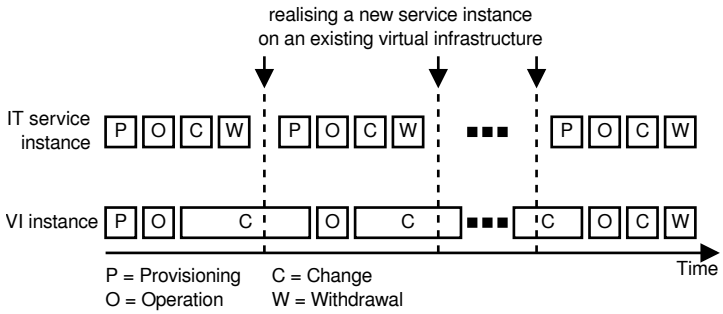


Figure 5.3.: Alignment of life cycle phases of a VI providing several user faced IT service instances

5.2.3. Life cycle for virtual infrastructures

The previous sections introduce and describe life cycles for high level IT services, individual VMs and isolated VNs. VIs are the missing link that combine VMs and VNs to platforms suited to provide high level IT services. This purpose arranges the life cycles with each other, as depicted in Figure 5.3. A VI is provisioned with the first user faced service and withdrawn with the last used faced service.

Following the concept introduced in Section 2.2.2, the six life cycle phases introduced in the following are:

1. planning phase
2. negotiation phase
3. provisioning phase
4. operation phase
5. withdrawal phase

Planning Phase Before a VI can be provisioned, a topology is specified, that is suited to fulfil a customer faced services requirements. The planning phase includes a translation of service specific QoS requirements to network QoS requirements, that can be achieved and measured within the VE.

Negotiation Phase The negotiation phase determines how a VI is implemented and placed within a VE. This may, for instance, include specifications that a virtual switch is to be implemented using a specialised VM, rather than being implemented through the VMM. A reason for such a specification could be, that a customer requires more control over the switch, than the VMM implementation could provide. As part of the planning or negotiation phase the monitoring strategies are specified. The monitoring strategies determine how detailed and frequent a VIs state is gauged. This in turn determines how quickly and accurately QoS management can be performed.

Provisioning Phase With a clear specification of what needs to be provisioned by the VE, the last and most visible effects of a VI's provisioning phase are the creating of the initial VNs and VMs. To enable these last steps, the results from the planning and negotiation phases must be put into policies, which will guide creating VNs and VMs within a VI. An example for such a policy is selecting one of many methods to realise a virtual component. Another example could be restrictions on which hosts VMs may be placed. Such restrictions, which need to apply to all VMs of a VI, are formalised in VM groups, to which VMs are "added" during their provisioning (see Section 5.2.1).

Operation Phase During a VIs operation phase the VI is used as a platform for the user faced services, often operated by customers. While there may be changes to the VE, e.g. more VMs share resources of hosts, the VI itself remains steadily within the VE. The management performed during this phase pertains to detecting changes of the environment which may require action to ensure the VI serves its purpose.

Change Phase The change phase of a VI includes modifications of QoS and network QoS parameters as well as migrating individual VMs or entire parts of the VI. Depending on the current configurations, especially network configurations, a migration is quick and the VI can directly revert to

the operation phase. In general, such a change to a VI triggers *renew link* operations within VNs and *move to VM group* operations on VMs. Especially the *renew link* operation may result in a required provisioning phase, where network paths are created to fit the VI. This could entail the creation of new VN components as well.

Withdrawal Phase Withdrawing a VI is a task which mostly coordinates the decommissioning of the user faced services, the VMs and VNs. When all operational components are withdrawn, the VI itself can be removed from the system and all related policy sets discarded.

Conceptually, a VI can be regarded as a grouping of VNs and VMs and as a container to create a compound platform, that can be referred to as a single entity for management purposes. Operations performed on VIs are therefore group membership operations without a directly visible effect on the VE.

5.3. Prerequisites and behavioural requirements

The life cycles defined in Section 5.2 describe the management that is performed in a VE, before, during and after a VI is used by customers and useful to users. By putting this management in the context of how VEs are operated in the real world examples, analysed in Chapter 3, the use cases for a management system for network QoS management in VEs can be derived.

First, section 5.3.1 identifies and discusses use cases pertaining to managing roles and privileges. The management performed within these use cases is not part of the core management functions to facilitate network QoS, as described in Section 2.2.2. Therefore, detailed analysis of these use cases are out of this work's scope. After that, Section 5.3.2 provides an overview of the identified use cases and requirements, analogous to Section 3.

The full use case analysis can be found in Appendix A. The full descriptions of derived requirements can be found in Appendix B.

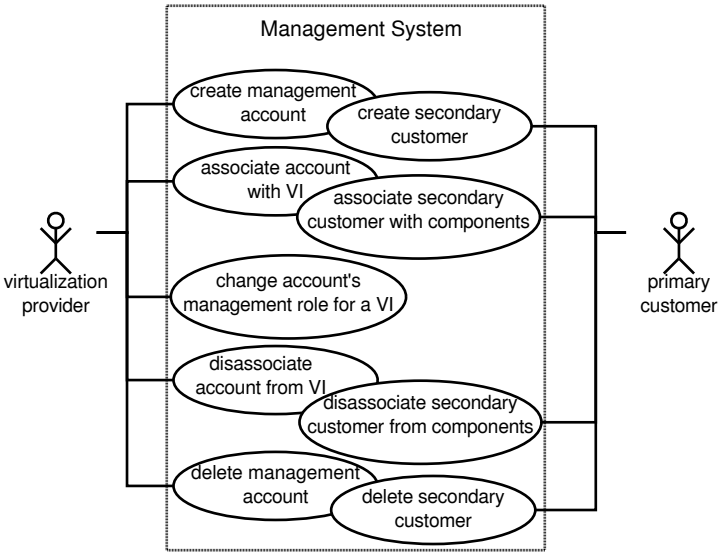


Figure 5.4.: Use cases for managing roles and privileges

5.3.1. Use cases pertaining to roles and privileges

Customers perform management for as long as they have associated VIs within a VE. Managers concerned with creating and managing accounts, assigning roles, privileges, resources and VIs are *virtualization managers* or *primary customers*.

The corresponding use cases for managing roles and privileges are summarised in Figure 5.4. The nature of these use cases describes how users of the management system become managers responsible for VIs. Overlapping use cases indicate that their general motivation is the same. Their difference arises from the fact, that the primary customer is operating within a limited scope, while the virtualization provider operates globally for the entire VE.

A manager is associated with VIs or components for which they take responsibilities in the form of a predefined role.

- Management users may perform management on all kinds of components of a VE.
- Management users are restricted to associated VIs.

The bare identification of roles and use cases also yields non-functional, characterising requirements on a management system (and therefore architecture) for network QoS management in VEs:

- Responsibilities for VIs can be split up in the form of sets of components managed by different users. This may result in dependencies and cascading resource allocations schemes. For example, the virtualization provider allocates primary customers resource shares, who in turn assign smaller shares to secondary customers, who assign resources to components and links. The management system must track and account for such cascades in order to provide accurate management information.

- A VI's life cycle may outlast many sequential VMs' life cycles. This has implications on how long, relative to the services life time, assignments are valid. Therefore, when assigning responsibilities and resources, an explicit distinction between VI or component must be made.

5.3.2. Use cases pertaining to networks and components

The derived use cases are structured by the goals of managers, identified in Section 5.1:

- Virtual infrastructures
- Virtual links
- Host systems
- Virtual components

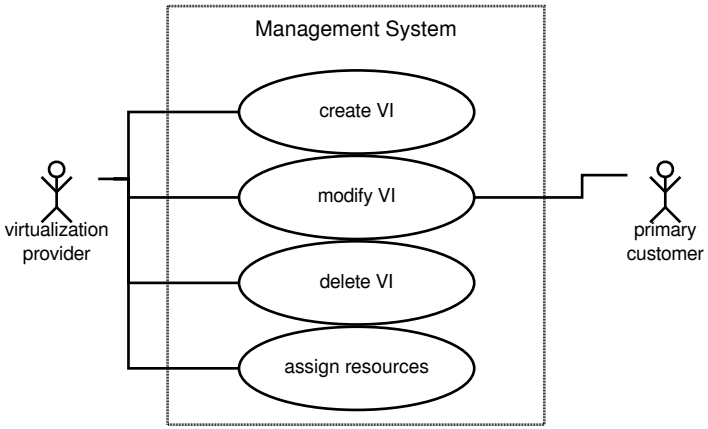


Figure 5.5.: Use cases relating to virtual infrastructures

VIRTUAL INFRASTRUCTURES Figure 5.5 depicts the use cases directly related to VIs. While all actors perform management on components and within VIs, the virtualization provider is the only actor performing management in the beginning and at the end of VIs' life cycles. The VI use cases foremost include or imply steps from other use cases or trigger other use cases. For example creating a VI includes the creation of many VMs and links between them.

VIRTUAL LINKS Directly managing VN links has the most direct effect on network QoS in VEs. The related management use cases are mostly triggered as part of a larger task when creating components or complete VIs. Figure 5.6 shows all link management related use cases, where users explicitly interact with the management system to change a virtual topology or aspects of individual links.

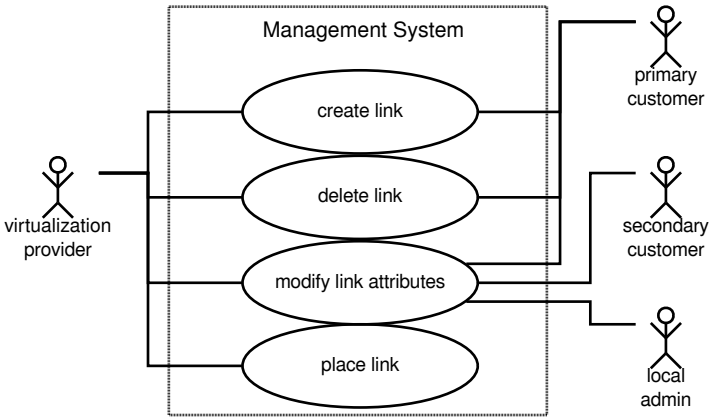


Figure 5.6.: Use cases relating to virtual links

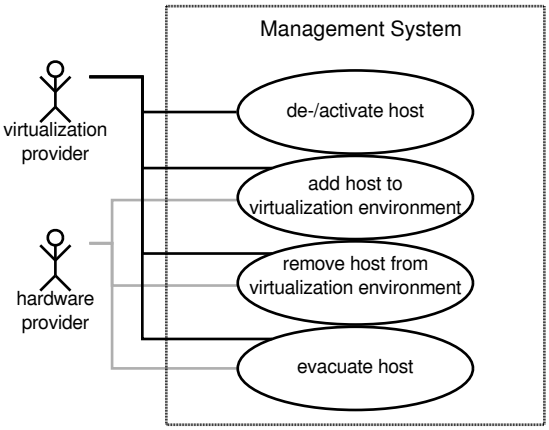


Figure 5.7.: Use cases relating to hosts

Host systems Part of the dynamics triggering migrations and new placements are changes in the physical topology. Physical network components and hosts are added or removed from the VE either as a planned change, or as part of fault management. The previously described capabilities for placing VIs and their components allow such changes without severely interrupting the user faced services, but may still affect network QoS. Figure 5.7 includes the relevant management use cases pertaining to physical virtualization hosts.

Please note, that virtualization hosts are never explicitly subject to management when managing individual VIs. The same observation can be made for physical network components. Physical components are introduced to and removed from VEs. All management performed during their operation phase to realise network QoS of virtualization infrastructures, is implicit. This is a main motivation for the approach introduced in Section 6.1.

Virtual components The use cases for management of virtual components have been derived in Section 3.4 and need not be derived from life cycles.

The full list of unique functional and non-functional requirements is:

- Automatic adaptation of links between virtual components
- Control over physical components
- Control over resource allocations in virtual and physical components
- Control over VMMs
- Control over virtual components
- Explicit release of allocated resources
- Explicit migration of individual components
- Full information about all available and deployed resources
- Full information about the current placement of components
- Information on components belonging to VIs
- Information on currently used resources by components
- Migration of virtual components
- Monitoring structures for VN properties can be set up
- QoS requirements are specified w.r.t. the selected components or VNs

Some requirements directly match the requirements described in Section 3.4, and must not be formulated as new requirements. Table 5.1 provides an overview of the requirements refined from the above list. The full specifications can be found in Appendix B.

Req.-ID	Title
Req.#11	Control over physical hosts
Req.#12	Control over virtual components
Req.#13	Control over VMMs
Req.#14	Migration of virtual components
Req.#15	Full information about currently used resources by components
Req.#16	Full information about available and deployed resources
Req.#17	Release allocated resources
Req.#18	All components may be subject to management
Req.#19	Full information about the current VI placement
Req.#20	Reverse mappings from components to VIs
Req.#21	Resource allocation within a hierarchy of management users
Req.#22	Management users can be tied to life cycles or life cycle phases

Table 5.1.: Summary of prerequisites and behavioural requirements

5.4. Summary

A main result of Chapter 3 is recognising, that due to shared physical resources, network QoS may be affected by any change to resource allocation and resource taxation within VEs. This chapter describes and analyses a VI life cycle, to identify such side effects of not QoS specific management.

As a result, two additional sets of requirements were derived. One set are prerequisites, which demand management capabilities and properties of VEs so that network QoS management can be performed reasonably. The other set describe behaviours of a management system, which can also be seen as supporting capabilities, which are used during multiple phases throughout VI life cycles. In anticipation of Chapter 6, the identified behaviours can be core aspects of functional components in the intended archi-

tecture. Similarly, the identified management roles in Section 5.1 contribute to the architecture's organisation model.

This chapter marks the end of preliminary work towards developing a network QoS management architecture for VEs.

Chapter 6

Architecture

Continuing the unified software development process, this chapter develops the intended architecture for network QoS management in VEs, such that its implementations are suited to fulfil the identified requirements.

Section 6.1 introduces the core idea to have one continuously repeating *management loop*, modifying the VE to adhere to the management performed by users on abstract VIs. The loop is refined into management tasks, leading to the architecture's functional model in Section 6.3.3 and communicating entities in Section 6.3.4.

The management loop foresees automated refinement of management on abstract VIs and networks into concrete implementable management operations for all managed physical and virtual components. Section 6.2 specifies a refinement procedure for this task, contributing a main component to the functional model and type specifications for links and endpoints to the information model in Section 6.3.1.

Section 6.3 contains the management architecture. While the information, functional and communication model obtain their most input from Section 6.1 and Section 6.2, the organisation model's main input are the identified management roles in Section 5.3.1

This chapter is summarised in Section 6.4, before the following Chapter 7 first discusses the architecture, before presenting a prototypical implementation and evaluation.

6.1. Continuously improving network QoS in virtualization environments

This Section targets the superordinate questions how to cope with dynamics and how to enforce network QoS in VEs, stated in Section 1.4.

The core idea is to achieve QoS through automated management loop, constantly working towards configuring the VE so that all VIs' QoS requirements are met. The loop is driven by two representations of the VE:

`target configuration` … all active VIs, with their QoS requirements fulfilled, and

`current configuration` … the VE's current state with up to date monitoring information.

The target configuration constitutes the goal specification for the VE configuration. It is an intermediary representation between abstract VIs and concrete configurations that can be rolled out onto components without further refinement or adoption. It contains the complete specifications of all active VIs, as managed by users, and may also include requirements and constraints on VI realisation and placement.

The current configuration is a snapshot of the VE's current state. It consists of all knowledge obtainable from the VE, e.g. realised virtual components and their placement, the state of physical hosts and their hardware, resource allocations and implemented networks.

The basic management loop is divided into four phases, as illustrated by Figure 6.1:

1. an acting phase, formulating or refining the target configuration, especially by extracting and deriving QoS requirements from VIs,

2. a planning phase, devising a series of configurations steps to achieve the target configuration,
3. an execution phase, managing components to implement the configuration steps,
4. a verification phase, where monitoring data is generated and collected to assess whether the target configuration has been achieved.

The concept adapted here is known as the Deming cycle [Walt 88] and is often known by its repeating four phases *Plan*, *Do*, *Check*, *Act*. The above adaptation starts out with an Act phase. Figure 6.1 includes interactions with managers (labelled "Actor") and the individual components in the VE.

The best way to handle virtualization's dynamics, is to perform every change as a change to the target configuration during the Act phase. The management system automatically devises a plan to implement the change without negatively affecting network QoS, or deny the change, if no such plan can be developed.

If changes are applied out of band, i.e. not through changing the target configuration, problems with delivering network QoS will be detected during the verification phase and addressed by the next planning phase. Repeating these phases over and over while allowing the Act phase to finish without a change to the target configuration, results in an enforcement of QoS settings, as long the planning phase can devise a suitable configuration for the given VE.

The following Sections introduce the four phases of the management loop in detail. This has been published previously in [Metz 14a]. As per Figure 6.1, the four phases are:

- Definition of a target configuration
- Development of a configuration strategy
- Effective component management
- Collecting Performance Data

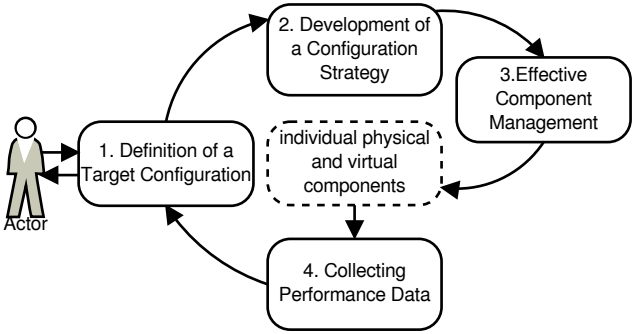


Figure 6.1.: Management phases

6.1.1. Definition of a target configuration

The information about VIs largely originates from the actors performing management in the identified use cases. Therefore it is part of this task to take management, not performed as part of the management loops, into account.

Figure 6.2 shows the three tasks of this phase:

- facilitating user interaction
- evaluating monitoring data
- coordination and event handling

All functionality pertaining to user interaction is combined into a functional component with the task *facilitating user interaction*. Users interact with the components implementing this task. They can obtain their VIs current state and configuration and may perform the use cases named in Section 5.3. Their actions modify the goal state of their VIs and therefore implicitly the target configuration. The current state is partly measured as monitoring data obtained from shared components and must be processed and information about individual VIs extracted. This is a result produced from the task *evaluating the monitoring data*.

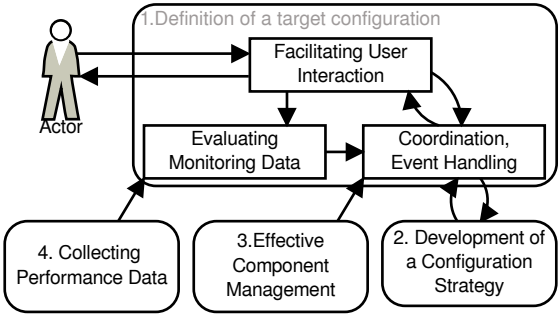


Figure 6.2.: Tasks assisting the definition of target configurations

Processing monitoring data is an independent task, because the evaluation of monitoring data may yield important information that may need immediate action, for instance, if a trend towards resource shortage with the current configuration is detected, a requirement for unallocated spare resources could be added for certain components, without involving user interaction.

Changes originating from evaluating monitoring data, as well as though management performed by users, are integrated into the target configuration as part of the *coordination and event handling* task. This task is the information hub and the target configuration is one of its products. It is also responsible for events as

- reactions to the evaluation of monitoring data,
- responses to problems during the following planning and execution phases, or
- consequences of explicit user requests.

These events may be automatic changes to the target configuration, or requests for information from other components, or the management user. Ultimately, this task marks the end of the act phase in the Deming cycle and provides the planning phase with a new target configuration.

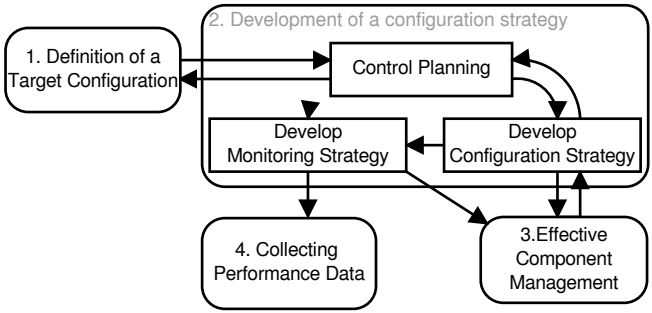


Figure 6.3.: Subtasks of developing a configuration strategy

6.1.2. Development of a configuration strategy

The problems of transitioning a system from a given current state into another desired state and minimising the required resources when implementing network QoS are not specific to virtualization. Developing a configuration strategy, i.e. implementing VIs with network QoS requirements in a VE can therefore be seen as an optimisation problem.

Developing a configuration strategy has a high dimensional problem space. There are

- different types of QoS requirements,
- changing numbers of network links,
- varying loads on physical hosts, and
- migrating components.

The planning procedure is correspondingly complex, if the configuration for optimal placement of virtual components and links is demanded following Bellman's optimality principle [Bell 57]. As far as operating VEs and performance management are concerned, any configuration that fulfils all VI constraints and requirements is sufficient.

Figure 6.3 shows the three tasks of this phase:

- develop configuration strategy
- control planning
- develop monitoring strategy

The task *develop configuration strategy* is representative for arbitrarily complex approaches that can generate an ordered list of concrete configuration requests for components. Sensible approaches generate lists such that the resulting configuration of the VE is closer to the target configuration. The information the planner may use are the architecture's model of the VI (presentation domain in the information model, cf. Section 6.3.1), the current and target configurations and it may also interact with individual components to obtain information directly.

The task *control planning* assumes an overseeing role, to monitor and control especially the development of a configuration strategy. It's core objective is to continue the management procedure. If a suitable configuration has been developed it triggers the *develop monitoring strategy* task and ensures the devised configuration steps are passed on to the execution task. It also handles any deviations, for instance problems reported back from the execution task, or if the planner is unable to develop a configuration strategy, or if the planner does not return any result. Figure 6.3 shows bidirectional interaction between tasks where feedback loops ought to be in place to ensure the management procedure is driven onwards.

Monitoring strategies can result in more accurate and detailed information, when specifically tailored to the placement of VI components and links. The exact placement is defined as part of the configuration strategy. To use it to develop a monitoring strategy, developing a monitoring strategy is a task that must follow after developing a configuration strategy.

6.1.3. Effective component management

This task is concerned with implementing the configuration steps devised previously on the components.

Figure 6.4 shows the three tasks of this phase:

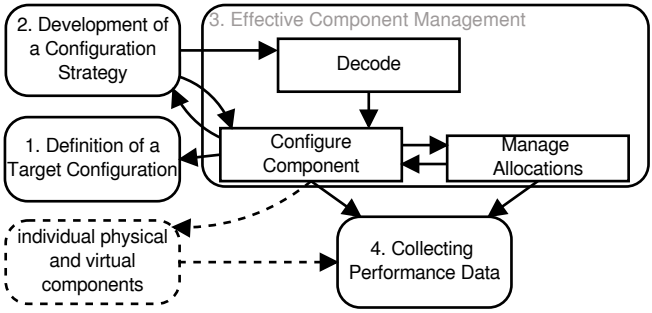


Figure 6.4.: Tasks to implement configurations on components

- decode
- configure component
- manage allocations

The feedback loop between *configure component* and *development of a configuration strategy* represents the planners capability to interact with components to support the planning process. The unidirectional information flow to the *decode* task is the list of configuration steps for individual components that is to be executed.

Decode is the final adaptation task to translate the high level VI configurations to instructions for the managed components. Every component has capacities in the form of resources and mechanisms to implement network QoS. Employing these capacities depends on the component and sometimes on the concrete product. For example, a VMM is in full control over a host's CPU(s) and assigns CPU time intervals to individual VMs, whereas switches and routers most often measure their resource assignments in PDUs or bytes per interval. When a virtual switch or router must implement network QoS, the single configuration step is decoded into two management operations:

- the allocation on the virtual component and
- CPU time allocation on the VMM.

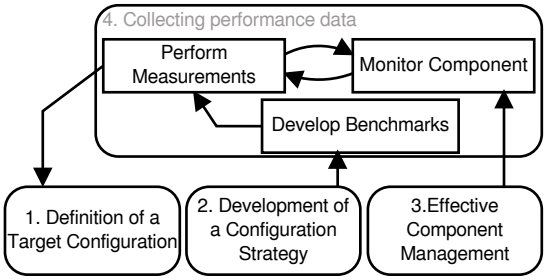


Figure 6.5.: Tasks to gauge the new current configuration

Both steps must be performed, so that the virtual component has the resources to actually fulfil the QoS requirement.

The tasks *configure component* and *manage allocations* are implemented for each managed component separately. Configuring the component means interacting with and performing management operations on the component. This is the task where the VE is actually changed. The corresponding resource allocations and QoS promises that are implied with the performed management operations are recorded and managed as the separate task *manage allocations*. Its objective is to make the component provide adequate monitoring data.

6.1.4. Collecting performance data

The list of performance management activities, introduced in Section 2.2.2, shows that pure performance management begins when networks and paths with QoS properties have been deployed. Figure 6.5 shows the three tasks of this phase:

- monitor component
- develop benchmarks
- perform measurements

There are three discernible types of monitoring data that can be obtained from the running system:

- performance data provided by the individual components,
- records of past management operations from the components and the management system, and
- results from actively benchmarking the configured network.

The task *monitor component* is providing performance data and records about management operations from individual components. This task is implemented for each managed component separately, analogous to the configuring and managing task of effective component management.

The *develop benchmarks* task formulates tests that can be run on the active system to challenge the QoS guarantees. It also specifies corresponding measurements so that a later analysis can determine whether QoS guarantees were fulfilled. The third identified task, *perform measurements*, schedules and runs the benchmarks, collects the gathered data and stores it for future evaluation, which closes the Deming cycle with the anew definition of a target configuration.

6.1.5. Summary

This Section introduces a concept to facilitate and enforce network QoS management in VEs using a control loop with tasks derived from a known approach to continuous improvement. In this case, the VE's adhering to QoS requirements is continuously improved.

Figure 6.6 shows the individually introduced phases as combined approach. Its intention is a control loop realised through continuously repeating the four identified phases. The coordination and event handling task is the authority to control how frequently and automatically target configurations are generated and implemented according to the introduced control loop. This concept allows for systems with varying degrees of automation, ranging from fully automated adaptation to changes and performance management to systems that require a management actor to explicitly create target

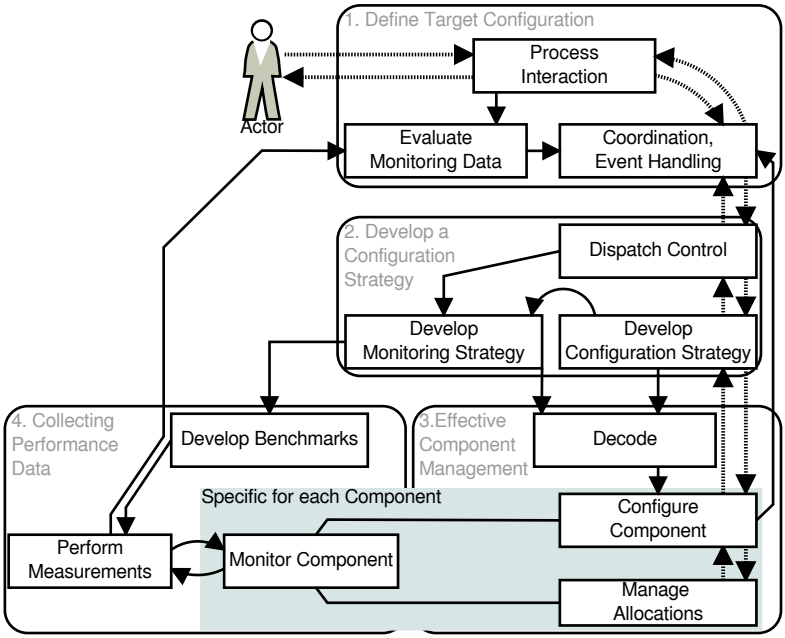


Figure 6.6.: Complete management control loop

configurations and trigger each loop iteration. Complex systems can be devised, that are fully automated in some cases and require actor interaction in other cases.

A specific procedure for assimilating the results of management performed by actors is foreseen along the dotted lines in the illustration. It illustrates processing of management operations from top to bottom, while the arrow chain from the bottom up, towards the actor, represents the possibility for feedback and escalation in case of failure. The procedure features three intermediary steps between the initial processing of an actor's performed management operations and the actual configuration of components. Along

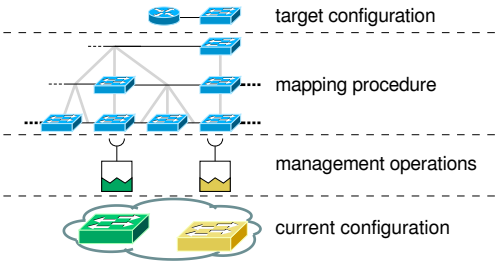


Figure 6.7.: Refining target to current configurations

this path all requirements pertaining to coordination, automation and refinement of the high level VI management must be performed, to end up with management operations on individual components suited to implement network QoS in VEs.

6.2. Automatically configuring virtualization environments

The approach introduced in Section 6.1 employs two models of the same VE. The *target configuration* represents what the VE should be, while the *current configuration* captures the VEs current state.

This section describes the procedure and corresponding information to derive management operations from the target configuration to meet network QoS requirements. To change the current configuration, the management operations must be specific to the components of the VE. Consequently, the step from abstract components to actual instances and implementations must be made as part of the refinement procedure. By developing a scheme for management operations, this step is delegated to the implementations of the management architecture, where products and implementations are known.

Figure 6.7 illustrates the role of the mapping procedure introduced in this section within the architecture. The example target configuration is a path between a router and a switch. The path is part of the target configuration, because it has a QoS attribute and a corresponding requirement. For instance, the QoS attribute is data rate (cf. Section 2.2.1) at OSI layer 3 and the requirement is a lower bound of 200MBit/s. The procedure, introduced in this section, refines the path until

- every node along the path can be mapped to an existing component, such that
- every edge connecting two nodes matches an existing (and manageable) data path,
- with no intermediary nodes/components.

While refining the path, the QoS attribute and its requirement are also adapted, to be meaningful for each identified path segment. The above mentioned scheme for management operations has been specified for each existing component. For performing management, the refined path is as a list of such management operations on each path segment's endpoint. The QoS attribute is a controlling parameter. This leads to a configuration of the components described in the current configuration, according to the intentions and requirements defined as the target configuration.

Developing the refinement procedure is an adaptation of the generic mapping approach identified in [JiNa 04]. The target configuration corresponds to the *user layer* and the refinement procedure's output is the *resource layer*. The classification of components within a VE has been published previously in [Metz 10] and the refinement procedure has been introduced in [Metz 14b].

The refinement procedure is based on a generic model of links and endpoints, used to describe paths. The following Section 6.2.1 derives develops this model, before Section 6.2.2 specifies the refinement procedure. The scheme of management operations that must be realised by implementing components is laid out as part of the functional model in Section 6.3.3.

6.2.1. Components and links

Figure 6.8 shows a classification of network components by locality of configuration, previously published in [Metz 10]. *Physical* components are fixed within the VE. While they may be replaced, for instance due to hardware failures, their place and role within the network topology remains unchanged. *Local* components are placed on hosts. Their roles and function may be copied to other hosts, but their states are bound to their hosts. *Volatile* components are (equivalent to) VMs in terms of placement within the VE. Their state is invariant to placement within the VE, so they can be integrated at different positions in the (virtual) network topology.

This classification indicates how much immediate control a component has over the underlying physical resources. While physical components can map available resources to network performance accurately, local components contest for shared resources with other components and VMs. Volatile components are in general unaware of physical resources and how they are coordinated. Following through, the introduced classification separates how network QoS must be implemented by components:

- Physical components can directly implement network QoS.
- Local components must additionally handle resource contention. Effectively they implement network QoS with respect to the current load of their host.
- Volatile components implement some QoS themselves, but must delegate the task of ensuring that the required resources are available to the virtualization platform.

Similar to their physical role models, virtual network links are passive and network QoS management is performed at their endpoints. For any functional kind of network component, e.g. switch, classification by locality of configuration yields different types, concerning how resources are committed and network QoS management is performed. Linking components of different types, yields a set of discernible link types, summarised in Table 6.1.

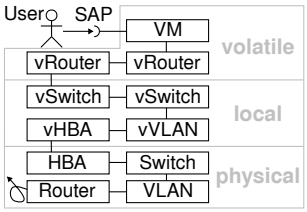


Figure 6.8.: Exemplary path across component classes

src ↓ / dst →	**p**	**l**	**v**
p	(a) p. link	(b) p. uplink	(d) pv-bridge
l	(b)	(c) E2E link	(e) v. link
v	(d)	(e)	(f) RDMA

physical
local
virtual

Table 6.1.: Generic discernible direct link types by endpoint types

The six link types are characterised by their endpoints and therefore how the links are managed. The types *physical link*, *physical uplink*, *E2E link* correspond to links also found in strictly physical infrastructures, while the types *pv-bridge*, *virtual link*, *RDMA* bear virtualization specific challenges. The links are characterized as follows:

(a) **physical link** Virtual components are never endpoints to this link type. For such a link, network QoS, does not require virtualization specific adaptations [FeHu 98].

(b) **physical uplink** Virtual components are connected to the physical network. Virtual components managed by the VMM compares to processes managed by an OS: the VMM is aware of all resource contestants and the system load they generate. The VMM can assign resources like an OS scheduler. For QoS management the virtual component can regarded as an endpoint in strictly physical topologies.

(c) **E2E link** A trivial end-to-end (E2E) link in the context of physical networks, as both endpoints are located on the same host. Physical network resources are not involved in implementing this specific type of link. However, this link type could be used in more complex set ups to serve as tunnel endpoint for encapsulated network traffic.

(d) **pv-bridge** The uplink of a volatile component to the physical topology. The volatile component is generally not aware of resource contestants,

but has been given an uncontested direct uplink through the VMM. Even though the uplink is uncontested, the VMM must ensure the volatile component has enough other resources available.

(e) **virtual link** The usual way of connecting virtual components, mostly a volatile component and a local component. The VMM manages both endpoints and can allocate the resources to meet QoS demands.

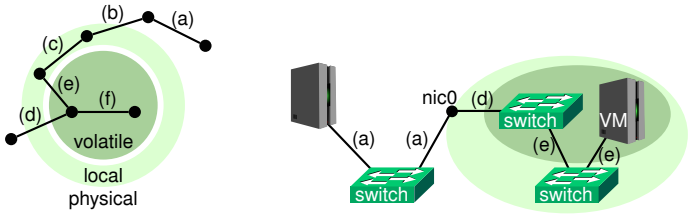
(f) **RDMA** (remote direct memory access) A theoretical direct connection between volatile components, without data ever crossing the VMM's domain. This link type is named RDMA, as this is most likely how it would be implemented, as both endpoints are placed on the same host and traffic between them is never handed to the VMM. This direct link type clearly violates the isolation of virtual containers.

Regarding the virtualization of endpoints, this is a complete list of link types, enabling topological descriptions of network paths through VEs. From a (QoS) management perspective, links also differ in the OSI layers on which requirements must be fulfilled and measurements are performed.

The above list is a generic template and specialized versions for the specific technologies employed in the endpoints must be derived. Technology and OSI layer characteristics for QoS handling yield a VE specific finite set of discernible link types, for which management functions must be available.

6.2.2. Resource layer topology view

A resource layer non-ambiguously displays the components and links introduced in this section [JiNa 04]. The different hosts must be clearly distinguishable, because knowledge about components sharing the same physical resources is essential for resource allocation in QoS management. A topological view which groups network components by host and distinguishes between local and volatile components seem reasonable. It is consistent with describing links by their endpoints and components by their degree of abstraction from the physical hardware. The resulting view contains all discussed information, thus enabling refined QoS management for all identified link and component types.



(a) Links from Table 6.1

(b) Enhanced topological view with virtualization host introspection

Figure 6.9.: Resource layer views

Figure 6.9(a) shows an abstract example featuring each link named in Table 6.1. By making the endpoints' domains (physical, local, volatile) explicit, the various link types are discernible without the labels. Figure 6.9(b) shows an application example, connecting two servers over three switches. Each switch is one of the types introduced in Section 1.5. The labelling is kept for clarity purposes only. The physical host and switch are displayed as single components, while the virtual components are displayed inside domains (shaded areas) within the same host.

Shaded areas group local and volatile components by host. All virtual components realised by the same physical host belong to the same area. With this illustration, volatile components are always contained within the local area. They are optically set apart, by using different shades. The link between the two virtualized switches crosses domains, which means it must be a type-e link. The intersection of a link with the host boundary in Figure 6.9(b) is marked and explicitly names the used host's NIC, *nic0*. Which NIC is used determines which resources are used for the specific link and must be represented on the resource layer.

This enhanced topology view is suited to include all identified component and link types. The increased information meets the increased complexity and qualifies as resource layer for QoS layer partitioning [JiNa 04] as described in Section 1.5.

6.2.3. Refinement procedure

With the resource layer established in Section 6.2.1, a generic procedure for mapping application layer management information or tasks to the resource layer can be described. In this case, QoS attributes and requirements. This is achieved by iteratively refining application layer links to eventually match the resource layer topology. QoS attribute refinement is then performed based on reasoning on intermediary results. With each step the QoS attribute and its requirements are adapted to suit the subpath which ultimately results in QoS requirements for each link at resource layer, which can then be implemented in a technology specific manner. Monitoring strategies are developed by analysing the topology at resource layer and accumulating QoS requirements for common subpaths. Subject to monitoring are components and subpaths with accumulated QoS, which allow to calculate the achieved QoS for the topology defined as target configuration.

Network components are often characterised by the highest layer of the ISO OSI reference model they implement, e.g. routers implement layer 3. Combined with our taxonomy in Section 6.2.1 every component within a virtualization infrastructure can be characterized by the highest OSI layer it implements and its locality of configuration. The refinement procedure must therefore trigger two tasks: adaptation of QoS attributes to the identified link segments and advancing the refinement process by detecting and resolving compound links. The procedure develops the resource layer topology (final topology) from the application layer topology (initial topology), based on available management information. Algorithm 1 shows the core procedure `refinePath`, which uses two noteworthy subroutines: `link` (lines 8 and 22) and `adaptQoS` (lines 8, 12 and 22).

The `refinePath` procedure uses the property of the OSI model, that every layer n uses the links and services provided by layer n-1, to break paths into subpaths, starting at the layer of an input-endpoint, proceeding to lower layers. If a subpath cannot be split up, it corresponds to a single resource layer link, for which adequate management functions are available. When a path with a directly corresponding link in the resource layer is found, the

Algorithm 1: The `refinePath` procedure

input : T_{res}(): all links in the final topology, P_{input}(): a path's endpoints and QoS attribute
output: $P_{refined}$(): list of links in the final topology associated with a QoS attribute

```
1  P_refined ← ϵ
2  e0, e1 ← P_input.endpoints
3  P_res−layer ← PathAtOsiLayer(e0,e1,e0.layer)
4  while P_res−layer ∉ T_res do
5      n ← NeighbourOf(e0,P_res−layer)
6      if n ≠ e1 then
7          if path(e0, n) ∈ T_res then
8              P_refined.append( link(e0,n,adaptQoS(P_input,n)))
9          end
10         else
11             P_sub ← path(e0,n)
12             P_sub.qos ← adaptQoS(P_input,n)
13             P_refined.append( elementsOf(recursion into subpath))
14         end
15         e0 ← n
16     end
17     else if n = e1 then
18         e0.layer ← e0.layer- 1
19     end
20     P_res−layer ← PathAtOsiLayer(e0,e1,e0.layer)
21 end
22 P_refined.append( link(e0,e1,adaptQoS(P_res−layer,e0.layer)))
23 return P_refined
```

function `link` is called. Calling `link` is a request to apply the input-path's QoS attribute to a resource layer link. To actually apply the attribute, `link` uses topology information to identify the technologies used for the endpoints and the link type. With this information `link` can determine the correct management for the path segment.

The task of `adaptQoS` is provide an adapted QoS attribute to `link`, by tracking the performed path refinement. The concrete adaptation procedure is specific to the QoS attribute. For instance, for a sensible implementation for an attribute "data rate", `adaptQoS` would alter the attribute values to account for protocol header fields, when regarding data rate at varying OSI layers, or special implementations of virtualized links.

Besides accounting for implementation characteristics, `adaptQoS` tracks paths and subpaths, to consolidate requirements of multiple high-level links sharing the same resource layer link. This happens when `link` is called multiple times for the same resource layer link. Also, the information collected by `adaptQoS` is used to set up monitoring.

The result of the procedure are the application layer links as resource layer paths, where each path segment is a resource layer link with a refined QoS attribute associated. The actual application of the attribute is resource allocation and deploying monitoring.

The implementation of links, resource allocation and QoS is technology specific. For example, in previous work [Metz 11], we analysed that VMMs often use different metrics to model similar aspects of virtual and physical components. As QoS management heavily depends on the monitoring data obtained from the managed systems, QoS management must become ever more specific for the managed object. This can be seen as one of the main causes for the existing gap in QoS management, when introducing virtual components to IT infrastructures. The advantage of using the model and procedure introduced herein, lies in the generic structure and methodology for describing links and components. The model's application results in a finite and sensible set of managed object types and corresponding management functions or strategies that allow for automated adoption, propagation and application of management information and functions from the high level view of VIs to the concrete provisioning components.

6.3. Submodel conception

In management architectures according to the OSI reference model, the aspects information, function, communication and organisation are specified as individual submodels, cf. Section 2.3. Submodels specify goals, notations, structures, procedures and relations, which are substantiated to implement management systems. Their intentions, according to [HAN 99] are:

`6.3.1 Information model`: define the modelling approach and an unambiguous notation for describing the management information.

`6.3.2 Organisation model`: define actors, their roles and the fundamental principles of their cooperation.

`6.3.3 Functional model`: provide the basis for libraries of partial management solutions and for the delegation of management functionality to agents.

`6.3.4 Communication model`: specify communication partners, communication mechanisms, exchange formats and embedding into the underlying communications architecture.

The presented order is also the order in which the models are developed. The information and organisation models are structured into *domains*, while the functional model consists of *functional components*. The communication model distinguishes *interaction types*.

Section 5.2 lists numerous management tasks along the life cycles of virtual components, networks and infrastructures which are relevant for the management of VEs. Many of them are beyond the scope of network QoS management and therefore beyond the scope of this architecture.

Especially with sustainability in mind, the architecture must be either extensible to enable all identified tasks, or integrable with other architectures, with different concerns. To that end, the information meta model foresees an association of management information with life cycle tasks *LifeCycleOperations* and the functional model features an unnamed interface, which is an abstract representation for interfaces concerning management of VEs, but not network QoS.

6.3.1. Information model

An information model captures relevant management information for the problem at hand, in this case: network QoS management in VEs. It imposes a structure on management information which leads to *managed objects*

(MO), representations of the (relevant) aspects of the subjects to management. To reach its goal, the information model must specify a modelling notation and structure management information pertaining to conceptually independent domains [MNM Schi 07, MNM Marc 11]. With a complete information model, all meaningful management functions affect MOs.

Regarding the architectural concept in Section 6.1, independent information model domains (IM domains) can be identified along the defined phases. As QoS plays an important role throughout the entire management loop, the QoS domain is specified as independent IM domain. From the management loop, three additional domains can be identified. The four IM domains and their purposes are:

`6.3.1.1 QoS domain`: describe network QoS.

`6.3.1.2 Presentation domain`: present isolated VIs to managers.

`6.3.1.3 Translation domain`: derive configuration strategies from target configurations.

`6.3.1.4 Realisation domain`: describing the physical infrastructure and the realisation of VIs.

Figure 6.10 illustrates the four identified IM domains as UML packages. The shaded classes indicate completely new developments, while clear classes mark the adaptation of existing approaches. The depicted package classes are reduced to classes with associations between domains.

To perform management on the VE, the relevant information from the *presentation* domain is accumulated into the *TargetConfiguration* while the relevant information from the *realisation* domain is accumulated into the *CurrentConfiguration*. The association between *TargetConfiguration* and *Path* in Figure 6.10 is dashed, to indicate it hides the internal structure of the *translation* domain, where a direct association between these two classes does not exist.

The *presentation* domain is based on the VN-SLA schema, first introduced in [FAP 10]. The Common Information Model (CIM), maintained by the

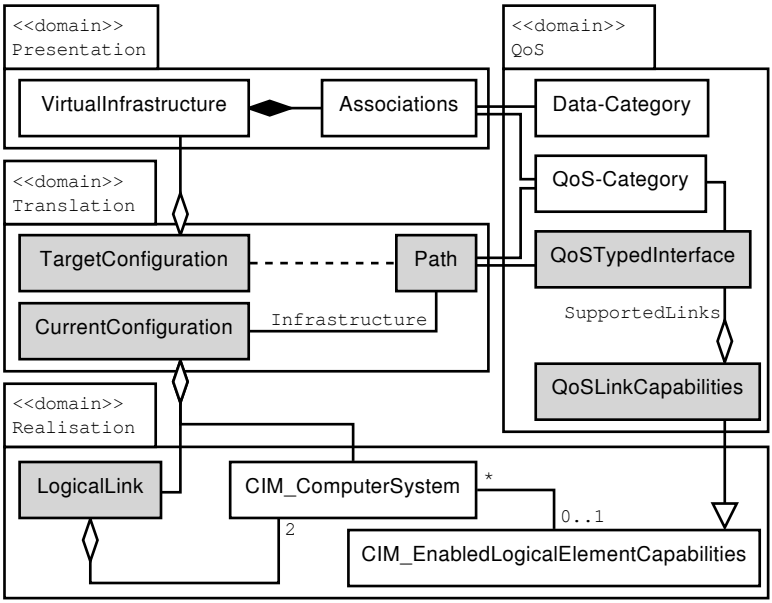


Figure 6.10.: Management domains within the information model

Distributed Management Task Force (DMTF)[1], covers most of the *realisation* domain. Modelling network QoS and its requirements is based on the approach introduced in [MNM Roel 05]. Since the VN-SLA, CIM and [MNM Roel 05] all use UML for modelling information, UML is chosen as the specification language for this model as well.

To serve the purpose of this architecture, all MOs follow the meta model illustrated in Figure 6.11. Every attribute of a MO is important to network QoS management in three aspects:

`specification`: The managers intention for this attribute. For instance,

[1]http://www.dmtf.org

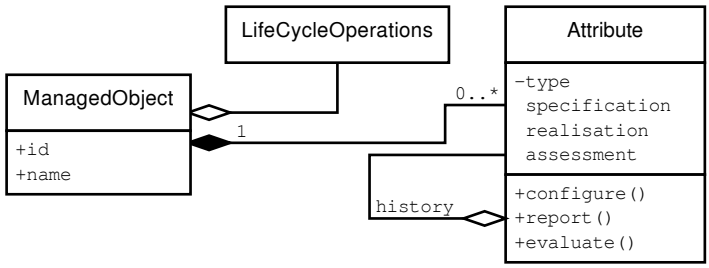


Figure 6.11.: Meta model for managed objects of the presentation domain

the attribute "placement" of a *Node* could describe a wish to pin a VM onto a specific host.

`realisation`: The measured/determined current state pertaining to this attribute. For instance, the host currently running a VM.

`assessment`: The result of an evaluation determining whether the realisation is acceptable as implementation of the specification. For instance, "fulfilled" if the *Node* is currently placed on the specified host, "not fulfilled" otherwise.

For advanced reporting, each attribute may offer access to its history of previous versions, that are earlier specifications or preceding measurements. The corresponding generic management operations, to access or modify the management information stored as MO attributes are:

`configure()`: Set the attribute *specification*; inform the system about a change in intention.

`report()`: Set the attribute *realisation*; capture the current situation, e.g. measuring results.

`evaluate()`: Set the attribute *assessment*; gauge the difference between *specification* and *realisation*.

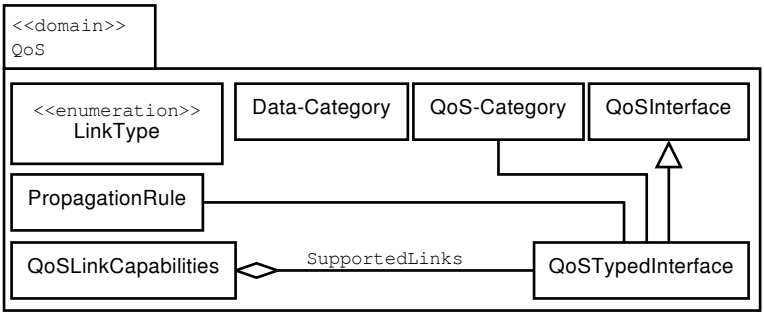


Figure 6.12.: Classes of the *QoS* domain

With this meta model, managers read *realisation* and *assessment* values and set the *specification* value. The indicated *LifeCycleOperations* are management operations that advance MOs along their life cycles, identified in Section 5.2.

When introducing the four IM domains in the following, full descriptions are only included for the classes depicted in Figure 6.10. Complete specifications for all classes can be found in Appendix C.

6.3.1.1. QoS domain

With elaborate QoS models available, their application to VEs and especially this architecture are the main concern of the *QoS* domain. Figure 6.12 shows the *QoS* domain and the therein specified classes. This domain is based on the QoS model introduced in [MNM Roel 05]. *Data Category* and *QoS Category* are proxies, representing the entire QoS model introduced in [MNM Roel 05], where data traffic is assigned QoS properties, by associating *Data Category*s with *QoS Category*s. The other classes model additional information required to apply QoS to VEs. This information is associated to MOs in three ways:

- *Association* of the *presentation* domain formulates network QoS and requirements by associating a *QoS Category* with a *Data Category*, analogous to *Session* in [MNM Roel 05].
- *Path* of the *translation* domain relates to *QoS Category* to control the refinement procedure introduced in Section 6.2.3.
- *LogicalLink* and *CIM_ComputerSystem* of the *realisation* domain use, i.e. implement, the interfaces and capabilities to effectively realise network QoS in VEs.

The type *LinkType* is used to classify a link or path(-segment) or interface according to the taxonomy for links introduced in Section 6.2.1. A complete specification for all classes can be found in Appendix C.1. The following describes the classes of the *QoS* domain, with associations to classes of other domains. These classes are:

- DataCategory
- QoSCategory
- QoSTypedInterface
- QoSLinkCapabilities

`Class DataCategory`

---

*Data Category*s describe communication patterns in terms of direction, senders and receivers. This information is attributed to *Association*s from the *presentation* domain to specify how *QoS Category*s must be applied. The classic *Topology* (see page 229) is n:m bidirectional communication, where every node may interact with every other node, while the classic *Path* (see page 232) is a 1:1 bidirectional communication, where two specific endpoints communicating are singled out.

`Class QoSCategory`

The class *QoS Category* is a set of *QoS Parameter*s specifying the properties that data traffic of this category has, or should have. *QoS Parameters* [MNM Roel 05] include semantic descriptions of the individual properties, criteria determining its fulfilment and an assurance type, describing how vigorously the property is enforced.

`Class QoSTypedInterface (derived from QoSInterface)`

QoSTypedInterface combines management information and methods, so that an implementing component can be managed, with the proposed approach, to provide and guarantee network QoS in VEs. For a specific *QoS Category* the *QoSInterface*(see page 222) is refined and extended by a *PropagationRule*(see page 224). With this combination, the refinement procedure can map and adapt the *QoS Category*, for a specific *LinkType* (see page 221), while maintaining its semantics and corresponding requirements.

`Class QoSLinkCapabilities (derived from CIM_EnabledLogicalElement-Capabilities)`

This derivation from *CIM_EnabledLogicalElementCapabilities* [DSP 1041] models a *CIM_ComputerSystem*'s (see page 236) capabilities to provide network QoS in VEs. To that end it must always attribute an OSI layer to the component, as the layer where the component fulfils it's main purpose. The concrete capabilities are an aggregation of *QoSTypedInterface*s, which can be read as a list of *LinkType*s for which the component can realise network QoS in VEs.

6.3.1.2. Presentation domain

This is the most abstract representation of VEs. It contains VIs as initially described and expected by customers and is used to manage reports on the topologies and QoS achievement. The MOs within the *presentation* domain are exposed to human managers and fulfil the task of structuring and storing specifications and reports.

All tasks performed by the mangers create or modify specifications that govern how the software system implementing the feedback loop performs management. Conversely, monitoring data acquired from components is processed, to report meaningful data pertaining to individual VIs.

Figure 6.13 shows the *presentation* domain's MO classes and their relations. This domain is based on the *virtual network specification* submodel in [FAP 10]. Table 6.2 maps the corresponding classes. The introduced

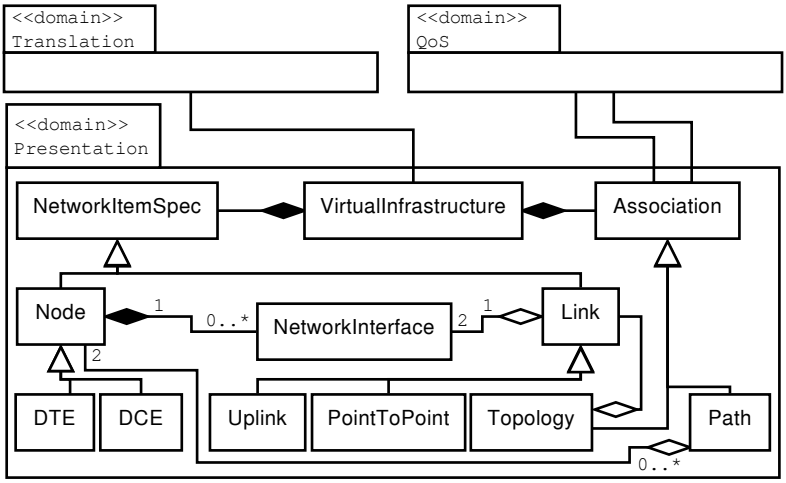


Figure 6.13.: Classes of the *presentation* domain

Classname	**Role model**
VirtualInfrastructure	Virtual Network Specification
NetworkItemSpec	Network Components
Association	Associations
Node	Node
Link	Link
NetworkInterface	Network Interface
Topology	topology
Path	Path

Table 6.2.: This model's classes mapped to their role models from [FAP 10]

classes model the same VE elements as their role models, maintaining the claim for suitability to describe VIs, i.e. VNs. The evolution performed pertains to relations and the introduction of the meta model for MOs, in order to use the model for steering and reporting. Classes derived from *NetworkItemSpec* model information pertaining to the haptic aspects of VIs, i.e. components and links. The derivations from *Association* attribute QoS aspects to selections of certain *NetworkItemSpec*s.

The refinements of *Node* to *DTE*, *DCE* and of *Link* to *Uplink*, *PointToPoint* fan out the building blocks for topologies to discern all types of components exposed to managers. While users directly interact with *DTE* components, that realise unspecified (sub-) services, *DCE* components have a clear task in a topology and are used transparently. *Uplink* and *PointToPoint* distinguish between the subtypes of *Node* they link. While an *Uplink* links a *DTE* and a *DCE*, *PointToPoint* links two components of the same type. This information is mainly used while developing configuration strategies, but is also very sensible to discern when interacting with human managers.

Typically every instance of *VirtualInfrastructure* contains exactly one instance of *Topology*, constituting the VI's network. Especially in larger VIs, it is conceivable that VIs manage individual *Topology*s for disjoint connected components of the network. A *Path* ties network QoS properties to data traffic between two *Node* endpoints. The following describes each MO in detail.

A complete specification for all classes can be found in Appendix C.2. The following describes the classes of the *presentation* domain, with associations to classes of other domains. These classes are:

- VirtualInfrastructure
- Association

`Class VirtualInfrastructure`

The representative of an infrastructure that has been created within a VE. Its main purpose is to group MOs into one entity that represents the fulfilment of the requested infrastructure. As such, it is a composition of

NetworkItemSpec (see page 227) and *Association* objects. *VirtualInfrastructure* objects can be associated with *Manager*s (cf. Section 6.3.2) as part of access control.

Specifically for the task of QoS management, *VirtualInfrastructure* has a `resourcepool` attribute. It is a list of the VE's allocatable resources, with an upper bound for the accumulated allocation of each resource to the VI's elements. The upper bounds specified in the resource pool do not need to trigger any allocations. Their minimal use is merely providing reference values for monitoring purposes.

The concrete modelling of allocatable resources is not important for this architecture. Its presence and consistent implementation is required to compare available resources vs. allocated and employed resources. The quality of allocation modelling therefore greatly influences efficiency of allocation and planning, but not basic applicability. A lightweight example for such a model are the dynamic *SLS* constraints in [FAP 10], while the CIM Resource Allocation Profile [DSP 1041] provides a very sophisticated approach.

The management information stored in this MO can be modified by adding and removing elements form the lists. For the resource pool, the upper boundaries for each resource can be changed individually.

`Class Association`

The base class for associating network QoS to elements of *VirtualInfrastructure*. This class merges the selection idea from [FAP 10] with the model introduced in [MNM Roel 05]. Derivations of this class are associated with specific *Data Category* classes. Therefore they map to Sessions in [MNM Roel 05]. Derivations from *Associations* also specify how *QoS Category*s are to be applied to *NetworkItemSpec*s.

6.3.1.3. Translation domain

As indicated by Figure 6.10 on page 111, the *translation* domain is the biggest contribution to filling the gap in managing VIs, network QoS mod-

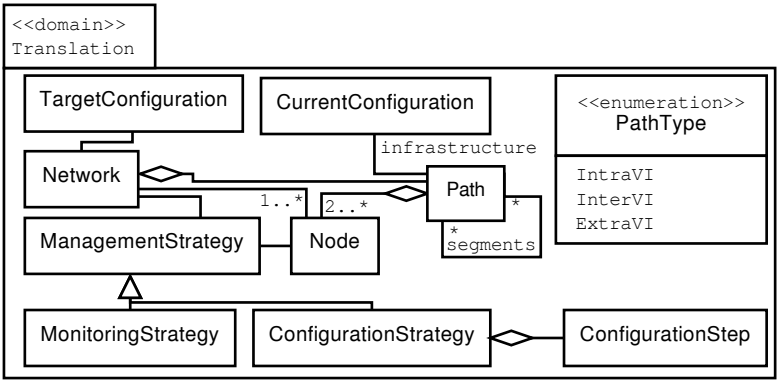


Figure 6.14.: Classes of the *translation* domain

els and infrastructure models. MOs of this domain serve the mapping procedure (cf. Section 6.2) and assist automated management.

The *TargetConfiguration* and *CurrentConfiguration* are views on the *presentation* domain and *realisation* domain, respectively. The former is restricted to currently active *VirtualInfrastructure*s and its associates. The latter is a snapshot of the current topology and realisation of virtual components.

Figure 6.14 shows *Network*, *Node* and *Path* as the only classes intended to model virtual components and topologies. These three classes forge a relation between *TargetConfiguration* and *CurrentConfiguration*, which allows the creation of *ManagementStrategy*s. Objects of class *Network*, *Node* and *Path* exist solely for this purpose and are valid only for the specific relation they form.

For the task of performing management, *ManagementStrategy* and its derivation are the classes that define the management operations performed on components. They control changes to the VE and how feedback is provided to the managers. A complete specification for all classes can be found

in Appendix C.3. The following describes the classes of the *translation* domain, with associations to classes of other domains. These classes are:

- TargetConfiguration
- CurrentConfiguration
- Path

`Class TargetConfiguration`

A *TargetConfiguration* is a collection of all currently active *VirtualInfrastructure*s domain. Active VIs have their components placed within the VE. This means there are *CIM_ComputerSystem* (-derivation) instances, that map 1:1 to *Node*s and the *Link*s and *Association*s of the *presentation* domain can be mapped onto sequences of *CIM_ComputerSystem*s (-derivations) and *LogicalLink*s of the *realisation* domain.

`Class CurrentConfiguration`

A *CurrentConfiguration* is a collection of all *LogicalLink*s and their endpoints where both endpoints are currently active.

`Class Path`

The *Path* of the *translation* domain is the model's main information hub for mapping network QoS from VIs on the VE. When a *Path* is created its endpoints are analysed to determine the *Path*'s `type`. The corresponding enumeration *PathType* is:

`IntraVI`: Both endpoints are part of the same *VirtualInfrastructure* and corresponding *Node*s can be obtained canonically.

`InterVI`: Both endpoints are in this VE, but not part of the same *VirtualInfrastructure*. There must be a "gateway" between both VIs and QoS must be configured for both VIs. This may include extending the *TargetConfiguration*.

`ExtraVI`: One endpoint is outside this VE. A separate mechanism must be employed to determine how and where data traffic enters and leaves the VE and VI. A *Node* must be created that marks the end of the path segment which can be managed as part of this VE. Management of the path beyond that *Node* must be delegated, i.e. cannot be guaranteed.

Only when `type` is *IntraVI*, refinement as described in Section 6.2 can be performed. All other types require the creation of subpaths stored in the *segments* list, that are either *IntraVI* or can be delegated completely.

6.3.1.4. Realisation domain

In Section 6.2 a refinement procedure is elaborated, revolving around the idea of the properties of the "resource layer" introduced in [JiNa 04]. The idea is to refine management requests so that they can be trivially mapped onto management operations for real, configurable resources. The result of the refinement procedure is part of the *translation*. By basing the *realisation* domain on a sophisticated model for IT infrastructures, suitable to accurately capture the components states and management functions, little additional information is needed to derive a complete *realisation* domain. The CIM is such a model.

Figure 6.15 shows the classes of the *realisation* domain. All classes carrying the prefix `CIM` are taken from the CIM profiles. The class *LogicalLink* is added for adequate means to model topologies within the *realisation* domain. The type *ConfigurationLocality* discerns types of *CIM_ComputerSystem*s, according to the taxonomy introduced in Section 6.2.1.

While the classes and the information they represent are as specified in the CIM, separating between physical, local and volatile components exceeds the CIM's capability to discern between virtual and logical. Figure 6.15 illustrates this by showing *CIM_VirtualEthernetSwitch*. This class is not immediately adoptable from the CIM, as it is necessary to separate virtual from volatile components, especially switches.

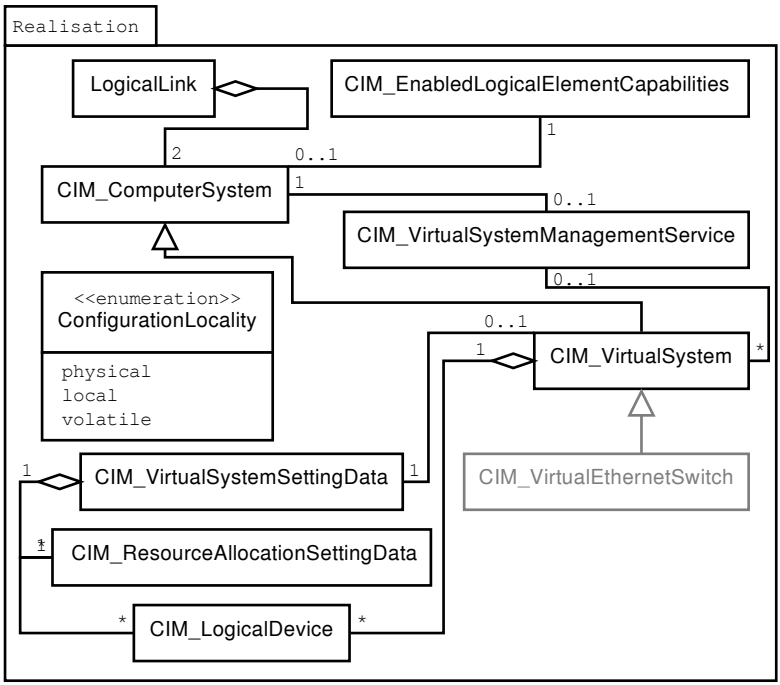


Figure 6.15.: Classes of the *realisation* domain

Implementations of this model must derive classes from *CIM_ComputerSystem* and *CIM_VirtualSystem* that define the topmost layer of the ISO OSI model the component implements, e.g. 2 for a switch, 3 for a router. This definition is a stored as a capability of the derived class. As this information is used for QoS mappings, the corresponding attribute is part of the *QoSLinkCapabilities* class in the *QoS* domain.

The following describes the new class *LogicalLink* and the extensions to *CIM_ComputerSystem* required for this model.

Role	Short description
User	Manage the services provided on VIs.
Local admin	Integrate services provided on VIs into local network.
Hardware provider	Operate and maintain physical components.
Virtualization provider	Operate and maintain virtualization platform. Provide VIs.
Primary customer	Manage VIs and their components to enable services.
Secondary customer	Manage VI components to enable services.

Table 6.3.: Summary of actors and their roles identified in Section 5.1

`Class LogicalLink`

This is a direct connection between two derivations of *CIM_ComputerSystem*, such that no intermediary managed component handles the data traffic between the endpoints. Classifying this capability using the ISO OSI model, this is layer 2 functionality, hence a logical link. Representing an existing communication path, this class stores its *LinkType*. The actual implementation of QoS for the link is realised by its endpoints.

`Class CIM-ComputerSystem`

This class from the CIM has been extended by three methods to provide information required by the procedure developed in Section 6.2.

6.3.2. Organisation model

The task of an organisation model is to describe actors, their roles and the fundamental principles of their cooperation [HAN 99]. Table 6.3 summarises discernible roles when performing management in VEs, identified in Section 5.1.

Each role focuses on a relatively small aspect of the VE. For instance, while the *hardware provider* is concerned with physical components only, the *secondary customer* is concerned with only virtual components. A *secondary customer* depends on the resources managed by the *hardware provider*, but

a *hardware provider* does not depend on the *secondary customer*, to fulfil their tasks. Such relations warrant cooperation to ensure VIs are implemented as necessary, to operate the services as intended by their users.

This organisation model follows the approach introduced in [MNM Schi 07]: grouping roles into organisation model domains (Section 6.3.2.1) and describing their cooperation as interaction channels between the domains. (Section 6.3.2.2). Specifying the interaction channels, allows the delegation of role specific tasks, which implies interfaces of functional complexes, which guide the development of an functional model and communication model as well.

6.3.2.1. Organisation model domains

The identified roles are grouped into organisation model domains (OM domains), by common intentions, goals and concerns. Roles belonging to the same OM domain perform management with a similar view on the VE. Figure 6.16 illustrates the four identified OM domains introduced in the following. The domains and their main management concerns are:

`service domain`: service components realised through a VE.
`virtual infrastructure domain`: VIs as service realisation platform.
`resource allocation domain`: resources for realising VI specifications.
`hardware provisioning domain`: providing physical resources.

SERVICE DOMAIN Performing management as a role of the *service* domain is targeted at service components that are realised through a VE. These roles usually have no means to manage VI components directly. Their management results in requirements, which specify goals for managers fulfilling roles of the *virtual infrastructure* domain. While the `user` is concerned with the service itself, the `local admin` focuses on the network access to the service components. The concrete doings of this domain is outside the scope of this management architecture, but it provides motivation, legitimation and the source for requirements for the *virtual infrastructure* domain.

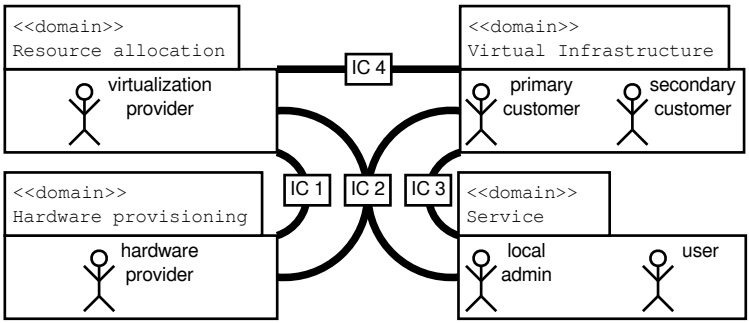


Figure 6.16.: Organisation model domains

Virtual infrastructure domain The roles of the *virtual infrastructure* domain perform management on components and networks independent from their implementations. This domain's view on VIs is scoped to the relevant aspects for users/customers of a VE, as described in Section 2.1.2. Within this domain, the *resource allocation* domain is the enabling partner, providing implementations for the abstracted management. On the other hand, the *service* domain provides goals and relies on the VIs that are managed by the roles of this domain.

A `secondary customer` is restricted to a subset of the VI managed by its associated `primary customer`. `Primary customers` can modify this restriction as well as create and delete `secondary customers`. With regard to the information model, both roles perform management using MOs from the *presentation* domain, only.

Resource allocation domain Within the *resource allocation* domain, management is performed to enable the *virtual infrastructure* domain and allow their specifications be realised. To that end, all resources provided by the *hardware provisioning* domain must be marshalled in a manner that allows combination and allocation to VIs and their components. Continuous effort must be made to ensure the resources allocated are still sufficient for

the VIs requirements. The *virtual infrastructure* provides specifications for components and networks and relies on their availability. The *hardware provisioning* domain is the enabling domain, providing resources.

The actor identified in the scenarios, which lead to the specification of the `virtualization provider` role, also performs management in the role of the `primary customer` of the *virtual infrastructure* domain. It does so for every VI. This is very sensible in the presented scenarios, but management does not need to be that centralised. VIs, as perceived and managed by the *virtual infrastructure* domain, can be provided in parts through multiple `virtualization providers`, belonging to different organisations. Therefore the `virtualization provider` role must not imply the `primary customer` role. This concept of such distributed VIs has also been previously published in [Metz 10].

HARDWARE PROVISIONING DOMAIN This domain comprises roles that are concerned with physical components and topologies, only. As pointed out in Section 5.1, the larger an IT infrastructure, the more likely that management of physical components is divided between roles, for instance with separate roles for network components and servers. Consequently, in the generic case multiple `hardware providers` are involved in providing a VE and vice versa, effects on VIs is beyond any `hardware providers` cannot appreciate the effects of their management on individual VIs. Roles of this domain are scoped to MOs of the information model's *realisation* domain and, even further, to MOs of physical components. As such, this domain is in charge of the basic components.

6.3.2.2. Interaction channels

Considering their motivations, the relations between OM domains are very similar to the IM domains: There is a domain concerned with the management of physical components and topologies, another domain is concerned with the management of virtual components and a third domain is concerned with management enabling virtual components and topologies

by levering physical components and topologies. The organisation model specifies four interaction channels between the OM domains, illustrated in Figure 6.16. The interaction channels and their intentions are:

`IC 1`: obtaining resources.
`IC 2`: coordinating management.
`IC 3`: delivering virtual infrastructures.
`IC 4`: realising components and topologies.

`IC 1`, OBTAINING RESOURCES The purpose of interaction channel `IC 1` is providing and using physical resources. The `virtualization providers` obtain information on resources in the form of physical components, foremost hosts, and networks. Through this channel, additional physical components can be requested as well as changes to physical networks. The `hardware providers` use this channel to facilitate access from the *resource allocation* domain to physical resources.

`IC 2`, COORDINATING MANAGEMENT In terms of production and products, the *hardware provisioning*, *resource allocation* and *virtual infrastructure* domains are production stages, where physical resources are processed, refined and cultivated, to produce a product, used to realise IT services. In that, the *hardware provisioning* domain is the first production stage and its work affects all other domains. The *resource allocation* domain is the second stage, and its work affects the *virtual infrastructure* domain and *service* domain. The *virtual infrastructure* domain is the third stage and its work affects the *service* domain. This implies an ordering of domains along an imaginary production line, before the resources of physical components are useful for some users outside the VE:

1. *hardware provisioning*
2. *resource allocation*
3. *virtual infrastructure*
4. *service*

The intention of `IC 2` is to coordinate management with all managers of objects that may be affected. For instance, if a new server is deployed and connected through a previously unused switch port, there is no immediate need to communicate this management task. The removal of a server from the data centre, however, affects every VI with at least one component placed on that host. Consequently, every `user`, `primary/secondary customer` and `virtualization provider` in charge of anything concerning any of the VIs, may experience effects of the server removal. The interaction channel `IC 2` may be used to communicate the server removal, so that other managers have the opportunity to react accordingly.

This channel is a generic coordination platform. Coordination can be unidirectional, simply announcing management operations about to be performed. Or actual coordination may be performed, where a management operation is proposed and may only be performed after it has been signed off by all affected managers. The latter causes a significant delay between the initial intention to perform management and the actual performance.

The manager announcing management using `IC 2` may be a different manager than the one performing management. For instance, `hardware providers` may only tell their immediate customers, *virtualization provider*s, about their intentions. In turn, the *virtualization provider*s notify their immediate customers of the *virtual infrastructure* domain. Notifications are always directed towards the higher numbers in the above list.

`IC 3`, DELIVERING VIRTUAL INFRASTRUCTURES If the entirety of infrastructure management, including QoS, components and networks, physical as virtual, was regarded as a single black box service that "just works", then `IC 3` would be the service access point.

Through this channel, specifications and requirements for servers, networks and QoS are communicated in a technology agnostic manner, to be implemented by customer managers. The customer managers, of the *virtual infrastructure* domain, are in charge of realising the specifications and requirements, so that the resulting VI is suited to serve the needs of user, who intend to operate services on the requested infrastructure.

Using IC 3, managers of the *service* domain gain access to the components realising service components and reports concerning the achieved QoS are requested and provided through this channel. No immediate management, directly affecting the behaviour of the VE is preformed through this interaction channel.

IC 4, REALISING COMPONENTS AND TOPOLOGIES Channel IC 4 specifies the interaction between the *virtual infrastructure* domain managers, and *resource allocation* domain managers, to provide the components, topologies and resources, realising a VI. The customer managers of the *virtual infrastructure* domain communicate the infrastructure they want to realise to the virtualization providers. In turn, virtualization providers provide information on the realisation, if it has been accomplished, and monitoring data, measuring the implementation.

Through this channel the customer managers gain access to the VEs maintained by virtualization providers and hardware providers. Usually information on how components and topologies are realised is not communicated to customer managers.

6.3.2.3. Integration with the information model

To provide a formal specification, this section introduces an UML class diagram of the organisation model. The diagram also serves as a means to associate interactions with MOs. As a result, information required for interaction is specified. Additionally, the reasons and subjects for interacting roles is formalised.

As a basis for the class diagram, the meta model from [MNM Schi 07], illustrated in Figure 6.17, is reused. Each interaction channel is a class and its instances associate management roles with each other, for the purpose of performing management.

The class diagram, Figure 6.18, associates interaction channels with classes from the information model. These associations must be interpreted as the

Figure 6.17.: Meta model for classes of the organisation model, from [MNM Schi 07]

reasons for interaction and are therefore defining for the concrete interaction. Analogous to Figure 6.16, interaction channels are depicted between organisation domains, meaning any role within this domain.

`ManagementRole`s aggregated by `OrganisationDomain`s, means a role belongs to a domain and the role represents a set of responsibilities attributed to this domain. There are three *IC 2* derivations, differing in the types of associated *OrganisationDomain*s and classes from the information model.

For clarity purposes in Figure 6.18, the associated classes from the information model to the *IC 2* derivations were replaced with a dotted association between *IC 2* and other interaction channels, which refer to the identical set of information model classes. All shaded classes in the Figure are organisation model classes, those without background colouring are classes from the information model.

6.3.3. Functional model

The idea of a functional model, according to [HAN 99], is to provide building blocks of management functions, that may be combined to (partial) management solutions, fitting the managed distributed system's needs.

With the management loop introduced in Section 6.1, a segmentation and ordering of management tasks has already been performed. The functional model provides a sensible grouping of the tasks to functional components and their interfaces.

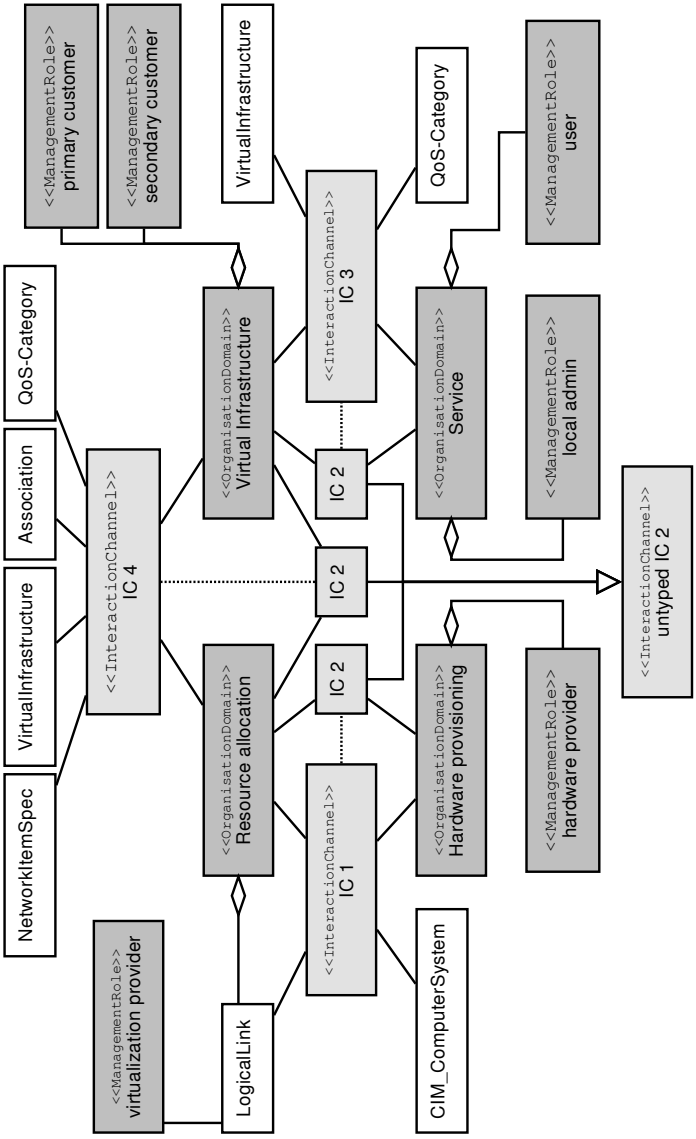


Figure 6.18.: Class diagram of the organisation model

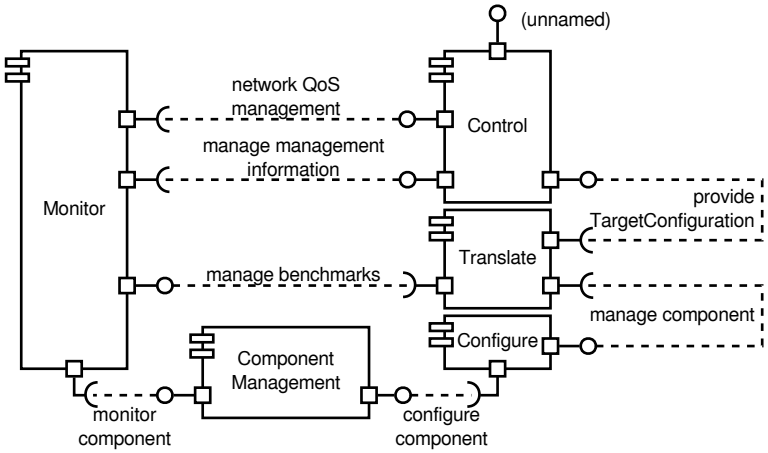


Figure 6.19.: Component model illustrating the relations between the five main functional components

To advance the management loop, it is necessary to complete the current phase. This makes it intuitive, to specify functional blocks, that may implement individual phases. Yet, sensible segmentation is also performed along the involved MOs, to end up with functional components that handle a subset of the MOs described in the information model and require a reduced amount of information to work.

Figure 6.19 shows five main functional components, their interfaces and couplings. The Component Management at the bottom of Figure 6.19 is the functional basis, on which this architecture builds to realise network QoS in VEs. At the top of Figure 6.19, the unnamed interface for Control is the starting point for extensibility and integration with other architectures, alluded to on page 108. This interface is also the main service access point for management systems following this architecture, introduced as `IC 3` in the organisation model (cf. page 128).

In general the five components depicted in Figure 6.19 are intended to

each implement the tasks of a specific phase from the management loop introduced in Section 6.1. The contribution of the functional model is the specification of management functions used to fulfil these phases' tasks. The functions are grouped to interfaces, exposed by subcomponents and depicted as component facets. There are two deviations when attributing phase tasks to functional components:

- Component Management is an adaptor for a component's management interface through which the component may be managed and monitored. This functional component is provided by or for each managed component, used by implementations following this architecture. It must always be provided by the component, i.e. the vendors. Its presence and functionality for configuration and monitoring is a prerequisite as per `Req.#11`, `Req.#12`, `Req.#13` and `Req.#15`. The adaptation into the management architecture are the *decode* task of phase 3 (Section 6.1.3) and the *perform measurements* task of phase 4 (Section 6.1.4).
- Monitor is the functional component corresponding to collecting performance data phase of the management loop (Section 6.1.4). It also realises the *evaluate monitoring data* task, originally attributed to the phase defining the target configuration (Section 6.1.1). This task was moved to functionally group all monitoring aspects, especially the required management information, together.

The following sections describe each main functional component in detail. First, each component is broken down into functional subcomponents, then the exposed interfaces with their management functions are described.

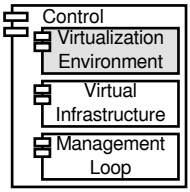


Figure 6.20.: Orientation example

To help orientation, each introduction of subcomponent begins with an icon, highlighting the current subcomponent within the main component. Figure 6.20 gives an example, illustrating the first subcomponent of the main functional component control.

All main functional components contain the subcomponent Management Loop, intended for the task of driving the management loop and providing

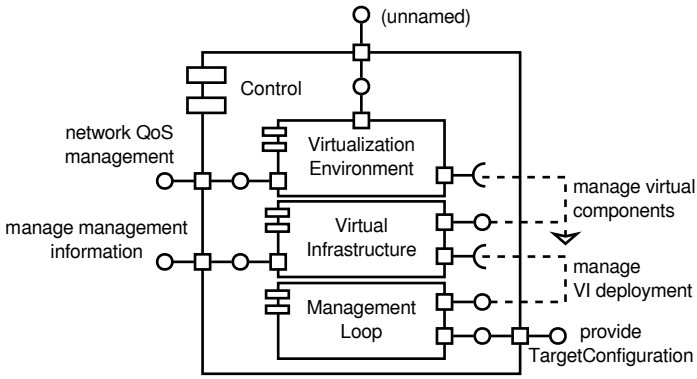


Figure 6.21.: The functional group Control.

the interface between the main components. The main functional components and their corresponding phases are:

`6.3.3.1 Control component`: phase 1, Act, Section 6.1.1
Definition of a target configuration

`6.3.3.2 Translate component`: phase 2, Plan, Section 6.1.2
Development of a configuration strategy

`6.3.3.3 Configure component`: phase 3, Do, Section 6.1.3
Effective component management

`6.3.3.4 Monitor component`: phase 4, Check, Section 6.1.4
Collecting performance data

6.3.3.1. Control component

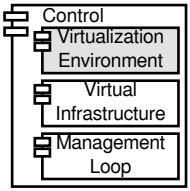
The main concern of the functional component Control are management functions guiding and controlling how management systems configure VEs,

Subcomponent	Realised task
Virtualization Environment	process interaction
Virtual Infrastructure	process interaction
Management Loop	coordination, event handling

Table 6.4.: Architectural components for Control management tasks

by modifying of management information. Figure 6.21 shows this component's decomposition into subcomponents and Table 6.4 links subcomponents with the tasks identified in Section 6.1.

With the unnamed interface, the Control component is expected to be extended or even split into more subcomponents. Therefore, this component is described more detailed in depth than Virtual Infrastructure and Management Loop, by explicitly introducing internal interfaces *manage virtual components* and *manage VI deployment*. Internal interfaces are exposed only to other Control subcomponents. The intended internal relations are depicted as dashed lines between facets and receptacles in Figure 6.21. The functionalities provided by the subcomponents are:



Virtualization Environment The Virtualization Environment subcomponent instantiates, advances and finalises the life cycles of physical resources, as well as virtual components, networks and infrastructures, as described in Section 5.2. The *network QoS management* interface contains all management functions relating to network QoS management, the focus of this thesis. All other functionality is attributed to the unnamed interface. The intended effect is to have the Virtualization Environment component always as the service access point, when interacting with external components and systems, including human managers.

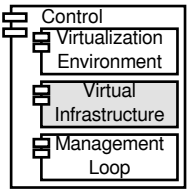
All functions belonging to an interface realised by this component are atomic transactions, finalized with success or failure replies. The important aspect of the transaction is read stability [KeEi 06], ensuring that the

Management Loop component always accesses the results from successfully completed transactions. All replies concern changes to management information, not the implementation thereof. There are separate management functions, pertaining to the effective implementation and its progress.

The interface and its functions, exposed by this component, is:

`network QoS management`: Manage links and topologies to expose features described by QoS Categories. The management functions target either QoS Categories (QoS domain), virtual links or topologies (presentation domain), as well as logical links (realisation domain).

- Create, modify and remove *QoS Category*. *QoS Category*s are a collection of *QoS Parameter*s, where parameters have values or value ranges, representative for the QoS they describe. The *QoS Category*s are used as QoS target by *Association*s.
- Create, modify and remove *Association*s. Managers create *Association*s to attribute *QoS Category*s to parts of the VE. As modification, the attributed *QoS Category* can be replaced, as well as the *NetworkItemSpec*s, grouped by the *Association*. The removal of an *Association* withdraws the QoS settings and monitoring pertaining to the *Association*s elements.
- Obtain QoS reports. Get the specification, realisation and assessment aspects of *QoS Category*s from *Association*s. This function triggers the `Evaluate()` method of *PropagationRule* objects, to produce the assessments, stating if, or even how well, the specified QoS has been achieved. This data is condensed into a single report, together, with data from past requests or special monitoring events; the `history` of the attributes belonging to the presentation domain.
- Enforce QoS. Have the management system realise a specified *Association* and its associated QoS with specified restrictions. Usually this function is used to exclude components or links from the realisation. Its main use is finding an alternative implementation as a response to frequent QoS issues, by excluding subpaths of the current implementation.



Virtual Infrastructure The Virtual Infrastructure component handles virtual components, links and their combination to VIs. This component influences the target configuration, by creating and managing representations of VIs. Analogous to the Virtualization Environment component, all functions belonging to this subcomponent only modify stored management information and the functions of the internal interface must be realised as transactions.

This component's management functions are separated into the following two interfaces:

- manage management information
- manage virtual components

`manage management information`: Feed monitoring data back into the management loop, manage `history` of presentation domain MOs and indicate monitoring events. The functions exposed by this interface change the information stored in existing objects, representing the current state. These updates are single values and partly completed updates are not significant enough to warrant transaction semantics. The motivation for this interface is to enable the Monitoring component to feed its observations and conclusions back into the development of *TargetConfiguration*s.

- Update measured data. Monitoring data belonging to physical components is stored in their respective MOs belonging to the realisation domain. Data, representing the state of MOs of the presentation domain, must be provided, i.e. this subcomponent does not perform processing monitoring data. This data updates the realisation aspect of presentation domain MOs.
- Create, save and delete history data. All representation domain MOs can store snapshots of their current state, stored as `history`. This is used for trend analysis, reports and reviews as part of performance management (cf. Section 2.2.2). This component implements `history` management by overseeing the creating of snapshots for all components of specified VIs, the deleting of specified generation of snapshots from

specified VIs. As a third function, snapshots can be copied and saved outside the management system for persistent storage.

- Signal problems. The results of processing monitoring data may indicate a problem with the achieved network QoS. Typically problems are detected through threshold violations. Such events may demand immediate attending and must be communicated to the Control component and, depending on individual policies, to managers. The severity of an event and the MOs it concerns are provided to the signalling function and the actions taken through its implementation depend on the specified severity.

`manage virtual components`: This is an internal interface, intended to be used by the Virtualization Environment to implement life cycle operations, i.e. creating, modifying, grouping and deleting objects. Its main purpose is to request life cycle operations for virtual components and infrastructures. Request, in this case, means performing the operations on the stored management information, so that the management system will eventually perform life cycle operations as described in Section 5.2. In the organisation model, `IC 4` specifies, that all provisioning results in a VI. Consequently all created components and links are always part of a VI, throughout their life cycles. This and the delayed implementation of life cycle operation requests are the reason why there is not a precise counterpart for every operations in Section 5.2.

- Create, modify and delete VIs. These functions target the MO for VIs, which is required as "parent" or "container" for all *NetworkItemSpec* and *Association* objects. Consequently it can only be deleted if it does not contain any objects of these types or their derivations. Modifications to MOs of VIs include adding and removing components and links, as well as resource pool allocations and associating the VI with managers.

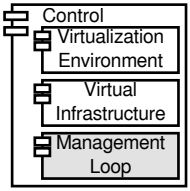
- Create, modify and remove *Associations*, as described for the `manage network QoS` interface.

- Create, modify and delete virtual components. Manage MOs that have the management system realise virtual components. When a

component is created, it must belong to a VI. All other prerequisites and settings are implementation specific. Other specifications, pertaining to the component, can be provided for the initial creation, or later, as a modification to the existing component.

- Create and delete links. Creating a link between two components has the management system connect these components. Usually this means they become part of the same OSI layer 2 subnet.

- Register and unregister components and infrastructures for realization. Only if a VI is registered for realisation, it becomes part of the *TargetConfiguration* compiled by the Management Loop component, which implies its placement and becoming operational. Individual components can be explicitly unregistered, so that they are not implemented, even if the remaining VI is. If they are not unregistered, their implementation is controlled through the VI.



MANAGEMENT LOOP The Management Loop component enables the realisation of VIs, by providing a complete target configuration. The target configuration is a selection of multiple independent VIs. This component's management functions are separated into the following two interfaces:

- manage VI deployment
- provide TargetConfiguration

`manage VI deployment`: This interface groups management functions controlling which and how VIs are included as part of the target configuration. With independent VIs, a system can partially realise a target configuration, by realising only some of its VIs. While this does not fulfil all configuration requests, the VE is still in a working and valid configuration. VIs are added or removed from the target configuration and constraints on its implementation are included or excluded from the configuration specification.

- Add and remove VIs. Make a VI part of the target configuration or remove it from the target configuration. The herein followed idea of target and current configurations implies that any component realised in the VE is legitimated by the target configuration. Removing components or entire VIs from the target configuration literally revokes the legitimacy of its realisation and consequently all allocations, components and links of the realisation must be purged from the VE.

- Add and remove realisation constraints. These functions allow adding further information to the target configuration to control the planning process. This is mostly used to provide the planner with information on previous (failed) configuration attempts, or special placement requirements that do not describe individual components. For instance, a constraint may exclude placing two specific VMs on the same host.

`provide TargetConfiguration`: Th interface provides the main input for the configuration of VEs and to obtain feedback on the realisation. This allows for implementations that reason and decide what to do about partial implementations.

- Create a recent target configuration and return the id of the created object. Target configurations are snapshots of the entire presentation domain of an information model instance. Consequently two subsequent calls to this function may well create to different target configurations. Especially with all the automation in place.

- Get target configuration. Retrieve a specific target configuration. The Management Loop keeps track of every released target configuration to correlate feedback on the realisation with the released target configuration. This information can be used for generating constraints, changing the target configuration, provide feedback to managers or even request action by managers.

- Report (un-)successful realization of VIs. This function tells the Management Loop about partial implementations of target configuration releases, or, depending on the implementation, on the progress of the implementation.

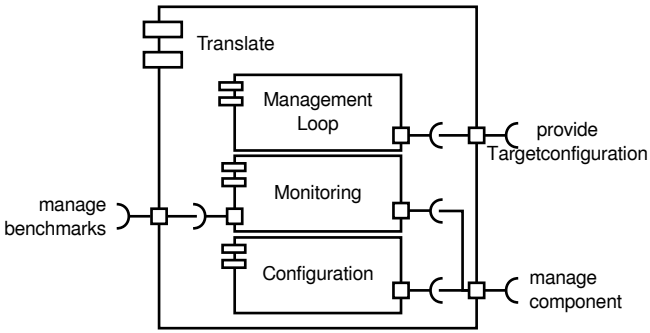


Figure 6.22.: The functional group Translate.

- Report constraint violation. A planner may have reasons to ignore some constraints when realising a target configuration. This function allows reporting of such behaviour, to discern explicit constraint violations from unwanted side effects.
- Implementation finished. This function tells the Management Loop that its current stored state on the implementation (progress) of a target configuration is considered final, and the planner is not going to provide any more feedback.
- Implementation error. If a target configuration cannot be implemented as the planner determined, implementing components can directly interact with this Management Loop to report an error outside of the normal monitoring procedure. Such errors are mostly out of the scope of network QoS management and require handling by other managing entities.

6.3.3.2. Translate component

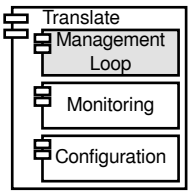
The Translate functional component, its subcomponent and the interfaces they use are illustrated in Figure 6.22. It is the main actor in implementing

Subcomponent	Realised task
Management Loop	dispatch control
Monitoring	develop monitoring/configuration strategy
Configuration	develop configuration strategy

Table 6.5.: Architectural components for Translate management tasks

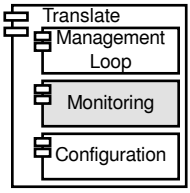
the management loop introduced in Section 6.1. This is also visible in the Figure, as it only uses interfaces provided by other components, but does not expose any interfaces to be controlled from external instances. Table 6.5 links the identified subcomponents with the tasks identified in Section 6.1.

This component's core task The core task of this component is the translation of the target configuration into sets of management operations for the components of the VE. The translations range from not changing anything, because the current configuration already matches the target configuration, on the one hand, and creating new virtual components and arranging them in newly created VNs, on the other hand. This is a high dimensional planning problem, as described in Section 6.1.2. Any planner is intended to be implemented as the Management Loop subcomponent, whereas the Configuration and Monitoring subcomponents generate the concrete configuration steps from the placement of VIs in the VE.

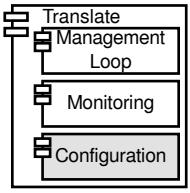


Management Loop The Management Loop subcomponent coordinates the realisation of target configurations. First it retrieves the target configuration using the `provide TargetConfiguration` interface exposed by the Control component and then oversees the development of the monitoring and configuration strategies. For network QoS management, an initial embedding is assumed, so that the next step of this subcomponent is the refinement of network QoS requirements, using the procedure introduced in Section 6.2.3. The result are *Path* objects from the translation domain, as described in Section 6.3.1, comprising the refinements of VI links, networks

and paths. These objects are the input from which the Monitoring and Configuration components derive their strategies. The other subcomponents report success or failure deriving their strategies to this component. In turn, this component has information on progress, problems, success and failure of the overall planning and can report to the Control component, also using the `provide TargetConfiguration` interface. For more thoroughly integrated management of VEs, there must at least be an additional planner for the embedding problem, which sees that all components are created and placed, as discussed in [Scha 12].



MONITORING The Monitoring subcomponent derives monitoring strategies as well as a configuration strategies from the refined *Path* objects, provided by the Management Loop subcomponent. The configuration strategies are targeted at the monitoring capabilities of the involved components, to ensure monitoring data is gathered. The monitoring strategies specify how monitoring data becomes performance data, which is suited to support assessments on the achieved network QoS, as described in the information model on page 230.



CONFIGURATION The Monitoring subcomponent derives configuration strategies, with the intention of realising network QoS. With the refinements provided by the Management Loop in the form of nested translation domain *Path* objects, this task means sorting the refined QoS requirements by component, to ultimately generate a set of configuration steps for each individual component. After this step the configuration strategies can be checked against the specifications of their components to evaluate whether the components are actually capable of implementing them. For instance, a host could be asked to block more data rate on a specific interface than the interface can provide. Consequently the configuration strategy cannot be implemented as specified. This is reported to the Management Loop which

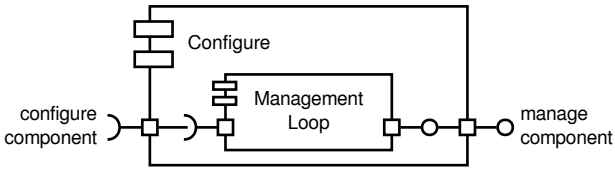


Figure 6.23.: The functional group Configure.

Subcomponent	Realised task
Management Loop	decode

Table 6.6.: Architectural components for Configure management tasks

can either devise new configuration strategies, or further escalate the problem to the Control component. By detecting such problems early, these problems can be handled through the management system, planners and managers, without implementing a configuration that has great potential to impede network QoS.

6.3.3.3. Configure

The Configure functional component, acts as a façade [FSBR 04] for the management of individual components. The interface exposed for this task is `manage component`. The Translate component uses this interface to trigger and delegate the realisation of configuration strategies. To uphold the herein used schema for describing functional components, this component's one task is realised by the Management Loop subcomponent, illustrated in Figure 6.23. The mapping to the resulting management tasks of Section 6.1 is shown in Table 6.6. The used `configure component` is only a substitute for the plethora of component specific configuration interfaces, which cannot be listed and described within the scope of this thesis and must always be implementation specific.

At this stage in the management loop, information may have been refined, consolidated and reordered so often, that it may not be possible for this component to attribute individual configuration steps to their originating VIs. If an error is encountered and the affected VI cannot be determined, the entire configuration strategy must be rolled back and reported as not implementable. To allow for partial implementations in the cases where this component can identify the affected component,

`manage component`: Receive configuration strategies for individual components and obtain feedback on its realisation.

- Realise strategy. The Management Loop is given a configuration strategy it is to convert into technology and implementation specific management operations for the specified component. After that, it performs these management operations on the component and reports back to the Translate component or may even directly notify the Control component of failures. When this function is called, it may be notified that an implementation may be partial, i.e. not all configuration steps must be realised. In this case the component may only notify the Control component of success or failure and must include the performed configuration steps in the report.

6.3.3.4. Monitor

The Monitor functional component, closes the loop approach, by observing the VE and interacting with the Control component correspondingly. Figure 6.24 illustrates this component and its three subcomponents, as well as the exposed and used interfaces. Mapping of the management tasks laid out in Section 6.1 onto individual subcomponents is shown in Table 6.7. The only exposed interface is `manage benchmarks`, which is used by the Translate component, to instruct this component how to contribute to the overall network QoS management. The actual feedback and management is implemented in the Evaluation subcomponent, while the Measurement subcomponent is a façade for the monitoring of individual components, analogous to the Configure component.

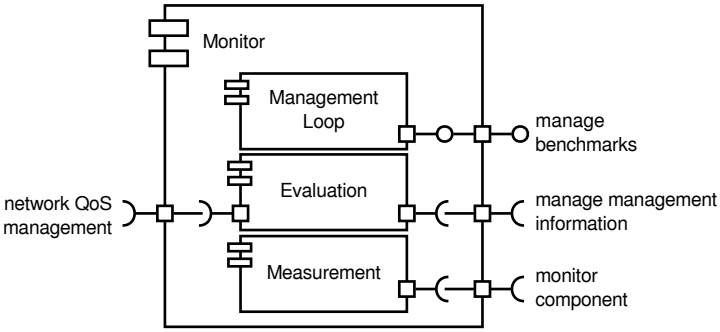
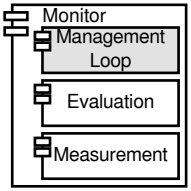


Figure 6.24.: The functional group Monitor.

Subcomponent	Realised task
Management Loop	develop benchmarks
Evaluation	evaluate monitoring data
Measurement	perform measurements

Table 6.7.: Architectural components for Monitor management tasks

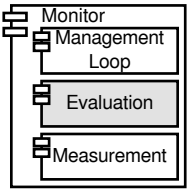
The three subcomponents and the exposed interface are:



MANAGEMENT LOOP The Management Loop subcomponent is the controlling entity for Monitor component. It ensures monitoring strategies are implemented and their results provided to the Control component. To that end it implements the `manage benchmarks` interface, which enables other components to send monitoring strategies for implementation and further control them after initial implementation.

`manage benchmarks`: Manage implementations of monitoring strategies.

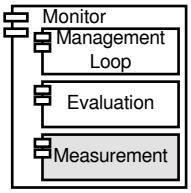
- Realise, replace and withdraw strategy. The Management Loop subcomponent is given a monitoring strategy it is to implement. According to the received strategy, the Measurement component is instructed to monitorm specific components at specific intervals. The Evaluation component is instructed how to process sets of monitoring data and act according to the results. In contrast configuration strategies, monitoring strategies can change without a corresponding change to the components in VEs, for instance, if the *QoS Category* of an *Association* is changed to demand another kind of monitoring. Therefore the implementations of monitoring strategies are tracked, so that they can be replaced or even withdrawn from the system.
- Deactivate an reactivate strategy. To prevent or suspend automated management and updates of management information, monitoring strategies can be deactivated and reactivated at a later point in time. This is mainly used for large changes to the VE, where the management system is creating temporary heavy load that may negatively affect network QoS.
- Perform evaluation. In addition to the regular measuring and evaluating of monitoring data, a one time performance of an implemented monitoring strategy can be requested, to update the management systems knowledge about the current configuration of the VE.



Evaluation The Evaluation subcomponent processes monitoring data and acts accordingly. This activity ranges from storing the monitoring data using the `manage management information` interface, to automatically perform network QoS management using the `network QoS management` interface, both exposed by the Control component. It is the initially provided monitoring strategies, that describe how monitoring data must be correlated to yield meaningful evaluations of the implementations of *QoS Categories*. This can include monitoring threshold values and automated trend analysis.

Type	Description	Example
access data	access management information	update measured data
singular instruction	have components perform a task	perform evaluation
compound instruction	step components through a task list	realise strategy

Table 6.8.: Basic interaction types



PERFORM MEASUREMENTS The Measurement subcomponent provides the basis for all contributions of the Monitor functional component. It is a façade, analogous to the Configure component, but with the intention of monitoring instead of altering their behaviours. Its main task is obtaining monitoring data and provide it in a generalised manner, so that the data is comparable to data obtained from other implementations. To achieve this goal, the Measurement component must perform adaptations and mappings, as previously published in [Metz 11].

6.3.4. Communication model

The communication model completes this management architecture, by describing the communicating entities of this architecture and the relevant aspects of their interaction. The communication model describes such communication that takes place for the monitoring and control of potentially dispersed resources [HAN 99]. Applied to the approach followed in this architecture, this targets the implementations of the management loop phases. Hence, the communicating entities are the main functional components Control, Translate, Configuration and Monitor and their interactions are method invocation, progress and status reports, as well as failure notifications. Relevant aspects of these interactions are communication mechanisms, describing interaction procedures, required state information and the duration of an interaction.

For every interaction, there are always two communicating entities, an *initiator* and a *target*. The management functions specified in the functional

model are the reasons for interaction. There are three discernible interaction types, named in Table 6.8, including a short description and an example. With hosts realising multiple similar virtual components, it must be assumed that when two entities start communicating, there are often multiple reasons, i.e. multiple requests. To handle this, the generic scheme, common to all interaction types, is:

1. The entities agree on the interaction's intention.
2. The entities agree on the interaction's subjects.
3. The intention is fulfilled.
4. The interaction is finalised.

Having a common structure with these four explicit elements allows for various extensions and varying degrees of error handling. This scheme implies a state on each interaction, which must be maintained until the interaction is finalized. The concrete state information is specific to the interaction type and intention.

The details of each interaction type are described in the following. For each interaction type a prototypical interaction is shown, where an interaction progresses without problems. Each message in the examples is an arrow in a sequence diagram, labelled with the intention of the message and a specification for the payload. The intention also represents the progress of the interaction. To denote the payload items, the Backus-Naur form (BNF), as introduced (for example) in [Schö 03], is used to describe what information must be provided for the various interaction types.

6.3.4.1. Access Data

This interaction type is used to access already existing MOs. Management information is either retrieved from the information base, or updated. Figure 6.25 shows the prototypical procedures for retrieving and updating management information. The *initiator* tells the *target* the intention to either `update` or `retrieve` information, the *target* acknowledges it is `READY` to fulfil the requests, and, in the end, the interaction is finalized by the *initiator* signalling it is `done`.

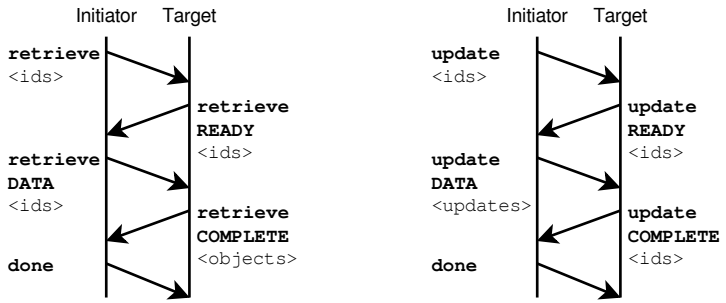


Figure 6.25.: Prototypical interactions to `retrieve` and `update` management information

When the *initiator* first signals `update` or `retrieve`, it also provides a list of references to the objects that are of concern to the *initiator*. As a reference the objects ids, as introduced with the information model, are used. The *target* replies the subset of the requested references it can retrieve or update, when signalling `READY`. For a complete and successful interaction, these list of object ids are the same. An empty list means the *target* cannot fulfil the requests. Partial lists could indicate, that the *target* is responsible for only some MOs and the *initiator* must find other recipients for other MOs. The communication model allows for such constructs, yet the distributed implementation of single functional components is outside the scope of this thesis.

The *initiator* signals `DATA` to fulfil the requests. For retrieving data, it merely tells the *target* to deliver the requested objects. For updating data it sends the updated data and the object for which it is intended. For a successful interaction, the *target* responds with `COMPLETE`, which either contains requested objects or the ids of updated objects. It remains with the *target* to decide if a request is reported `COMPLETE` as soon as it finishes, or if it waits and reports multiple requests as `COMPLETE` with one message.

The interaction may be ended by either entity, through signalling `done` to

the other. However, for a successful interaction, only the *initiator* signals `done` when it has received all objects/ids it expects.

The BNF for the message payloads is:

```
objects ::= <object> | <object> <objects>
ids     ::=     <id> | <id>, <ids>
updates ::= <update> | <update> <updates>


object  ::= <A MO instance as specified in the
              information model>
id      ::= <An object id as specified in the information
             model>
update  ::= <id> <A specification of the data that is
                  updated>
```

6.3.4.2. Singular Instruction

The idea of a singular instruction is to have the *target* perform a task and the tasks outcome does not affect the interaction's progress. The tasks are management functions, that may be parametrised. For a single call to a management function, the interaction is very similar to a remote procedure call, as described, for instance, in [TaWe 10]. Figure 6.26 illustrates a prototypical interaction. The first three steps of the generic scheme are analogous to the *access data* interaction: the *initiator* signals the functions it would like to call, the *target* replies with a list of functions it can perform, the concrete calls with parameters are made and whatever the functions return is sent from the *target* back to the *initiator*.

All functions that are going to be called must not have any dependencies and no implications must be made pertaining to ordering, execution time or success. Therefore, the *target* must attribute every returned value to the function and parameters that yielded it. The *initiator* must send all parametrised function calls in a singe `DATA` message, while the *target* may decide if a call is reported `COMPLETE` as soon as it finishes, or if it waits and reports multiple calls as `COMPLETE` with one message.

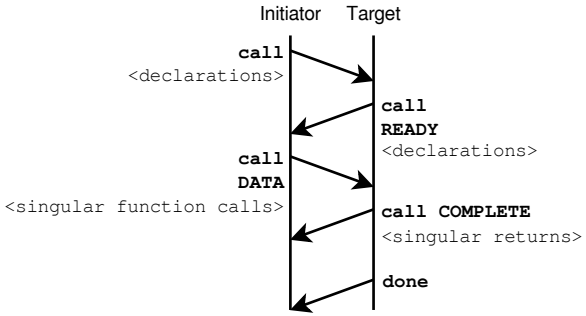


Figure 6.26.: Prototypical interaction to `call` singular management functions

The interaction may be ended by either entity, through signalling `done` to the other. The *initiator* signals `done` as soon as it does not want to receive any (further) return values. Usually, the *target* signals `done` when it is not going to send any more return values.

The BNF for the message payloads is:

```
declarations     ::= <declaration> | <declaration> <declarations>
singular function calls ::= <singular function call> |
                            <singular function call>,
                            <singular function calls>
singular function call  ::= <function name> <values>
singular returns ::= <singular return> |
                     <singular return> <singular returns>
parameters       ::= <parameter> | <parameter> <parameters>
values           ::= <value> | <value> <values>
singular return  ::= <function name> <values> : <value>
declaration      ::= <return type> <function name>
                     <parameters>
parameter        ::= <type> <name>
return type      ::= <type>
function name    ::= <name>
type             ::= <A data type>
name             ::= <A name>
value            ::= <Data of a specific <type>>
```

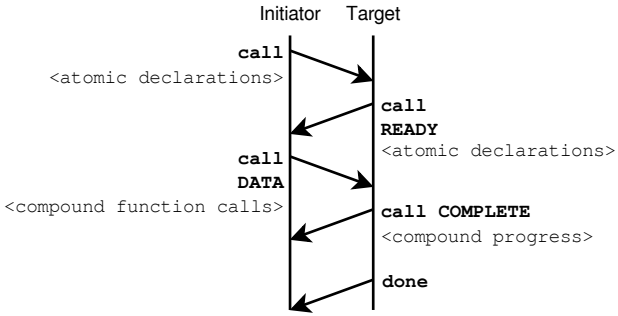


Figure 6.27.: Prototypical interaction to `call` compound management functions

6.3.4.3. Compound Instruction

Compound instruction is the interaction type intended for all management functions, that require functional components to perform a sequence of tasks successfully, for example implement a configuration strategy. Such management functions either require the entire sequence is performed successfully, or partial implementation is acceptable. The prototypical interaction to *compound instruction* is depicted in Figure 6.27 and is very similar to *singular instruction*, because it adheres to the same common scheme. Yet, *compound instruction* foresee more communication, depending on the length of the task sequence.

Analogous to *singular instruction*, the *initiator* starts an interaction of this type, with a list of functions it intends to call and the *target* acknowledges the list of declarations, before the *initiator* makes parametrised function calls Specific to *compound instruction* is, that each element of the initial list carries additional information on whether partial implementations are acceptable, or the entire task sequence must be completed successfully. The task sequence is a parameter of the function. When the functions are called, the length of the task sequence is provided as an additional parameter.

For every called function where partial implementations are acceptable, the

target must report on every individual completed task, successful or not. This results in as many reports for an individual function, as the length of its task sequence. While the *target* may include multiple reports in a single `COMPLETE` message, it may not group multiple reports originating from the same function call.

For every called function where every task must finish successfully, only the last task must be reported, if successful. With the first task that does not finish successfully, the *target* must report the failure and the number of the failed task. Ensuring a stable configuration after a failed task must be accomplished by the implementing functions and is not part of the communication model.

The interaction may be ended by either entity, through signalling `done` to the other. Usually, the *target* signals `done` after reporting on the last finished task of each called function.

The BNF for the message payloads is:

```
atomic declarations    ::=  <atomic declaration> |
                            <atomic declaration>
                            <atomic declarations>
compound function calls ::= <compound function call> |
                            <compound function call>,
                            <compound function calls>
<compound progress>    ::=  <singular return> <task> |
                            <singular return> <task>
                            <compound progress>

compound function call ::= <singular function call> <tasks>
atomic declaration     ::= atomic  <declaration> |
                           partial <declaration>
tasks                  ::= <The task list length>
task                   ::= <The completed task number>
```

6.4. Summary

This chapter introduces a management architecture for network QoS in VEs. Section 6.1 started out with an approach of continuous improvement, with the idea of having the VE converge towards behaving as specified by abstract models of VIs. Section 6.2 introduces a refinement procedure which is the tool enabling splitting up the overall management task, as suggested by the approach. The procedure derives management operations that modify the VE towards behaving as described by VI specifications.

Having laid out how network QoS management can be facilitated, Section 6.3 introduces the management architecture for developing management systems, that perform management with the presented approach. The architecture is specified reusing existing approaches where appropriate, most importantly the model for VIs from [FAP 10] and the model for QoS from [MNM Roel 05]. Also the architecture is based on the CIM to reuse existing infrastructure models.

The architecture was developed with the guides and intentions discussed in Chapter 5. The following Chapter 7 analyses how this architecture fits the stated goals and the requirements from Chapter 3 and introduces a prototypical implementation, to illustrate this approach is viable for performing network QoS management in VEs.

Chapter 7

Assessment

This chapter analyses the architecture developed in Chapter 6, to show its applicability for network QoS management in VEs. Section 7.1 discusses individual requirements and how the developed architecture may be used to create management systems fulfilling these requirements. Afterwards, a prototypical implementation is introduced in Section 7.2, before Section 7.2.4 showcases the realisation of network QoS management in VEs.

7.1. Validation

This section steps through all identified requirements and shows how they can be fulfilled, using the introduced architecture. Following the requirements on network QoS management from Section 3.4, the same procedure is applied to the requirements derived in Section 5.3.

Network QoS management requirements

Req.#1 Resource allocation to virtual machines The bare technological capacity to allocate resources must be available, as per prerequi-

site Req.#12. For VMs the capacity usually lies with the VMM, which can be managed as per prerequisite Req.#13.

The information model foresees storing resource requirements with any *Node* object, a capability inherited from the original model introduced in [FAP 10]. This includes *DTE* objects, the intended class for VMs. A *DTE* is part of a *VirtualInfrastructure*, which is scheduled for implementation by becoming part of the *TargetConfiguration*. As part of the target configuration, a management strategy will be devised to place a VM on an active host from the *CurrentConfiguration*. This results in a *CIM_VirtualSystem* object for the VM, which is associated with the *CIM_VirtualSystemManagementService* object for the VMM, belonging to the host chosen for the placement. Through these stages, high level resource allocation settings for the *Node* are mapped onto the realising components.

Req.#2 Resource allocation to virtual network components *DCE* objects for VN components derive from *Node*, analogous to *DTE* objects for VMs. Consequently this requirement can be fulfilled analogous to Req.#1, when *DCE* are realised as VMs. Section 6.2.1 identifies this and other conceivable methods for implementing VN components. Their technological capability to perform resource allocations is required per prerequisites Req.#11 and Req.#13.

The introduced refinement method identifies all network components, especially those not managed as *DCE* objects of the presentation domain. Besides their identification, the refinement method also determines the use of each component, i.e. its links with QoS requirements. Every component exposes its prerequisite functionality using the specified *QoSTypedInterface*, enabling the realisation of QoS requirements. In combination, management strategies can perform resource allocations to VN components, independent from their realisation.

Req.#3 QoS links can be specified with virtual endpoints Managers specify *Associations* using objects from the presentation domain. Endpoints may be virtual as well as physical, when specifying QoS links.

`Req.#4` SUPPORT FOR DIFFERENT TYPES OF QOS PATHS Every managed object from the presentation domain is eventually mapped to a realising component. This allows for specifications of *Associations* with *QoS Categorys* independent from a components role in the infrastructure. Restrictions on the paths that can be realised result from management privileges and the management systems capability to perform management operations on the involved components. The nine discernible path types identified for this requirement can be implemented and managed as follows:

`(VSi, VSi)`: Both endpoints belong to the same VI. This particular QoS path is the *Path* class in the presentation domain and can be implemented using the refinement procedure as introduced.

`(VWS, VSi)`: Analogous to (VSi, VSi). A *VWS* derivation from *DTE* allows for equivalent behaviour in QoS path management, while *VWS* objects can still have unique properties.

`(VSe, VSi)`: Analogous to (VSi, VSi).

`(VSe, VSe)`: Analogous to (VSi, VSi) if the path manager is responsible for the VIs to which the endpoints belong. Otherwise, if an endpoint is not within the manager's responsibility domain, an additional MO of the presentation domain must be created as endpoint for the path. The newly created MO must be associated with the original MO, so that management strategies can be derived correctly.

`(VWS, VSe)`: Analogous to (VSe, VSe).

`(VWS, RAS)`: Analogous to (VSe, VSe). The RAS is never part of the managers responsibility, thus requires a “proxy” presentation MO.

`(VSe, Outside)`: The refinement procedure can detect path segments, that are beyond the management system's access. These segments are classified as `ExtraVI` and not refined any further. This allows the management system to perform network QoS management to the boundary of its controlled area. An *Outside* specialisation of *DTE* can act as a VI component that cannot be configured, but be a *Path* endpoint.

`(VWS, Outside)`: Analogous to (VSe, Outside).

`(RAS, Outside)`: When the RAS is within a VI, this segment is analogous to (VSe, Outside). Usually, either endpoint is outside a managers responsibility domain. Still, the refinement procedure can detect such segments and management may be performed on them.

Through generalisation and the capabilities of the refinement method, this requirement can be fulfilled.

Req.#5 VIRTUAL INFRASTRUCTURE SEMANTICS FOR PERFORMING MANAGEMENT Through separating presentation and realisation MOs, this architecture allows for VI semantics for performing management.

Req.#6 MONITORING STRUCTURES FOR VIS Monitoring strategies can be derived directly from the result of the refinement procedure, which is a refinement of abstract network topologies to the implementing components. This allows for placement specific monitoring set ups and also adequate handling of migration.

Req.#7 MANAGEMENT USERS ARE RESTRICTED TO THEIR ASSOCIATED VIS Managers are given the capabilities to perform management on their individual VIs. The separation between presentation and realisation avoids any necessity to provide managers with information that does not belong to their VIs. As used introduced for requirement Req.#4, proxy objects can augment a managers information base explicitly. The possibility to restrict managers to their associated VIs is provided through the separation of presentation and realisation MOs.

Req.#8 MULTIPLE CONCURRENT AND CUSTOMER MANAGERS The management loop, formulated in Section 6.1, as the core approach to network QoS management in VEs, foresees VI specifications that have no immediate effect. To implement VI specifications in the VE, a snapshot of the VI specifications is created in the form of a target configuration. The function model in Section 6.3.3 describes how a consistent state of the specifications for the sake of creating a target configuration can be achieved. The management loop separates the configuration of VEs into streamlined phases. By incorporating a mechanism ensuring consistent VI specifications, it becomes possible to have multiple managers concurrently performing management, without negative effects on the realisation phase.

Req.#9 AUTOMATED ADAPTATION OF LINKS Through the separation of presentation and realisation of components and paths, their MOs are implementation independent. The continuous evaluation of the management loop ensures that changes to the specifications are propagated to the implementations. The mapping performed by the refinement procedure ensures changes to the VI's state are handled correctly.

Req.#10 AUTOMATED ENFORCEMENT OF NETWORK QOS REQUIREMENTS The management loop includes a feedback phase, where monitoring data is evaluated and reported back to the phase determining the target configuration. This enables the detection of QoS problems and allows the system to react accordingly. The system may automatically adapt *QoS Category*s to match the current needs. By realising management functions currently attributed to the unnamed interface of the Control component, the management loop's capabilities to enforce network QoS can be extended, for example perform automated migration of components.

Behavioural requirements

Req.#17 RELEASE ALLOCATED RESOURCES By storing the refined links and topologies used for the generation of configuration strategies, an "antistrategy" can be devised, undoing the resource allocations performed to realise network QoS. The management loop can perform this task implicitly when there are changes in the new target configuration. The required versioning of *TargetConfiguration*s and history of path refinements are foreseen in the architecture.

Req.#18 MANAGEMENT MAY BE PERFORMED ON THE ENTIRE VIRTUALIZATION ENVIRONMENT The refinement procedure leads to performing management on all components. To add another implementation or technology, the interfaces and properties specified in the QoS domain in the information model must be implemented. The generalised interfaces and refinement procedure allow for the configuration of the entire VE.

Req.#19 Full information about the current VI placement The refinements performed by the refinement procedure includes all intermediary steps towards implementing virtual topologies, including the final physical host realising the component. Through the streamlined management task realised in the management loop, this information can be used for effectively performing management, without the need to expose this information to the managers.

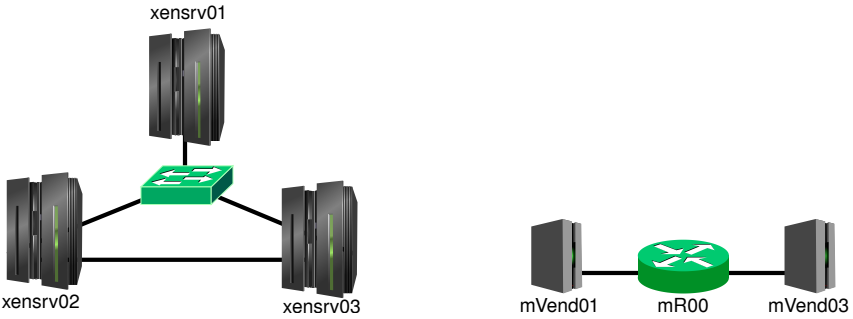
Req.#20 Reverse mappings from components to VIs By analysing the results from the refinement procedure, all VIs using a specific component can be determined. Input for reverse mapping are the same refinements as for requirement Req.#19 and can be used analogously.

Req.#21 Resources can be allocated within a hierarchy of management users Hierarchies of customers have been taken into account for creating the organisation model. While different roles are responsible for different aspects of provisioning platforms, there are also the primary and secondary customers, allowing for hierarchies of customers for VIs.

Req.#22 Management users can be tied to life cycles or life cycle phases This requirement can be fulfilled, by restricting management users to individual interfaces of the Control component. While the architecture allows for such behaviour, interfaces beyond network QoS management have not been specified as part of this thesis.

7.2. Prototype

This section showcases the implementation of a prototype, illustrating an effective, working management approach. The focus lies with the automatic transformation of abstract VI specifications into technology specific management operations for the VE. Figure 7.1 shows the main topologies used throughout this section.



(a) The physical infrastructure (b) The main virtual infrastructure

Figure 7.1.: Core infrastructures of the Xen lab

Figure 7.1(a) shows the physical infrastructure, consisting of three hosts, named `xensrv01`, `xensrv02` and `xensrv03`. There is a switch interconnecting all three hosts and one direct link between `xensrv02` and `xensrv03`. The switch does not have any QoS capabilities and is therefore not subject to management. The hosts use the Xen hypervisor as VMM, and Open vSwitch (OVS) for the provisioning virtual switches. Figure 7.1(b) shows the main VI, two endpoints and a router, which will be used as an example throughout this chapter.

The prototype is developed towards meeting the characteristics of the Xen lab used for development and experimentation. Some corners are cut and specific implementations are made, where a generic approach would have generalised representations and specialised refinements only where needed. This results, for instance, in the fact that the specific descriptions of VMs, as provided by the Xen hypervisor, and virtual switches, as provided by OVS, are directly usable as the current configuration of the VE, without further processing or generalisation. The prototype's implementation follows the common model-view-controller (MVC) pattern, because it suits the abstract management loop and the concept of target and current configurations. This section introduces the implementation. Afterwards, it is used in Section 7.2.4 to manage network QoS in the experimental envi-

ronment, depicted in Figure 7.1, enforcing network QoS in a set up where resource shortage lead to QoS degradation.

7.2.1. MVC architecture

The MVC pattern is a common software design pattern where all forms of interaction and functionality either modify a model or are triggered as a response to the model, or as a change thereof. The model is self contained and complete for the scope of the developed software system. Every software component that accesses the model is a view and acts as a representation of a model, or a part thereof. Views have associated controllers, that modify model or view in response to a change in one or the other. The model also holds central logic, realising the core concepts of the software system.

This pattern is chosen for the prototype. It is used to devise components of the intended architecture. Different management views can be designed to provide adequate high level management interfaces for users. Every subsystem that actually performs management on components is a view on the component aspects of the model. The entire management loop is part of the model. Controllers are the agents responsible for configuring components and providing monitoring data as well as the management interfaces. There are two management interfaces, one for reporting to the users about the status of VIs and another one for creating VI specifications.

Figure 7.2 illustrates architectural components, having applied the MVC pattern to the problem at hand. The model consists of a *data model*, a *transaction engine* and the *translation logic*. The data model contains all managed objects for all managed components, collected monitoring data and all configurations and requirements pertaining to network QoS.

The transaction engine ensures consistent states when creating the target configuration. For larger implementations this could be a relational database system implementing transactions as common for modern systems as described in [KeEi 06]. In this implementation, focusing on the automated translation from abstract VIs to technology specific management operations, the transaction engine will directly write through any data.

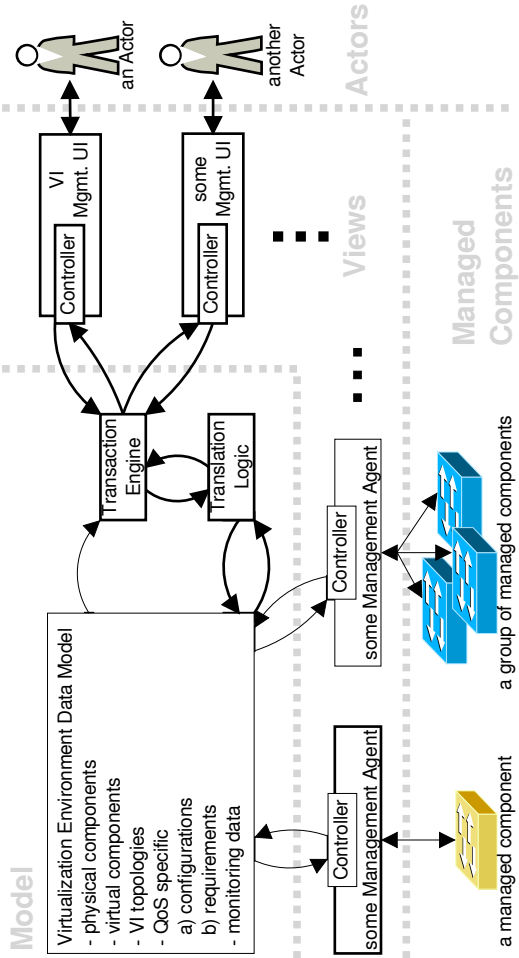


Figure 7.2.: An overview of the architectural components and their interactions

The translation logic is the implementation of the translation phase and the core aspect of the prototypical implementation. It refines the abstract VI specifications to be mapped onto the current configuration and develops configuration strategies, having the management agents change the experimental set up accordingly.

The VE for the following evaluation has Linux VMs on virtualization hosts using the Xen hypervisor and OVS for virtual and local switches.

The main implementation is the Translate component, using the generalised approaches introduced with the architecture, for transforming VI specifications into management operations for the given VE. The Configure component is implemented as a simplified agent realising a QoS interface to implement configuration strategies.

The Monitor and Control components are realised levering the available management tools "Shinken" and "Salt", respectively. The latter is also used to hand configuration strategies to agents.

All data is stored using text files, common configurations as well as VI specifications and feedback provided by the monitoring system. Monitoring strategies are formulated as Shinken configurations and configuration strategies as Salt states. The following Section 7.2.2 introduces used management information and its notation, before Section 7.2.3 shows the implementation of the management loop. After that, Section 7.2.4 applies the management system to a VE and shows an experiment illustrating the effectiveness of the system.

7.2.2. Management information

The information required for the prototype are physical and VIs, QoS paths and a QoS Category. The general notation for management information is JSON. It allows for structured data, named data and JSON is the standard information presentation method for Xen and Open vSwitch management tools and Shinken reports. Salt uses YAML, very similar to JSON, but

noticeably different in syntax. As an effect, information exchanged with managers is JSON while configuration strategies are written in YAML.

Physical and VIs are specified a collection of nodes and links. There is meta information helping the usability of the system, by giving names to specifications and there are constraints, for embedding heuristics and QoS requirements. Listing 1 shows a minimal example of a VI specification. The specification is named `Simple VM example` and holds a single specification for a node named InitialTestVM. For this prototype names must be unique because they also serve the purpose of an id, to reference a managed object.

Listing 1: Minimal VI specification

```
{
"meta": {
   "name": "Simple VM example"
   },
"nodes": [
   { "name": "InitialTestVM" }
   ]
}
```

When this VI specification becomes part of the target configuration, it will be implemented. The management system expects the hypervisor to know about a VM named `InitialTestVM`, or have it create one, with default settings. Being part of the target configuration, the VM must also be activated. This would be the two steps of a corresponding configuration strategy. Listing 2 shows how this is implemented in the prototype. Checking or creating VMs is not an explicit step, but implicitly done prior to starting a VM. Hence, the configuration strategy merely tells a host to transform a VM named InitialTestVM into the *running* state.

Listing 2: Minimal configuration strategy

```
InitialTestVM:
   vm:
      - running
```

Monitoring is performed using the Shinken, which has a predefined format for monitoring data. Listing 3 shows an example for a single record of monitoring data as provided through Shinken. The most important fields are the monitored component, the performed monitoring task, a short status message and task specific

Listing 3: Exemplary monitoring data

```
('1395389295', 'xensrv02', 'runningVMs',
 'WARNING', '1', 'SOFT', '8.37400889397',
 '0.875785112381', 'Idle machine', "")
```

data. These are the second, third, ninth and tenth field in the example, respectively. In this example, the monitoring data originates from *xensrv02* as result of the task *runningVMs*, the short status message is *Idle machine* and no task specific data.

Listing 4: VI specification for experiment set up

```
{
"meta":
   { "name": "Zwei Kisten ein Router" },
"nodes": [
   { "name": "mVend01" }, { "name": "mVend03" },
   { "name": "mR00",
     "type": "dce",
     "layer": "3"
   }],
"links": [
   {  "name"     : "mVend01net1",
      "fromto"   : [ "mVend01", "net1" ],
      "type"     : "l2plain"
   },
   {  "name"     : "mVend03net3",
      "fromto"   : [ "mVend03", "net3" ],
      "type"     : "l2plain"
   },
   {  "name"     : "mR00net1",
      "fromto"   : [ "mR00", "net1" ],
      "type"     : "l2plain"
   },
   {  "name"     : "mR00net3",
      "fromto"   : [ "mR00", "net3" ],
      "type"     : "l2plain"
   }],
"constraints": [
   {  "name"     : "namesaregood",
      "type"     : "disjointplinks",
      "subjects" : [ "mR00net3", "mR00net1" ]
   },
   {  "name"     : "one_host_for_each_vm",
      "type"     : "notonthesamehost",
      "subjects" : [ "mVend01", "mVend03", "mR00" ]
   }]
}
```

Listing 4 shows a more detailed example, featuring all four information blocks, meta information, nodes, links and constraints. It is the specification for the VI depicted in Figure 7.1(b). There are three nodes: mVend01, mVend03, mR00. Two of the nodes are "normal" VMs, while mR00 is a virtual router, a DCE active on OSI layer 3.

Without any further specification, the management system assumes a VM is a DTE, as is the case for mVend01 and mVend03. For the virtual router mR00, there are explicit statements, telling the management system the VMs `type` is *dce* and the `layer` attribute gives information on the highest OSI layer, the DCE implements. For routers this is layer 3.

Following the specifications of nodes, there are the specifications of links interconnecting the nodes. In Listing 4 the links' `type` is always *l2plain*; a basic duplex connectivity between two nodes on OSI layer 2. With the attribute `fromto` the two endpoints of a layer 2 link are identified. If both endpoints are VMs, a virtual point to point link is assumed and will be implemented. An endpoint that is not a VM means this link is an uplink of a VM (or network) to a network, as net1 and net3 in this example. These networks will be implemented as virtual OSI layer 2 topologies allowing communication between all components with an uplink.

For a management system that can deterministically set up experimentation environments, it was necessary to implement two placement constraints, *disjointplinks* and *notonthesamehost*, also shown in Listing 4. Constraints of any type are always applied to a list of subjects which must be specified with the constraint. With this scheme, every constraint must have `type` and `subjects` attributes set, for a complete specification. Constraints of type *disjointplinks* target a list of links, instructing the management system not to share a physical link between any two list items. The other constraint is of type *notonthesamehost*, analogous to *disjointplinks*, but targeting nodes.

The configuration describing the physical infrastructure follows the same pattern an principle. Nodes are assumed of type *host*, with *ethswitch* as a special case for physical Ethernet switches. There are two types used to describe the links of a physical infrastructure *hostswitch* and *hosthost*

indicating an uplink of a host to a switch and a direct link between two hosts, respectively. Both link types require the attribute `nic` to know which NIC a host uses for this link.

Listing 5: QoS path demanding 200 MBit/s

```
"constraints": [
   {  "name"     : "datarate",
      "type"     : "enforcedqospath",
      "subjects" : [ "mVend01", "mVend03" ],
      "min_rate" : 200
   }
   ]
```

Listing 5 shows an additional constraint, specifying network QoS on the VI specified in Listing 4. It requires a minimum data rate of 200 MBit/s on the path between mVend01 and mVend03, in both directions. The following Section 7.2.3 describes this prototype's implementation of the management loop introduced in Section 6.1. The refinement procedure preparing this QoS requirement for implementation is part of the translation.

7.2.3. Management loop

The implementation of the management loop mainly incorporates, Shinken for monitoring, Salt for configuration of components, and two newly developed automatons. One automaton realises the translation of specifications to strategies, while the other newly implemented automaton augments Salt's capabilities by implementing network QoS management using the *QoS Interface* as described in the information model. To control the management loop and provide management information, text files are used instead of an explicit implementation of a management interface. The following describes the implementations of the management phases introduced in Section 6:

`7.2.3.1 Control`: Definition of a target configuration

`7.2.3.2 Translation`: Development of a configuration strategy

`7.2.3.3 Configuration`: Effective component management

`7.2.3.4 Monitoring`: Collecting performance data

7.2.3.1. Control

The management loop is managed using text files. A VI is specified as JSON object and stored using one file per object. Listing 4 is a complete example for a VI specification. To change a VI, its text file must be edited. To schedule a VI to become part of a target configuration, a symbolic link to its specification must be created in a special directory `deploy`. Using a versioning system, e.g. CVS, on the text files allows the retrieval of the most recent accepted configurations, acting as a rudimentary implementation of the transaction engine, as depicted in Figure 7.2, providing read stability when generating target configurations.

For the physical infrastructure there is a JSON object similarly as for VIs, except for it is interpreted differently and may never be scheduled for the target configuration. To distinguish the physical infrastructure specification from VI specifications, it has to have the filename `physical.setup`.

The scheduled VIs, the specification of the physical infrastructure and monitoring data constitute the input for the automaton facilitating the translation phase of the management loop, illustrated as *Translation Logic* in Figure 7.2.

The repetition of the management loop is realised using GNU `make`. The core functionality of `make` is executing recipes for building target files from dependency files. For the purpose of the prototype, the configuration strategies are the files that need building, while the VI specifications and monitoring data are the dependencies.

To reduce the workload, `make` implements a mechanism determining when and if a file needs to be built. This mechanism is based on timestamps in file system information: a file is only built if there is at least one dependency with a modification time more recent than the target file. This is trivially true for non existent files. Existing files will be rebuilt, if one dependency was altered since the last built. Mapped to this implementation, configuration strategies are only rebuilt if a VI specification or the special file containing only short status messages changes. This allows frequent calls to `make`, but target configurations are only rebuilt if need be.

The recipe for creating configuration strategies is calling the translation automaton. For an easier implementation, the recipe first calls the translation automaton and immediately afterwards triggers the implementation of configuration strategies, which is done using Salt.

7.2.3.2. Translation

The translation automaton could not be provided by available tools and is a completely new development. First it creates a target configuration from all its inputs. Then it then derives configuration strategies for the physical hosts from the target configuration. In this implementation and lab environment, there are always four configuration strategies: one strategy for each host, pertaining to its virtual switches and one configuration strategy, that is given to all hosts, pertaining to VMs. Additionally, there may be host specific strategies for network QoS, which are the result of the refinement procedure. In this example, with the QoS path introduced in Listing 5, there is be a QoS strategy for each host.

The choice for having one VM strategy valid on all hosts requires that placement information is stored as part of the strategy. This extends the definition of a configuration strategy from the architecture specification. However, during development and initial experimentations, migrating VMs was a very frequent use case, warranting automation. Apparently this is implemented very efficiently with global knowledge on VM placement available.

Listing 6 shows the status messages of the automaton as it generates target configurations. It starts out reading the current state, i.e. monitoring data and the physical infrastructure JSON object; *MNM Xen Lab* is the `name` meta information of the physical infrastructure specification. The host name beginning with `xensrv` indicates monitoring data is attributed to the hosts that generated it.

Having initialised its state, it reads the VI specifications and augments them with placement information to generate a complete target configuration. Lines 10 and 11 of Listing 6 shows the meta names of the VIs to be included

in the target configuration. Lines 12 and 20 mark the beginning of the individual processing of the VI specifications. The printed *placement stack* in lines 13, 15, 17, 21, 23 and 25 shows the remaining workload while handling individual VIs. Following each *placement stack* line, there is a placement line, showing the intention of the management system to place a VM on a host. This is where the management system decides whether a VM must be created and activated, or it has already been placed and is active.

Listing 6: Progress of the translation implementation

```
1  Reading current state ...
2      MNM Xen Lab
3      xensrv01.virt.lab.nm.ifi.lmu.de
4      xensrv02.virt.lab.nm.ifi.lmu.de
5      xensrv03.virt.lab.nm.ifi.lmu.de
6      Got current state!
7  Reading configuration files ...
8      Read configurations!
9  Determining target configuration ...
10     Einfaches Beispiel
11     Zwei Kisten ein Router
12     Einfaches Beispiel
13         placement stack: [u'tva', u'tre', u'ett']
14         ett             -> xensrv02
15         placement stack: [u'tva', u'tre']
16         tre             -> xensrv02
17         placement stack: [u'tva']
18         tva             -> xensrv02
19         [empty stack]
20     Zwei Kisten ein Router
21         placement stack: [u'mVend03', u'mVend01', u'mR00']
22         mR00             -> xensrv02
23         placement stack: [u'mVend03', u'mVend01']
24         mVend01             -> xensrv01
25         placement stack: [u'mVend03']
26         mVend03             -> xensrv03
27         [empty stack]
28     Done placing nodes on hosts!
29     Identified virtual switches!
30     Target configuration complete!
31 Grinding Salt states ..
```

If a VM is to be created or activated, a list of hosts is refined using all applicable constraints. From the remaining list all hosts are, by definition, equally suited to run the VM and one host is randomly picked. If the VM is already placed and no constraints are violated by this placement, the new placement will be the old placement. This prototype has limited capabilities to migrate VMs in order to meet constraints, but this is merely for practical purposes and not core concern of this implementation.

The example output shows all three VMs belonging to the VI *Einfaches Beispiel* are placed on `xensrv02`, together with mR00 from the *Zwei Kisten ein Router* VI. The placements of mVend01 and mVend03 on `xensrv01` and `xensrv03`, respectively, are the consequence of the *disjointplinks* and *notonthesamehost* constraints, shown in Listing 4.

The last line of output generated by the automaton, line 31 in Listing 6, means it is generating configuration strategies for the VE. The result is shown in Listings 7, 8 and 9. The VM strategy, Listing 10, consists of one block for each VM, beginning with its name. Every VM shall be *running* on the provided *run_host* and be connected to the listed virtual switches.

The Listings 7, 8 and 9 show the configuration strategies for virtual switches for `xensrv01`, `xensrv02` and `xensrv03`, respectively. The *xenbr* blocks describe virtual switches incorporating a single physical NIC, and therefore an uplink to the physical OSI layer 2 topology. On the other hand, the *vswitch* blocks describe isolated virtual switches to which VMs may be connected. To reach nodes and networks beyond the physical host, a *vswitch* instance must have an *uplink* to a *xenbr* instance. Every *vswitch* instance has a *require_in* attribute, which is a list of all the VMs it will link. The *require_in* attribute belongs to Salt's functionality and ensures that the virtual switches are configured before any VM the are required in. Across all listings, *vswitch* names are unique. This is a design decision, making the results more human readable and reverse mappings from virtual switch to VI easier, but it is not necessary.

The QoS strategies are Listings 11, 12 and 13. In the herein used example, the QoS path required by the constraint introduced in Listing 5 involves all

three hosts. This prototype can realise path segments end-to-end, or end-to-uplink. As mVend01 and mVend03 are on separate hosts, with mR00 in between. The result are four end-to-uplink path segments, where xensrv01 and xensrv03 implement one each and xensrv02 implements two path segments, one for each uplink to the physical network. The concrete steps of the refinement procedure resulting in this strategy are discussed in the upcoming Section 7.2.4.

Listing 7: xensrv01, Switch strategy

```
xenbr0:
   vswitch.present:
      - interfaces:
         - eth0
vswitch2:
   vswitch.present:
      - uplink: xenbr0
      - require:
         - vswitch: xenbr0
      - require_in:
         - vm: mVend01
```

Listing 8: xensrv02, Switch strategy

```
xenbr0:
   vswitch.present:
      - interfaces:
         - eth0
xenbr2:
   vswitch.present:
      - interfaces:
         - eth2
vswitch1:
   vswitch.present:
      - require_in:
         - vm: tre
         - vm: tva
         - vm: ett
vswitch4:
   vswitch.present:
      - uplink: xenbr0
      - require:
         - vswitch: xenbr0
      - require_in:
         - vm: mR00
vswitch5:
   vswitch.present:
      - uplink: xenbr2
      - require:
         - vswitch: xenbr2
      - require_in:
         - vm: mR00
```

Listing 9: xensrv03, Switch strategy

```
xenbr2:
   vswitch.present:
      - interfaces:
         - eth2
vswitch3:
   vswitch.present:
      - uplink: xenbr2
      - require:
         - vswitch: xenbr2
      - require_in:
         - vm: mVend03
```

Listing 10: VM strategy

```
mVend03:
   vm.running:
      - run_host: xensrv01
      - networks:
         - vswitch3
mVend01:
   vm.running:
      - run_host: xensrv03
      - networks:
         - vswitch2
ett:
   vm.running:
      - run_host: xensrv02
      - networks:
         - vswitch1
mR00:
   vm.running:
      - run_host: xensrv03
      - networks:
         - vswitch2
         - vswitch4
tre:
   vm.running:
      - run_host: xensrv02
      - networks:
         - vswitch1
tva:
   vm.running:
      - run_host: xensrv02
      - networks:
         - vswitch1
```

Listing 11: `xensrv01`, QoS strategy

```
datarate:
   qos.enforcepath:
      - type: datarate
      - units: 201
      - segment:
         - mVend01
         - xenbr0
      - require:
         - vm: mVend01
         - vswitch: vswitch2
         - vswitch: xenbr0
```

Listing 12: `xensrv02`, QoS strategy

```
datarate00:
   qos.enforcepath:
      - type: datarate
      - units: 201
      - segment:
         - mR00
         - xenbr0
      - require:
         - vm: mR00
         - vswitch: vswitch4
         - vswitch: xenbr0
datarate01:
   qos.enforcepath:
      - type: datarate
      - units: 201
      - segment:
         - mR00
         - xenbr2
      - require:
         - vm: mR00
         - vswitch: vswitch5
         - vswitch: xenbr2
```

Listing 13: `xensrv03`, QoS strategy

```
datarate:
   qos.enforcepath:
      - type: datarate
      - units: 201
      - segment:
         - mVend03
         - xenbr2
      - require:
         - vm: mVend03
         - vswitch: vswitch3
         - vswitch: xenbr2
```

7.2.3.3. Configuration

Component configuration is done using Salt, which is a management tool specifically created for configuration management. Salt realises a Manager-Agent architecture. Consequently, the depicted management agents with controllers in Figure 7.2 are Salt agents, with the extensions introduced in the following. Salt already has the capability to manage Xen hosts, but not with the API used in this experimentation environment. Consequently this prototype implements three Salt *modules*:

- one for managing VMs on the Xen hypervisor,
- one for managing virtual switches using OVS, and
- one for enforcing network QoS.

The modules for Xen and OVS can interpret VM and switch configuration strategies, respectively, and configure hosts accordingly. The correct distribution of configuration strategies, *states* in Salt's terminology, their processing and feedback is all performed by Salt. This is implemented using heuristics on file names. Files whose names end with …

- `all.sls` are distributed to all hosts,
- `01.sls`, `02.sls` and `03.sls` are distributed only to `xensrv01`, `xensrv02` and `xensrv03`, respectively.

The module for enforcing network QoS must perform further processing of configuration strategies, because neither the Xen hypervisor, nor the OVS have a canonical implementation of network QoS. The QoS module is an automaton, implementing the architecture's *QoS Interface*, specialised to OVS switches on a Xen hypervisor capable of enforcing network data rate. It receives demands for a lower bound for data rate on a path, either end-to-end, or end-to-uplink. The demands are translated into Xen and OVS specific configurations. These configurations are resource allocations and resource consumption restrictions, implemented by both, the hypervisor and the OVS, which set aside enough resources so that the overall load on the system cannot become high enough to impede network QoS.

7.2.3.4. Monitoring

A very important task for any implementation with a monitoring loop is repeated constantly is matching the target configuration to the current configuration, because where the target configuration matches the current configuration, no further configuration steps must be performed. Only the differences require performing management on the VE. Consequently this prototype implements three monitoring *packs*:

- one for obtaining the current state of the VE pertaining to VMs,
- another one for obtaining the state of virtual switches, and
- a data rate measuring benchmark.

The benchmark realises the monitoring strategy of QoS paths specified through the *enforcedqospath* constraint. The data rate benchmark creates only one record every 48 hours, or when triggered manually.

As indicated in the previous section when showing the data format, monitoring is done per component. For each monitoring task, there is one record for each monitored component. For this prototype, this results in three records about VMs and another three records on virtual switches every 15 minutes. When there are VMs and virtual switches on a host, the record will include their complete descriptions as task specific data. This constitutes the current configuration.

The current configuration is stored completely once and the short status messages a second time in a separate file. The separate file with the short status messages contains exactly one entry for each pair of host and monitoring task. Only if a management task returns a different short status message than the one stored in the file, the file is updated. The control implementation uses the modification time of this file to notice if the monitoring has noticed a status change important enough to generate and implement a new target configuration.

7.2.4. Experimentation

This section performs experiments using the introduced prototype and infrastructures to showcase the proposed management approach can be used to perform network QoS management in VEs. The experiment itself is performing the data rate benchmark, the monitoring strategy associated with the *enforcedqospath* QoS path. The results have been published previously in [Metz 14b].

The experiment is performed five times under varying conditions:

- **T1** The monitored VI is the only active VI and the QoS strategies are not implemented.
- **T2** There is an additional VI, creating massive CPU load on `xensrv02` and the QoS strategies are not implemented.
- **T3** There is an additional VI, creating massive network load and the QoS strategies are not implemented.
- **T4** There is massive CPU load on `xensrv02` and the QoS strategies are implemented.
- **T5** There is massive network load and the QoS strategies are implemented.

To create a data flow as the benchmark, the program `mbuffer` is used, to copy a ≈ 1 GByte large ISO image between the endpoints. By using `mbuffer` with in RAM buffers larger than the actual file, potential side effects due to disc I/O are eliminated. Also, `mbuffer` shows the achieved data rate over the last second, which is sufficiently accurate, as there is not anything else competing for resources on the endpoints.

To avoid further interference with the measurement, the entire benchmark becomes one monitoring record, with an excessively large amount of task specific data. Listing 14 shows the first and last three lines of the task specific data, measured for experiment **T1**. Each line shows the average throughput, in and out, during the last second, the total amount of data transferred so far and the buffer fill level.

Figure 7.3 illustrates the implementation of the main VI for the experiments on the physical infrastructure, both introduced with in Section 7.2. The six

Listing 14: Excerpt of the task specific data from experiment T1

```
out @ 38.6 MB/s, 31.5 MB total, buffer  95% full
out @ 38.9 MB/s, 70.4 MB total, buffer  92% full
out @ 39.1 MB/s,  109 MB total, buffer  89% full
          [ ... 24 lines skipped ... ]
out @ 39.2 MB/s, 1087 MB total, buffer   7% full
out @ 39.0 MB/s, 1127 MB total, buffer   4% full
out @ 38.8 MB/s, 1165 MB total, buffer   1% full
```

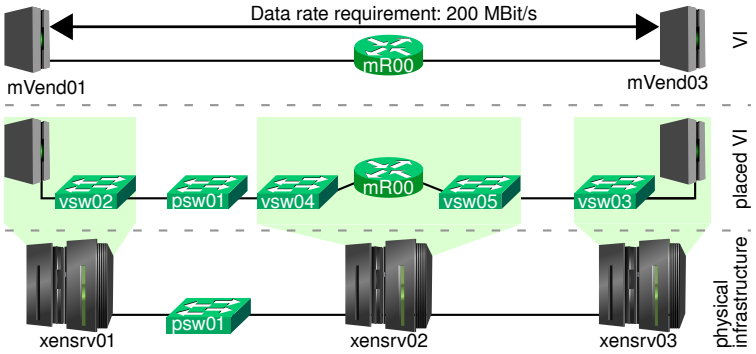


Figure 7.3.: The main VI implemented on top of the physical infrastructure

lines shown in Listing 14 are representative for the entire experiment T1. Using `mbuffer` to transfer data from mVend01 to mVend03 yields a relatively steady data rate of 39 MByte/s .

For the other experiments, the VMs ett, tva and tre are used to create load on `xensrv02`, in order to create QoS degradation, i.e. a lower average data rate when performing the benchmark. The hosts, especially `xensrv02`, have one CPU with two physical cores and can process four instruction streams simultaneously. The idea was to end up with one more object to schedule than simultaneously executable instruction streams. This would be achieved with three VMs as load generators, mR00 for data forwarding

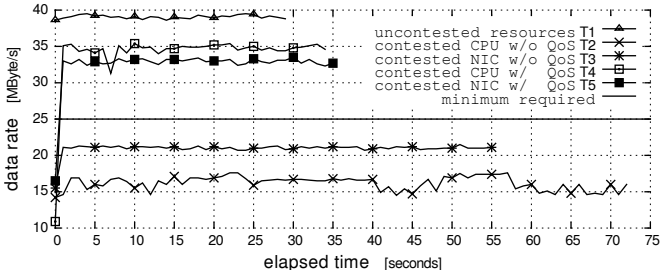


Figure 7.4.: Achieved data rates

and the privileged VM, dom0 as fifth scheduling object for the hypervisor.

As it turns out, for inflicting QoS degradation, it does not matter how load is created. Having three load generator VMs is as effective as having one load generator VM with three virtual CPUs. Similarly, one VM can saturate a 1 GBit/s link as effectively as three VMs generating network load can. When creating sufficient CPU load so that QoS degradation occurs, additional network load hardly leads to further degradation. These effects can be specific for this particular set up and need not hold for all VEs. However, these observations greatly reduces the number of experiments required to create a proper impression of this specific VE.

The results from the performed experiments are summarised in Figure 7.4. Under heavy load the achieved data rates in T2 and T3 drop constantly below the lower bound of 200 MBit/s. Notice, that QoS degradation due to CPU load yields a very constant data rate, while network load yields a lot less steady measurements. The achieved data rates in T4 and T5, where the QoS strategies are implemented, are above the lower bound and therefore fulfil the QoS requirements.

The same method for creating load in experiments T2, T4 and T3, T5 were used. To generate CPU load, the tool `stress` was used. This tool is specifically written for the task of consuming resources, especially CPU

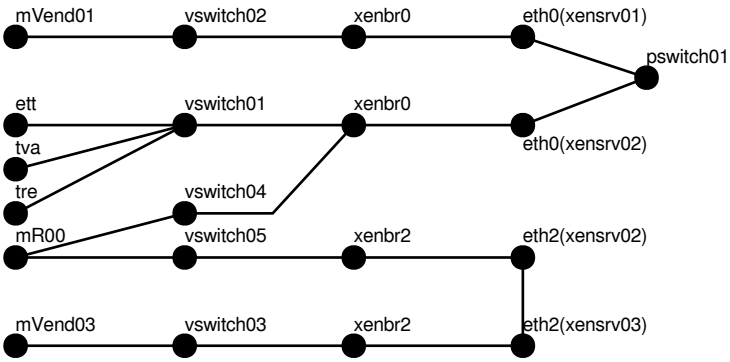


Figure 7.5.: All network links used to realise the target configuration

resources. Network load was created using `wget` to download a multi GB ISO image, but redirecting the download stream, so that the downloaded data would never be stored, thus avoiding unwanted load on `xensrv02`.

The management system achieves network QoS by devising and implementing the QoS strategies introduced in Section 7.2.3. The strategies are created after all placement is done, to limit the problem space. Starting point is the *enforcedqospath* constraint from Listing 5, specifying a lower bound of 200 MBit/s and the endpoints mVend01 and mVend03.

With the placement information available, a graph as depicted in Figure 7.5 can be constructed. This graph includes every network link required to build the whole target configuration. Within this graph, the end-to-end data path for the QoS path can be determined. The end-to-end data path is illustrated as thick lines in Figure 7.5, which is the same path as the depicted *placed VI* in Figure 7.3. This path is the knowledge required for a meaningful implementation the procedures `pathAtOsiLayer()` and `neighborOf()` used in Algorithm 1 on page 107. These were the last missing pieces to allow a canonical implementation of a refinement procedure as specified by Algorithm 1.

The adaptQoS() procedure, which has to be QoS Category specific, is an adaptation of the requested data rate to the protocol overhead introduced through IP and Ethernet PCI. Summing it all up, the steps of the refinement procedure to create the QoS strategies introduced earlier are:

1. Starting out with only the end-to-end path as depicted at the top of Figure 7.3: four iterations of pathAtOsiLayer(), moving from OSI l7 to l3, where mR00 is identified, because it has been specified as a layer 3 component in the VI specification.
2. adaptQoS → 200.3 MBit/s to account for IP PCI.
3. Twice: recursion into subpath
   a) Five iterations of pathAtOsiLayer(), moving OSI l7 to l2.
   b) adaptQoS → 200.6 MBit/s to account for Ethernet PCI.
   c) link for all found segments, building the longest possible subpaths that are implemented by an individual host.

Step 3c) yields the path segments as put in the QoS strategies shown in Listings 11, 12 and 13. The 201 units result from rounding 200.6 MBit/s.

7.3. Summary

This chapter assesses the suggested architecture, by first validating against the developed requirements and showcasing a prototype for network QoS management in VEs.

Section 7 discusses how the individual requirements can be fulfilled by a management system, following this architecture. Section 7.2 proceeds with developing a prototypical implementation, aimed at illustrating the automation aspects of the suggested management loop, before Section 7.2.4 performs experiments on a VE with the developed prototype.

The experiment shows, that, with the herein developed approach, implementation independent abstract VI specifications hold enough information to realise effective network QoS management in VEs.

Chapter 8

Summary and Outlook

Network QoS relies on guarantees, that all involved components perform in a certain manner. Virtualization environments feature constant change and uncertainty about a components current implementation. This opposes network QoS, because it makes guaranteeing a components performance much more difficult. Consequently, network QoS management must be adapted to overcome challenges, that are specific for VEs, in order to achieve network QoS analogous to strictly physical environments.

While the management of virtual network endpoints is widely understood, management of networks and network components in VEs used to take a subsidiary role. As a result, available management approaches for VEs are not fully adopted to VE specific challenges. This work contributes a management architecture that structures the network QoS management task, such that the previously impassable problems can be controlled and network QoS in VEs implemented.

VEs are versatile platforms, hence network QoS management is one aspect amongst many management concerns. To develop a sustainable approach, this work determines a prototypical life cycle for VIs and arranges the network QoS management task with it. The intention is to enable this approach being used in combination with solutions for other management tasks, such as embedding.

After a closer analysis of use cases in real world scenarios and existing work, automation appears as the only sustainable approach to dealing with virtualization dynamics. Especially as data centres grow to span the area of multiple football fields, the sheer numbers of MOs becomes overwhelming for human managers. There must be automated assistance for performing management tasks.

The stacking of technologies and abstractions in VEs is problematic for network QoS, which often employs resource sharing mechanisms to enforce a certain behaviour of components. To enable similar functionality in VEs, the architecture models abstract network QoS from any implementation, while remaining specific enough for deterministic implementations.

A side effect of the implementation independent network QoS specifications is, that the most important aspects of a virtual component are its placement and implementation. This is one step ahead of today's management systems, where concrete properties like CPU, RAM and I/O channels usually are a VM's determinant.

The defining aspect of the herein developed management architecture is the underlying idea of a management feedback loop, constantly shaping the VE to implement and enforce network QoS. The management loop and the developed arrangement of subtasks around it enable fulfilment of all but one identified requirement.

Understanding the challenges of VEs, a management architecture according to ISO OSI has been specified and validated. A prototypical implementation of this management loop based approach has been provided. Experimental evaluations show that management systems following the proposed architecture are suited for efficient network QoS management in virtualization environments.

Outlook

The previous chapters often show how much preparation and preliminary work is required, before there is even enough information available, to ac-

tually consider network QoS. For example:

- A lot of use cases are identified in the real world scenarios, and while many are relevant to network QoS management, it is hardly an immediate concern of, or even motivation for, any use case.
- In the information model, QoS is merely one out of four domains, not larger than any other.
- When developing the prototype, quite the effort is made to describe and monitor VIs and VEs, before network QoS can be introduced as a single constraint with two simple attributes.

To deliver a sustainable architecture, the approach has to start out as an attempt at management in VEs in general and then immediately focus on network QoS. This arranges network QoS management properly within VEs, while avoiding picking up on problems outside the scope of network QoS management. As an effect, many new questions arise adjacent to the answered questions and already become visible in the proposed architecture.

For the information model, the object hierarchy and relations to classes outside the presentation domain are what is important for network QoS management. This made it acceptable to not require and specify attributes for the therein contained classes. This immediately raises the question for an accurate modelling of a virtual component, when aiming for a more holistic management approach for VEs.

The organisation model identifies discernible roles with varying management tasks. From a network QoS management point of view, these roles could be realised through multiple independent providers, but can these roles really fulfil their tasks when realised that far disjoint? Will there be additional challenges pertaining to sharing information and resources, when network QoS management of a single VI translates into configuration strategies implemented by multiple VEs?

The function model very prominently features a generic interface for all management concerns that are not network QoS management. Is this sole

position in the management loop sufficient for all management aspects that can sensibly be integrated with network QoS management?

The architecture presented herein allows the management of network paths, independent from their implementation. This becomes ever more useful, with the complexity of such paths' implementations. With the developed models and management capabilities, the question arises: which are the most effective and efficient constructs for implementing network paths? It is possible, that the answer to that question varies with the QoS requirements associated with a path. Today, a virtual machine could perform traffic engineering to optimally use the underlying physical resources. Yet, when technologies such as active queue management are used, the effect on QoS properties such as delay and drop rate could be disastrous. There could be benefit in determining recipes and building blocks for network topologies with QoS properties in VEs.

There is a constant development of new virtualization technologies. Recent developments are called "virtual network functions" (VNF) and "software defined networking" (SDN). VNF basically allows for quick development and deployment of new specialised types of virtual network components, while SDN realises new approaches to centralise and implement network configurations.

Both developments, VNF and SDN, have the potential to develop additional sources for dynamics and uncertainty about network component implementations. This might warrant additional types of strategies and functional components to ensure the management loop can still fulfil its purpose.

As network QoS becomes ever more important, these technology fields will as well. While developing the experiments presented in Section 7.2.4, another small test was the same benchmark, using a topology without the virtual router. With that set up, the virtual machines would achieve average data rates around 80 MByte/s, roughly double the throughput compared to the topology that includes the virtual router, even without resource shortage. This can be seen as an indicator that virtual network components are not yet very efficient, impeding large scale applications, while further motivating the development of new, more efficient, technologies.

Emerging technologies such as VNF facilitate ever more efficient virtual network components. On the one hand, the advent of these new components will create further heterogeneity, in the sense that these components must be managed in a different manner. On the other hand, these technologies allow for new services, or at least new service levels, where network QoS provides added value. This is when the concepts and mechanisms introduced in this work will become integral parts of management systems.

Appendix A

Use cases specifications

The following use cases' format:

`name`: Each use case is labelled and named for structuring purposes and later reference.

`description`: An informal description summarising intention and effect of the use case.

`precondition`: Dependencies and prerequisites so that the use case can successfully be performed.

`postcondition`: A description of the changes to the VE after the use case has been performed successfully.

`procedure`: A prototypical description of the steps transitioning the VE state from pre- to postconditions. For VMs and VNs this often means the execution of certain management operations, described in Section 5.2.1 and Section 5.2.2, respectively.

`effects`: Potential for side effects on network QoS, used in Section 3.4 to derive requirements on network QoS management.

`functional requirements`: Requirements on functional abilities of management systems, used in Section 5.3 to derive prerequisites and behavioural requirements.

`nonfunctional requirements`: Requirements on other aspects than functional abilities, used in Section 5.3 to derive prerequisites and behavioural requirements.

UC 1 **Assign resources to components**

DESCRIPTION A manager assigns resources to components. There may be different kinds of resource, for example CPU time, RAM and storage space. Resource assignment implicitly guarantees a minimum capacity for performing work, with respect to the kind of resources allocated.

PRECONDITION The VI has been assigned a pool of bulk resources and this assignment of resources to components does not overtax the VI's resource pool. The management system provides the manager with information on assigned, used and free resources.

POSTCONDITION The realising components of the VE enforce that the component has hat least the assigned resource available.

PROCEDURE

1. A component is assigned resources from a VIs resource pool within the management system.
2. The VE, i.e. all realising subsystems of the current component, are reconfigured to assure the assigned amount of resources. This may require a replacement of the component and its links within the VE.
3. The component is rearranged with its environment and its links are renewed, if necessary.

POTENTIAL FOR SIDE EFFECTS ON NETWORK QOS

- Resources allocated to VMs cannot be used by VN components.
- Network components cannot be allocated resources from resource pools.

FUNCTIONAL REQUIREMENTS

- Control over resource allocations in virtual and physical components
- Migration of virtual components

Non-functional requirements

- The currently used resources by components must be obtainable
- Links between virtual components are automatically adapted by the management system
- Full information about the current placement of components within the VE

UC 2 **Activate component**

Description A previously deactivated component is activated to actively participate in a VI.

Precondition The component has been created and deactivated. The VE has sufficient capacities available to operate the component within its specifications.

Postcondition The component can be used to provide user faced services and is part of a VI.

Procedure

1. The component is placed within the VE.
2. The component is activated, thus blocks and consumes resources.
3. The links to other components are placed.
4. The component becomes operational.

Potential for side effects on network QoS

- When a component is activated it binds and blocks resources on a host.
- Component placement defines how the VNs spans across the physical topology.

Functional requirements

- Control over virtual and physical components

Non-functional requirements

- The currently used resources by components must be obtainable
- Links between virtual components are automatically adopted
- Full information about the current placement of components within the VI

UC 3 **Deactivate component**

DESCRIPTION An active component is deactivated, but not removed from the VE.

PRECONDITION The component has been created and is active.

POSTCONDITION The component cannot be used to provide user faced services, does not consume any physical resources, but may still be assigned resources from a VI's resource pool.

PROCEDURE

1. Links terminated by this component are severed.
2. The component is deactivated.
3. Physical resources reserved for the component are freed.

POTENTIAL FOR SIDE EFFECTS ON NETWORK QOS

- Resources on the host can be released and in turn allocated to network components.

FUNCTIONAL REQUIREMENTS

- Control over virtual components
- Explicit release of allocated resources

NON-FUNCTIONAL REQUIREMENTS

- Not applicable to this use case

UC 4 **Create component**

DESCRIPTION A new component is created, placed, activated and linked with other components.

PRECONDITION A VI has been created successfully.

POSTCONDITION The new component is active and integrated into the virtual topology.

PROCEDURE

1. The VM is created.
2. The VM is linked with other components.
3. The VM is placed and activated.
4. Monitoring is set up for the virtual links.

Potential for side effects on network QoS

- When a component is activated it binds and blocks resources on a host.
- Component placement defines how the VNs spans across the physical topology.

A new component is created, placed, activated and linked with other components.

Functional requirements

- Control over the VMM
- Monitoring structures for VN properties can be set up

Non-functional requirements

- The currently used resources by components must be obtainable
- Links between virtual components are automatically adopted
- Full information about the current placement of components within the VI

UC 5 **Migrate component**

Description Migrate a virtual component between physical hosts.

Precondition The component is active and the target physical host has the capacities to host the virtual component.

Postcondition The migrated component works on its new host as it did before the migration and all links have been adapted so that every link still operates within all specified parameters.

Procedure

1. Migration of a virtual component is triggered.
2. All links of the component are renewed.
3. Monitoring for affected QoS network segments is adopted.

Potential for side effects on network QoS

- There is a time interval where one component blocks resources on two hosts.
- The components network uplink on the origin host are not required after the migration completes.

- A network uplink is provisioned on the target host.

Functional requirements

- Control over the VMM
- Monitoring structures for VN properties can be set up

Non-functional requirements

- The currently used resources by components must be obtainable
- Links between virtual components are automatically adopted
- Full information about the current placement of components within the VI

UC 6 **Delete component**

Description A virtual component is withdrawn from the VI.
Precondition The component has been created successfully.
Postcondition The component is completely removed and all links have been severed.

Procedure

1. The component is deactivated.
2. All links to and from the component are severed.
3. Resource allocations are undone and the component is removed from the VE.

Potential for side effects on network QoS

- Resources on the host can be released and in turn allocated to network components.

Functional requirements

- Control over virtual and physical components

Non-functional requirements

- Full information about the current placement of components within the VI

UC 7 **Create VI**

DESCRIPTION The virtualization provider creates a VI within the VE. VNs and VMs are created and linked and the ready to use VI is then handed over to the customer.

PRECONDITION Customer and provider have agreed upon a complete specification of the intended VI. The VI is ready to provide virtual components and links, and a management user for the customer has been created.

POSTCONDITION All components of the VI are active and can communicate within the specified parameters. Customer and provider can perform management on the VI.

PROCEDURE

1. The VN is created.
2. The VMs are created and linked according to the specification.
3. The virtual components are activated and placed within the VI.
4. Monitoring is set up to gather data on the newly activated VI.
5. The customer's management user is associated with the VI.

FUNCTIONAL REQUIREMENTS

- Control over virtual and physical components (to create and link components)
- Monitoring structures for VN properties can be set up

NON-FUNCTIONAL REQUIREMENTS

- Full information about the current placement of components within the VI (to activate and place VI components)
- Links between virtual components are automatically adapted by the management system (to match the virtual components' placement in the VE)

UC 8 **Modify VI**

DESCRIPTION Change meta information pertaining to an individual VI, but may affect many components. For instance, generic QoS requirements for VNs, or requirements on component placement may change over time. This use cases foremost influence how the management system handles the VI. If this happens while a VI is active, active components may be affected and consequently, their configuration and monitoring must be adapted. Note, this use case targets meta information, rules and policies, for VIs only. Changes with respect to components and the topologies they constitute are separate use cases, directly associated with virtual components and links.

PRECONDITION The VI has been created successfully.

POSTCONDITION The VI is in the same state as before and within the new parameters.

PROCEDURE

1. The specifications pertaining to the VI are changed.
2. All affected components and links are modified accordingly. Virtual components are rearranged with their environment for resource demands and their links renewed. This may include migrating some components, thus affecting more components and links.
3. The deployed monitoring is adapted to the new component and link placement.

FUNCTIONAL REQUIREMENTS

- Explicitly migrate individual components (to fit the VI to the new parameters)
- Monitoring structures for VN properties can be set up

NON-FUNCTIONAL REQUIREMENTS

- Full information about the current placement of components within the VI (to determine all affected components)
- Links between virtual components are automatically adapted by the management system (to match the virtual components' placement in the VE)

UC 9 **Delete VI**

DESCRIPTION When a VI is no longer needed, it is removed from the VE, including all its components and links.

PRECONDITION A VI has been created successfully. The user faced services are not employed any longer and the customer has requested that the VI is deleted.

POSTCONDITION All links to and between components of this VI have been severed, VNs deleted and all VMs, that belonged only to this VI, have also been deleted. There are not any associations from managing users to the VI and all previously set up monitoring has been removed.

PROCEDURE

1. All VMs belonging to the VI are halted.
2. All management users are disassociated from the VI.
3. All links to and between the halted VMs are severed.
4. All components are removed from the VNs and the VNs are deleted.
5. The VI is removed.

FUNCTIONAL REQUIREMENTS

- Control over virtual and physical components (to delete components and sever links)

NON-FUNCTIONAL REQUIREMENTS

- Full information about the current placement of components within the VI (to determine all affected components)

UC 10 **Assign resources**

DESCRIPTION Quantitative shares of physical resources are attributed to the VI. These amounts are the total available resources that can be assigned to VI components by primary and secondary customers.

PRECONDITION A VI has been created successfully. The amount of resources to be assigned is greater than the amount currently used by the VI components.

POSTCONDITION Primary customers can attribute resources to components or delegate resources to secondary customers.

PROCEDURE

1. A set of physical resource shares is attributed to the VI.

FUNCTIONAL REQUIREMENTS

- Not applicable to this use case, as the effect is limited to the management system and not directly filtered down to the VE.

NON-FUNCTIONAL REQUIREMENTS

- The currently used resources by components must be obtainable (to verify the precondition)

UC 11 **Create link**

DESCRIPTION A user instructs the management system to create the possibility for two components of the same VI to interact with each other over a network.

PRECONDITION Both components have been created successfully and are part of the same VI.

POSTCONDITION Data traffic between the components adheres to the specifications provided when creating the link.

PROCEDURE

1. The endpoints on the new link are selected.
2. Specifications describing the link and data transmission properties are made.
3. The link is placed within the VE.
4. Monitoring is set up.

FUNCTIONAL REQUIREMENTS

- Control over virtual and physical components
- Monitoring structures for VN properties can be set up

NON-FUNCTIONAL REQUIREMENTS

- QoS requirements are specified w.r.t. the selected components or VNs.

UC 12 **Delete link**

DESCRIPTION When a link is removed from a VI its specific requirements are removed from all ongoing QoS considerations and corresponding monitoring set ups are withdrawn. If creating the link has any spawned corresponding links and paths in the VE, they are removed as well.

PRECONDITION The link as been created successfully.

POSTCONDITION The specifications and requirements that were associated with this link do not influence any QoS implementation and monitoring any more.

PROCEDURE

1. The affected components and path segments are identified and reconfigured.
2. Components and paths implementing only this link are deleted.
3. All links that shared a network path segment or a component with the deleted link are renewed.
4. The link is removed from the management system.

FUNCTIONAL REQUIREMENTS

- Control over virtual and physical components
- Control over the VMM
- Control over resource allocations in virtual and physical components
- Explicit release of allocated resources

NON-FUNCTIONAL REQUIREMENTS

- Links between virtual components are automatically adapted by the management system
- Full information about the current placement of components within the VI

UC 13 **Modify link attributes**

DESCRIPTION Applying changes to requirements on the link's properties and specification. This results in renewing and potentially replacing the link in the VE.

PRECONDITION The VE can implement the link with the changed attributes.
POSTCONDITION The link can be used within the new specifications.
PROCEDURE
1. Properties and settings are changed within the management system
2. The link is renewed and newly placed if necessary

FUNCTIONAL REQUIREMENTS
- Control over virtual and physical components
- Control over resource allocations in virtual and physical components

NON-FUNCTIONAL REQUIREMENTS
- Links between virtual components are automatically adapted by the management system

UC 14 **Place link**

DESCRIPTION Links spanning across multiple components may have multiple conceivable implementations, which are all capable of fulfilling QoS requirements. With this use case, the virtualization provider aims to optimize resource usage by placing links within the VE using global metrics, beyond the scope of the individual link.
PRECONDITION The link endpoints are active and already placed.
POSTCONDITION The link can be used within the specified parameters.
PROCEDURE
1. Trigger management system to place the link

FUNCTIONAL REQUIREMENTS
- Control over virtual and physical components
- Control over resource allocations in virtual and physical components

NON-FUNCTIONAL REQUIREMENTS
- Links between virtual components are automatically adapted by the management system

UC 15 **Evacuate host**

DESCRIPTION The manager intents to have the host not realising any virtual links or components. All components currently placed on the host are migrated to other hosts, possibly requiring new placement of entire VIs.

PRECONDITION The host is active and implementing virtual components as part of the VE. There are other hosts that can host affected virtual components.

POSTCONDITION There are neither virtual components nor virtual links placed on the host. All previously hosted components have been migrated to other hosts and the affected VIs still operate within their specified parameters.

PROCEDURE

1. The host is stopped from accepting additional virtual components and links to be realised.
2. All components placed on the host are migrated to different hosts.
3. Links to the migrated components are renewed and possibly newly placed.

FUNCTIONAL REQUIREMENTS

- Control over the VMM
- Migration of virtual components

NON-FUNCTIONAL REQUIREMENTS

- Links between virtual components are automatically adapted by the management system
- Information on components belonging to VIs

UC 16 **Deactivate host**

DESCRIPTION A virtualization host is evacuated and then suspended. This use case is similar to UC15 with an additional step of suspending the host.

ADDITIONAL FUNCTIONAL REQUIREMENTS

- Control over physical host

UC 17 **Remove host from virtualization environment**

DESCRIPTION A virtualization host is evacuated, then suspended and completely removed from the VE so that it cannot be reactivated. From a network QoS point of view this use case is effectively similar to UC16, with the exception that the host cannot be activated at a later point in time.

UC 18 **Add host to virtualization environment**

DESCRIPTION A new host is connected to the VE and integrated as another virtualization host, that is capable of providing VIs and VI components. This increases the VEs available resources for providing VIs.

PRECONDITION The host has never been part of the VE.

POSTCONDITION The host can provide virtual links and components that are part of VIs.

PROCEDURE

1. The host is physically connected to the VE.
2. The host is added as additional resources that can be used for and allocated to VIs.

FUNCTIONAL REQUIREMENTS

- Control over virtual and physical components

NON-FUNCTIONAL REQUIREMENTS

- The management system is aware of all available and deployed resources.

UC 19 **Activate host**

DESCRIPTION A previously deactivated host is activated to host virtual components.

PRECONDITION The host has been added to the VE and is currently deactivated.

POSTCONDITION The host can provide virtual links and components that are part of VIs.

PROCEDURE

1. The host is brought up so that it is fully operational and ready to provide links and components.
2. The host is added as additional resources that can be used for and allocated to VIs.

FUNCTIONAL REQUIREMENTS

- Control over virtual and physical components

NON-FUNCTIONAL REQUIREMENTS

- The management system is aware of all available and deployed resources.

Appendix B

Requirements specifications

To capture all important aspects, the structural template is the requirements shell as specified in the Volere requirements specification template [RoRo 06]. The requirements shell is originally designed to capture all important aspects of a requirement and trace its origin, especially for projects where multiple interest groups contribute to the requirements engineering over time. As the latter aspect is not required, the following changes are applied to the requirements shell to suit this work:

- A short descriptive title is added.
- The priority is used as description of its contribution to the management system.
- The conflict field is omitted, because no conflicts were identified.
- The originator is not relevant for the following development and therefore omitted.
- Customer (dis-)satisfaction is omitted.
- No chronological tracking of requirements is performed, hence history is omitted.
- All supporting material is provided or referenced in the preceding scenario analysis and not repeated.

The requirements derived in this work demand functional and non functional aspects of a management system for network QoS management in

VEs. The requirements beginning from 11 are marked as prerequisite or behavioural requirement. Prerequiresites must be fulfilled by sub systems that are not developed as part of this work, while behavioural requirements shape the resulting architecture to suit today's VEs, but are not immediately required to achieve network QoS management.

Req.#1	Resource allocation to virtual machines

TYPE: non-functional PRIORITY: Resource management

DESCRIPTION: VMs must be assigned resources that control their capacities to fulfil their tasks.

RATIONALE: VMs require resources to perform their tasks. To guarantee QoS, VMs must be guaranteed a certain capacity to perform their tasks.

FIT CRITERION: After successful configuration an active VM can maintain its throughput under conditions where the host cannot assign any additional resources to any components.

Req.#2	Resource allocation to VN components

TYPE: non-functional PRIORITY: Resource management

DESCRIPTION: VN components must be assigned resources that control their capacities to fulfil their tasks.

RATIONALE: Similarly to VMs, VN components require shares of physical resources to perform their tasks and their capacities increase and decrease with the available resources. The more a service or program depends on communication to perform well, the more important it becomes that the VN components have enough resources available to fulfil their tasks. Therefore, the capability to assign resources to VN components, in order to guarantee a certain capacity to fulfil their tasks is essential for network QoS Management. Including network components into resource management avoids side

effects where VMs are allocated too many resources, so that network components are starved for resources.

FIT CRITERION: After successful configuration a network components can maintain its throughput under conditions where the host cannot assign any additional resources to any components.

Req.#3	QoS links can be specified with virtual endpoints

TYPE: non-functional PRIORITY: Integration of QoS management

DESCRIPTION: One or many endpoints of a network QoS specification for a link may be virtual components.

RATIONALE: In current virtualization approaches and especially the analysed scenarios, customers are not aware of implementation specifics. Therefore, a specification of a QoS link must be valid at the abstraction level of virtual components. This must provide sufficient management information for the VE to implement an end-to-end network link that satisfies QoS requirements.

FIT CRITERION: After its specification, a link can be placed as path through the virtual and physical network, so that it satisfies the QoS requirements

---

Req.#4	Support for different types of QoS paths

TYPE: non-functional PRIORITY: QoS Management

DESCRIPTION: The management system must support multiple kinds of QoS paths. With the four types of virtual endpoints identified in Section 3.3, together with *Outside* as representative for endpoints outside the VE these are:

(VSe, VSe)	(VSe, VSi)	(VSe, VWS)	(VSe, Outside)
(VSi, VSi)	(VSi, VWS)		
(VWS, RAS)	(VWS, Outside)		
(RAS, Outside)			

RATIONALE: When implementing QoS requirements assumptions may or even must be made on the endpoints and the interconnecting network. Especially when data traffic originates from, or is addressed to, endpoints outside the VE these assumptions may differ. To fully realise network QoS in VEs, a management system must allow network QoS on all conceivable path types.

FIT CRITERION: The management system can control and enforce network QoS on all kinds of QoS paths.

`Req.#5` Virtual Infrastructure semantics for performing management

---

TYPE: non-functional PRIORITY: QoS management

DESCRIPTION: To enable consistent management VIs must be managed as a whole, where each component and each network foremost represent a contribution to meeting the customers demand, rather than a concrete entity to be instantiated.

RATIONALE: The additional indirections through using virtualization to provision DTEs and DCEs allow for many equivalent implementations of the customers' requests. This introduces possibilities for heterogeneous platforms used for the same purpose and the concrete implementation for a VI may change, for example, through migration.

FIT CRITERION: none

Req.#6	Monitoring structures for VIs

TYPE: functional PRIORITY: QoS management

DESCRIPTION: With changes to the VE and the placement of components within, monitoring structures must be set up specifically for a VI's current implementation. Monitoring components, links and the networks they constitute, requires a planned strategy to incorporate all network participants.

RATIONALE: Depending on the placement of components and links, only a subset of physical and virtual components is relevant for a specific VI. In that sense, monitoring for a specific VI must be placed along with the implementation of VI components and links.

FIT CRITERION: For every constructable link monitoring can be set up to gain useful data in the sense of OSI performance management.

Req.#7	Management users are restricted to their associated VIs

TYPE: non-functional PRIORITY: QoS management

RELATED USE CASES: Not explicit as per Section 5.3.1.

DESCRIPTION: A management user may perform management only on their associated VIs. The domains and capabilities must be clearly separated.

RATIONALE: In a multi user environment where VIs are to be operated isolated from each other, the management system must also implement such isolation.

FIT CRITERION: A management user can only obtain information about and perform management operations on their associated VIs.

Req.#8	Multiple concurrent and customer managers

TYPE: non-functional PRIORITY: Resource management

DESCRIPTION: Any manager can always interact with the management system to perform management on their VIs.

RATIONALE: The service *project infrastructures* in Section 3.2.1 foresees at least one customer manager per VI. When performing performance management users are likely to interact more often with the management to retrieve reports and perform measurements. Together with the expansions plans of the real world scenarios frequent access of the management system by multiple users must be expected and the management system must be able to cope with multiple concurrent users, not baring their concrete management role.

FIT CRITERION: Multiple managers can concurrently perform management using the management system without generating side uncontrolled side effects.

Req.#9	Automated adaptation of links

TYPE: non-functional PRIORITY: QoS management

DESCRIPTION: Established network paths are automatically reconfigured to match and account for changes to the VI configuration and changes to VE's state.

RATIONALE: In VEs and especially with multiple customer managers, most management is performed on VIs an their components. The actual placement and used hardware components are only of subsidiary concern to the managers.

Fit Criterion: A network QoS affecting reconfiguration of a VI or physical change to the VE does not require additional management actions by users to adapt all VIs so that network QoS can be guaranteed.

Req.#10	Automated enforcement of network QoS requirements

Type: non-functional Priority: QoS management

Description: The management system automatically reconfigures the VE to enforce QoS requirements for all deployed VIs.

Rationale: To uphold the abstraction from physical hardware and topologies, implementation and enforcement of QoS requirements must be automated and performed implicitly. This is also required to coordinate possible side effects from multiple VIs of different customers with components on the same physical host.

Fit Criterion: VIs cannot influence each other to perform worse than the minimum guaranteed QoS.

Req.#11	Control over physical hosts

Type: prerequisite Priority: Required for comprehensive management of the VE

Related use cases: UC16

Description: Management functions to control physical hosts so they can provide their resources to the VE.

Rationale: Analogous to virtual components, physical components follow a life cycle where they are subject to management. While most management will be performed on the VMM, at the beginning and end of their life cycles, hosts may need to be managed directly.

FIT CRITERION: Physical hosts can be added and removed from VEs correctly.

Req.#12	Control over virtual components

TYPE: prerequisite PRIORITY: Critical for QoS management

RELATED USE CASES: UC7, UC9, UC2, UC3, UC6, UC11, UC12, UC14, UC18, UC19

DESCRIPTION: For each virtual component, all management functions, that are relevant to perform life cycle operations, must be available. This enables the management system to control the life cycle of each component, obtain its current state and trigger other life cycle phases. More functions, pertaining to a component's capabilities to allocate resources to a specific task or even network flow, are required.

RATIONALE: As part of the life cycle operations, virtual components are assigned resources to work with. These are essential to control the QoS a component can provide. The allocation functions are used to configure the individual components and therefore implicitly their immediate links to form network paths which can fulfil QoS requirements.

FIT CRITERION: The management system can configure and monitor all virtual components so that network QoS management can be performed and network QoS enforced.

Req.#13	Control over VMMs

TYPE: prerequisite PRIORITY: Required to control component placement

RELATED USE CASES: UC4, UC5, UC12, UC15

Description: Full management access to the virtualization technology employed on the hosts is required. This includes the VMMs and every management software that may be used to centralise or delegate management information or even tasks, e.g. load balancing.

Rationale: Access to each virtualization implementation is required for monitoring. VMMs often have a centralised management system so that hosts can be managed as clusters. These systems sometimes offer additional information and functionality and may always perform management that influences component placement and resource allocation. Therefore they must be part of network QoS management.

Fit Criterion: All monitoring data is available, component management and especially placement and resource allocation can be performed.

Req.#14	Migration of virtual components

Type: prerequisite Priority: Critical for component placement

Related use cases: UC8, UC1, UC5, UC15

Description: Virtualization technologies must offer the functionality to migrate active virtual components to other hosts.

Rationale: With changes being made to the VIs while they are active, the VE must be able to adapt to the new configurations without interrupting services and a complete replacement of all VIs. Moving individual components is a critical tool to adapt VI placement.

Fit Criterion: Active virtual components can be completely moved to another host, without any remaining dependency on the originating host, within a reasonable timespan.

`Req.#15` Full information about currently used resources by components

---

TYPE: prerequisite PRIORITY: Critical for QoS management

RELATED USE CASES: `UC10`, `UC1`, `UC2`, `UC4`, `UC5`

DESCRIPTION: The currently used resources must be attributable to the consuming (virtual) component.

RATIONALE: This is the essential dynamic information that must be monitored in the VE. This information is required for performance management.

FIT CRITERION: For every active component the used resources can be obtained from the management system.

`Req.#16` Full information about available and deployed resources

TYPE: prerequisite PRIORITY: Critical for QoS management

RELATED USE CASES: `UC18`, `UC19`

DESCRIPTION: The individual providing hosts must be able to determine the amount of resources they can allocate and the amount of deployed resources.

RATIONALE: This is complementary knowledge that combined with other required information allows for performance management, especially for predicting shortages and placement planning.

FIT CRITERION: Information on available and deployed resources can be obtained from each component.

Req.#17	Release allocated resources

TYPE: behaviour PRIORITY: Required for efficient QoS

RELATED USE CASES: UC3, UC12

DESCRIPTION: The core required functionality is the capability to immediately free resources bound by a VI component, so that it can be immediately used by other components.

RATIONALE: When components and links are deactivated or deleted, the management system can not know about the future intentions for the VI. With this explicit functionality, components can become "dormant" and all their resources are taken from them, but may remain part of the VI so they can be reactivated at a later stage.

FIT CRITERION: Reconfiguring a component to have less, or no allocated resources at all takes effect immediately and the resources can be allocated to other components.

Req.#18	All components may be subject to management

TYPE: behaviour PRIORITY: Critical for real world application

RELATED USE CASES: Not explicit as per Section 5.3.1.

DESCRIPTION: Management on all components relevant for network QoS may be performed.

RATIONALE: The management system is to implement various management roles to delegate tasks and responsibilities. To ensure management roles can fulfil their purpose, all components must be manageable from within the management system.

FIT CRITERION: No management user must resort to other means of management to fulfil their assigned roles.

Req.#19	Full information about the current VI placement

TYPE: behaviour PRIORITY: Required for efficient management

RELATED USE CASES: UC7, UC8, UC9, UC1, UC2, UC4, UC5, UC6, UC12

DESCRIPTION: The management system must have current information about the placement of all placed VI components.

RATIONALE: Placed components consume resources and must be linked. Information about placement is crucial for any planning and for reconfiguring placed components and adopting network links and paths.

FIT CRITERION: Correct information about all placed components and the mapping from virtual component to providing hardware can be retrieved from the management system.

Req.#20	Reverse mappings from components to VIs

TYPE: behaviour PRIORITY: Required for efficient management

RELATED USE CASES: UC15

DESCRIPTION: For every individual component it must be known to which VIs it belongs.

RATIONALE: Some components used to realise network segments may be employed by multiple VIs. For management operations that are directed at hardware components, especially not at VIs, the management system must determine which components and which VIs are affected. With that knowledge further action, for instance replacing links and components, can be performed.

FIT CRITERION: For any hardware component all realized virtual components can be determined and their corresponding VIs identified.

Req.#21 Resource allocation within a hierarchy of management users

---

TYPE: behaviour PRIORITY: Required for real world application

RELATED USE CASES: Not explicit as per Section 5.3.1.

DESCRIPTION: In a hierarchy of management users, preceding users can assign resource shares to following users to allocate to components, or their following users.

RATIONALE: In order to perform QoS management on a VI subtopology, a management user requires resources to allocate to components. On the other hand, VIs must not exceed their allocated resource shares.

FIT CRITERION: In a hierarchy of management users resources can be allocated to users, who are then restricted to that amount of resources to allocate to components.

Req.#22 Management users can be tied to life cycles or life cycle phases

---

TYPE: behaviour PRIORITY: Required for real world application

RELATED USE CASES: Not explicit as per Section 5.3.1.

DESCRIPTION: The duration of a management users existence can be bound to the life cycle of a VI, or of a concrete placed instance.

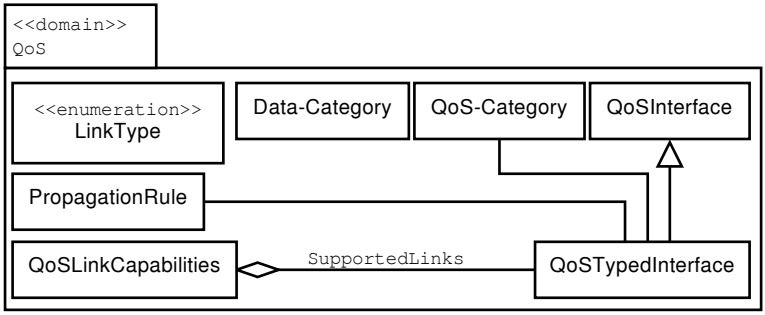
RATIONALE: There are hierarchies of customer managers where tasks and responsibilities may be delegated for a limited period of time. This period is often determined by life cycles, or life cycle phases.

FIT CRITERION: The management users' privileges and responsibilities change with the life cycle phases of VIs and components according to a configuration.

Appendix

Information domain classes

C.1. QoS domain



The type *LinkType* is used to classify a link or path(-segment) or interface according to the taxonomy for links introduced in Section 6.2.1. The following describes the classes of the *QoS* domain in detail:

`Class Data Category`

A *Data Category* describes communication patterns in terms of direction, senders and receivers. This information is attributed to *Association*s from the *presentation* domain to specify how *QoS Category*s must be applied. The classic *Topology* is n:m bidirectional communication, where every node may interact with every other node, while the classic *Path* is a 1:1 bidirectional communication, where two specific endpoints communicating are singled out.

`Class QoS Category`

A *QoS Category* is a set of *QoS Parameter*s specifying the properties that data traffic of this category has, or should have. *QoS Parameters* include semantic descriptions of the individual properties, criteria determining its fulfilment and an assurance type, describing how vigorously the property is enforced.

`Class QoSInterface`

A *QoSInterface* is a generic interface which must be implemented by components, links and paths alike, to realise network QoS. One interface for three different types of managed objects helps the refinement of *Associations* and automation of management. The final mapping of management operations to method calls on involved components is postponed until no further refinement and adaptation is required. The relevant methods for an information model of MOs implementing the *QoSInterface* are:

`GetQoSAvailable()`: Takes an argument of type *QoS Category* and returns a value quantifying current capacity to provide this particular *QoS Category*.

`GetQoSAmount()`: Takes an argument of type *QoS Category* and returns a value quantifying the maximum capacity to provide this particular *QoS Category*.

`GetQoSAllocated()`: Takes an argument of type *QoS Category* and re-

turns a value quantifying the amount of assertions made for this particular *QoS Category*.

`QoSSet()`: Takes an argument of type *QoS Category* as the QoS that must be realised.

`QoSAlloc()`: Takes an argument of type *QoS Category* as a request to facilitate a link with of this *QoS Category* in addition to the existing QoS settings. Returns whether it can fulfil the request, or not.

`QoSFree()`: Takes an argument of type *QoS Category* indicating one less link of this category should be provided and its allocations can be freed up.

`QoSClear()`: Does not take any argument but removes all QoS configurations pertaining to this object.

`Class QoSTypedInterface` `(derived from QoSInterface)`

A *QoSTypedInterface* combines management information and methods, so that an implementing component can be managed, with the proposed approach, to provide and guarantee network QoS in VEs. For a specific *QoS Category* the *QoSInterface* is refined and extended by a *PropagationRule*. With this combination, the refinement procedure can map and adapt the *QoS Category*, for a specific *LinkType*, while maintaining its semantics and corresponding requirements.

`Class QoSLinkCapabilities` `(derived from CIM_EnabledLogicalElement-Capabilities)`

This derivation from *CIM_EnabledLogicalElementCapabilities* models a *CIM_ComputerSystem*'s capabilities to provide network QoS in VEs. To that end it must always attribute an OSI layer to the component, as the layer where the component fulfils it's main purpose. The concrete capabilities are an aggregation of *QoSTypedInterface*s, which can be read as a list of *LinkType*s for which the component can realise network QoS in VEs.

Function class	Purpose
_COMPARE	Compare two values a and b. Result can be: “a is better”, “a is worse”, “a and b are equivalent”
_SELECT_BEST	Return the best value of a given value set.
_SELECT_WORST	Return the worst value of a given value set.
_AGGREGATE_LINKS	Aggregate property values of two links or paths.
_AGGREGATE_LINKPARTS	Aggregate two partial views at the same link to a single link weight.

Table C.1.: Function classes from [YHSH 10]

`Class PropagationRule`

A *PropagationRule* represents a method for applying a *QoS Category* to a path of links. This concept has been previously published in [Metz 12], based on previous work [YHSH 10], where a schema for generic treatment of network connection properties has been devised. Consequently, the *PropagationRule* is the complement to *QoSInterface* to enable configuration and monitoring of network QoS.

Table C.1 shows the table of function classes for operations on a single QoS parameter, previously published in [YHSH 10]. This is reused, but as a scheme for functions on a *QoS Category*, therefore a set of *QoS Parameter*s. The `_AGGREGATE_LINKPARTS` class is therefore reinterpreted to aggregate the set of *QoS Parameter*s to a single value. It's original meaning is targeted at inter domain aspects, not relevant to this model.

These function classes realise the monitoring part of QoS management and their implementations mainly concern *Path* of the *translation* domain, when monitoring data obtained from individual components is correlated to derive information for the *presentation* domain *VirtualInfrastructure*s and their associated classes.

For varying management purposes, there may be *PropagationRule* derivations with multiple methods for each function class. To support automation and its purpose as part of a *QoSTypedInterface* only the `_AGGREGATE` classes

must be implemented. The `_COMPARE` and `_SELECT` classes are for path selection, fitting the area of VN embedding, which is out of the scope of this thesis, but a logical next step in further integrating network management in VEs. The required methods for this class are:

`IsViable()`: A `_COMPARE` operation with dual use:

1. when refining high level configurations, determine if the refined path can meet its QoS requirements, and

2. having configured the system, validate if the new configuration is effective.

`MeasureQoS()`: A `_AGGREGATE_LINKS` operation, used to determine the current state of a path, in terms of QoS relevant data, by retrieving the corresponding values of all its segments and correlating them.

`Evaluate()`: A `_AGGREGATE_LINKPARTS` operation, to gain a measurement of the fulfilment of the *QoS Category* on the whole.

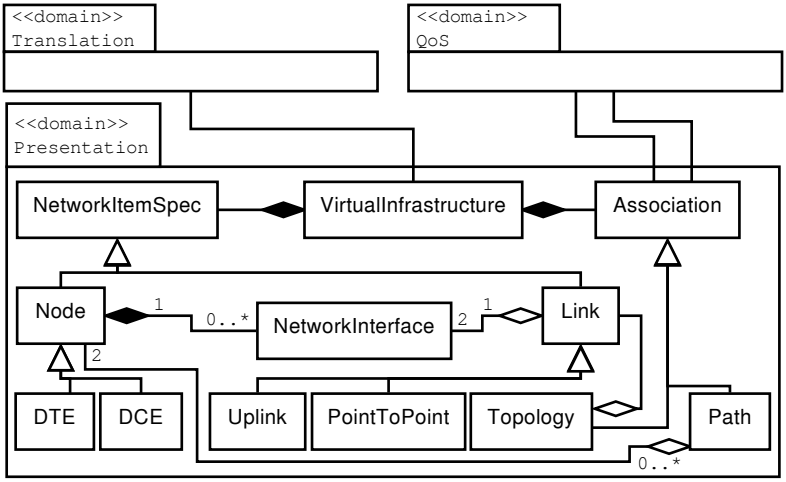
`AdoptMetrics()`: Takes an argument of type *QoS Category* to modify individual *QoS Parameters*, mostly to modify their value range.

`AdoptMeasurement()`: Takes an argument of type *PropagationRule* to change the specific implementations of this *PropagationRule*. This may be required when the host systems of volatile components change.

`MakeVirtualEndpoint()`: Changes the behaviour of the refinement procedure, to treat a *Path* as a *LogicalLink*, to account for network virtualization techniques that should not be managed by this system. For instance a leased line connecting two data centres, or simply components the data centre operators do not want to be automatically managed.

`AccountForProtocol()`: This is used to keep track of protocol overhead through repeated transport and encapsulation, common in VEs.

C.2. Presentation domain



`Class VirtualInfrastructure`

A *VirtualInfrastructure* is the representative of an infrastructure that has been created within a VE. Its main purpose is to group MOs into one entity that represents the fulfilment of the requested infrastructure. As such, it is a composition of *NetworkItemSpec* and *Association* objects. *VirtualInfrastructure* objects can be associated with *Manager*s (cf. Section 6.3.2) as part of access control.

Specifically for the task of QoS management, *VirtualInfrastructure* has a `resourcepool`. It is a list of the VEs allocatable resources, with an upper bound for the accumulated allocation of each resource to the VI's elements. The upper bounds specified in the resource pool need not trigger any allocations. Their minimal use is merely providing reference values for monitoring purposes.

The concrete modelling of allocatable resources is not important for this architecture. Its presence and consistent implementation is required to

compare available resources vs. allocated and employed resources. The quality of allocation modelling therefore greatly influences efficiency of allocation and planning, but not basic applicability. A lightweight example for such a model are the dynamic *SLS* constraints in [FAP 10], while the CIM Resource Allocation Profile [DSP 1041] provides a very sophisticated approach.

The management information stored in this MO can be modified by adding and removing elements form the lists. For the resource pool, the upper boundaries for each resource can be changed individually.

`Class NetworkItemSpec`

This is the parent class for all MOs that are perceived as atomic from any manager's point of view. Consequently *NetworkItemSpec* derivations provide the most refined view on a VE. Their attributes store the managers' intentions as well as the current state of. For sensible monitoring and reports, every *NetworkItemSpec* has an attribute for each resource listed in the composing *VirtualInfrastructure*'s `resourcepool`. This allows for correct calculation and representation of employed resources, even if there have not been concrete specifications.

`Class Node` `(derived from NetworkItemSpec)`

A *Node* is a component that is subject to management, because it fulfils a part of the purpose, most often service, for which the VI was created. *Nodes* have resource allocation attributes that are subsets of the *VirtualInfrastructure*'s `resourcepool`. Specifications for these attributes constitute resource allocation requests. The attribute `placement` contains the hosting entity. For instance the hypervisor running the VM. *Node*s may be endpoints of *Path*s. Every node has a list of *Path*s for which it is an endpoint. Similarly, *Node*s are endpoints of *Link*s, to which they interface through *NetworkInterface*s.

`Class DTE (derived from Node)`

A *DTE* is a refinement of a *Node* and is always an endpoint. It does not forward network traffic.

`Class DCE (derived from Node)`

A *DCE* is a refinement of a *Node* and is always a forwarding component within a VI. *DCE*s can be associated with a *Topology*, making it the sole representative of that topology, for management purposes. The attribute `distributed` controls whether a management system may implement this component as multiple components, each realising a (real) subset of the *DCE*'s *Link*s.

`Class NetworkInterface`

A *NetworkInterface* holds per link information for its corresponding *Node*.

`Class Link (derived from NetworkItemSpec)`

A *NetworkItemSpec* is the MO for links in VIs. It has a list containing exactly two *NetworkInterface*s. *Link*s build *Topology*s and determine how *Path*s may be realised and consequently implemented.

`Class Uplink (derived from Link)`

An *Uplink* is a *Link* connecting one *DTE* with one *DCE*.

`Class PointToPoint (derived from Link)`

A *PointToPoint* is a *Link* connecting two *Node*s of the same type, e.g. two *DTE*s or two *DCE*s.

`Class Association`

This is the parent class for associating network QoS to elements of *Virtual-Infrastructure*. This class merges the selection idea from [FAP 10] with the model introduced in [MNM Roel 05]. Derivations of this class are associated with specific *Data Category* classes. Therefore they map to Sessions

in [MNM Roel 05]. Derivations from *Associations* also specify how *QoS Category*s are to be applied to *NetworkItemSpec*s.

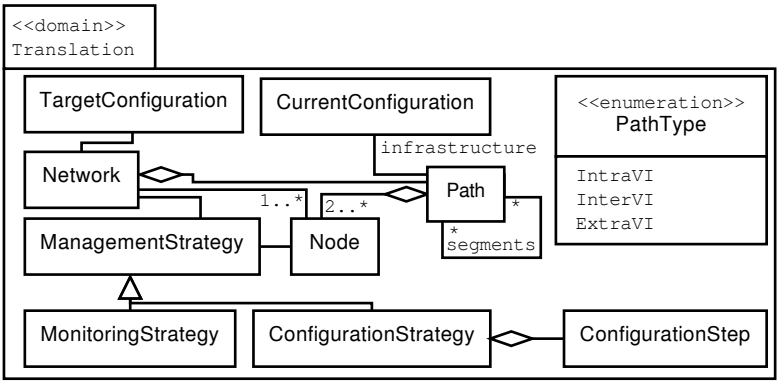
`Class Topology (derived from Association)`

A *Topology* is a selection of links with the intention to facilitate communication between all *Node*s associated with one of the selected links. This *Association* derivation's *Data Category* (*QoS* domain) is a bidirectional unicast, which means any associated *QoS Category*s (*QoS* domain) is to be applied to every link.

`Class Path (derived from Association)`

A *Path* is a selection of exactly two *Node*s with a bidirectional unicast *Data Category*.

C.3. Translation domain



`Class ManagementStrategy`

*ManagementStrategy*s are the backbone of performing management with the proposed approach. The attributes `applicability` and `intention` store information on the circumstances, method and ordering when a *ManagementStrategy* must be implemented. Typical values for `applicability` are "once" and "always", specifying a *ManagementStrategy* may be discarded after its first (and only) implementation or it has to be applied whenever management is performed on the associated *Network*, respectively. The `intention` most commonly stores the goal of the strategy, for instance "component placement", "connectivity", "QoS enforcement". This allows the management system to order and prioritise *ManagementStrategy*s.

`Class MonitoringStrategy (derived from ManagementStrategy)`

A *ManagementStrategy* specifies how management information originating from MOs of the *realisation* domain are attributed to MOs of the *presentation* domain. In it's most compact form, a *MonitoringStrategy* associates an attribute of an object of the *presentation* domain with an attribute of an object of the *realisation* domain. For instance, the CPU time used by a VM is accessed through a *CIM_VirtualSystem* object and copied to a *DTE* object.

In general a *MonitoringStrategy* needs to specify what information it needs from the *realisation* domain what information it will contribute to the *presentation* domain and an implementable procedure which takes the input data and generates the specified output.

For network QoS management *MonitoringStrategy*s often correspond to *QoS Category*s and the types of guarantee associated with their properties. For instance, monitoring for best effort services is a compact strategy facilitating the copying of measured data. At the other extreme, a *MonitoringStrategy* could specify collecting monitoring data in short, fixed time intervals, automatically correlating it with threshold values, and immediately react to threshold violations by modifying specifications of MO attributes and

triggering the creation of *ConfigurationStrategy*s, to make the VE enforce a *QoS Category* on the monitored MOs.

`Class ConfigurationStrategy (derived from ManagementStrategy)`

A *ConfigurationStrategy* is a recipe for a management system that transforms the VE so that it performs more according to the *TargetConfiguration*. To fulfil its task, a *ConfigurationStrategy* aggregates *ConfigurationStep*s into an ordered list. Executing the steps of the list is considered implementing the strategy. Objects of this class are used for coordinating and controlling implementations. Grouping steps to strategies, yields discrete states of the VE where assertions can be made according to the `intention` of the strategies. This allows for partial fulfilment and fault management, giving managers and the management system opportunities to adjust specifications, rather than completely denying the implementation of a *TargetConfiguration*.

`Class ConfigurationStep`

A *ConfigurationStep* is an atomic operation of a *ConfigurationStrategy*. One step often corresponds to a parametrised call to a management function of a component within the VE. The *ConfigurationStep* has an attribute `errorhandling`, which specifies the effect on the superordinate *ConfigurationStrategy*, when an error is encountered during the call to the management function.

`Class TargetConfiguration`

A *TargetConfiguration* is a collection of all currently active *VirtualInfrastructure*s domain. Active VIs have their components placed within the VE. This means there are *CIM_ComputerSystem* (-derivation) instances, that map 1:1 to *Node*s and the *Link*s and *Association*s of the *presentation* domain can be mapped onto sequences of *CIM_ComputerSystem*s (-derivations) and *LogicalLink*s of the *realisation* domain.

`Class CurrentConfiguration`

A *CurrentConfiguration* is a collection of all *LogicalLink*s and their endpoints where both endpoints are currently active.

`Class Network`

The components possibilities to communicate are modelled as *Node*s associated with *Network*s. It corresponds most to *Association*, but its main task is to group elements from the *presentation* domain with the *Node*s and *Path*s of the *translation* domain. *Network*s are the working sets for the refinement procedure and also tie *ManagementStrategy*s to *VirtualInfrastructure*s.

`Class Node`

A *Node* within the scope of the *translation* domain is essentially a stub that is part of a *Network* or *Path*. Initially, *translation Node*s are created for *presentation Node*s. As *Path*s are created and refined additional *Node*s may be "discovered". To be referenced in a *ManagementStrategy*, a *Node* must be associated with a MO from the *realisation* domain.

`Class Path`

A *Path* of the *translation* domain is the model's main information hub for mapping network QoS from VIs on the VE. When a *Path* is created its endpoints are analysed to determine the *Path*'s `type`. The corresponding enumeration *PathType* is:

`IntraVI`: Both endpoints are part of the same *VirtualInfrastructure* and corresponding *Node*s can be obtained canonically.

`InterVI`: Both endpoints are in this VE, but not part of the same *VirtualInfrastructure*. There must be a "gateway" between both VIs and QoS must be configured for both VIs. This may include extending the *TargetConfiguration*.

`ExtraVI`: One endpoint is outside this VE. A separate mechanism must be employed to determine how and where data traffic enters and leaves the VE and VI. A *Node* must be created that marks the end of the path segment which can be managed as part of this VE. Management of the path beyond that *Node* must be delegated, i.e. cannot be guaranteed.

Only when `type` is *IntraVI*, refinement as described in Section 6.2 can be performed. All other types require the creation of subpaths stored in the *segments* list, that are either *IntraVI* or can be delegated completely.

The endpoints of a *Path* are referred to as `start` and `end`, to imply a direction and a corresponding search or traversing order. This also allows for directed *QoS Category*s, if a derivation from *Association* with a directed *Data Category* was created. For bidirectional *Association*s, two *Path*s must be created.

When a *Path* cannot be refined any further, either because its management is delegated completely, or it has a corresponding *LogicalLink* from the *realisation* domain, it is `atomic`. When it is `atomic`, its *Node* endpoints can be associated with their implementing components of the *realisation* domain and a *ConfigurationStrategy* for implementing the *QoS Category*, retrieved from an *Association*, can be devised using the associated *QoSTypedInterface*.

Any *Path* that is not `atomic` is refined into subpaths, stored as `segments`. When refining a *Path*, usually new *Node*s are must be added to the *Network*. By accessing the associated *CurrentConfiguration*, a *Path* retrieves the information especially necessary to implement the methods `PathAtOsiLayer()`, `NextNeighbourOf()` and `PrevNeighbourOf()`.

The methods of this class are:

`PathAtOsiLayer()`: Takes an integer as argument, numbering the layer of the ISO OSI reference model for this request. Returns a copy of this *Path* with `segments` filled with all subpaths at the requested OSI layer from `start` to `end`.

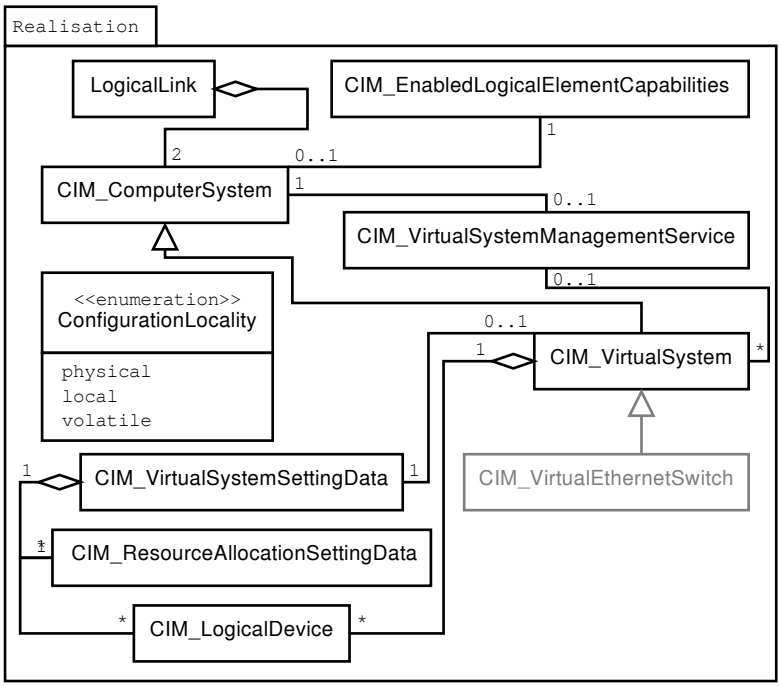
`NextNeighbourOf()`: Takes a *Node* and an integer as arguments. The integer is an OSI layer, analogous to `PathAtOsiLayer()`. Semantically, this method first calls `PathAtOsiLayer()` with the provided OSI layer, to get its result. It then gets the *Path* form the result's `segments`, where `start` is the provided *Node*. If such a *Path* exists in the result's `segments`, then its `end` property is returned, otherwise the provided *Node* has no neighbour on the provided OSI layer on this *Path*.

`PrevNeighbourOf()`: Analogous to `NextNeighbourOf()`, but selecting the segment where `end` is the provided *Node* and `start` is the returned property.

`AdoptQoS()`: Takes a *Node* as argument, and

1. calls `MakeVirtualEndpoint()` if the argument's `layer` is 7,
2. calls `AccountForProtocol()` if the argument's `layer` is higher, i.e. greater, than `start.layer`,
3. calls `AdoptMetrics()` on the subpath from `start` to the argument
4. calls `AdoptMetrics()` on the subpath from the argument to `end`

C.4. Realisation domain



Class LogicalLink

A *LogicalLink* is a direct connection between two derivations of *CIM_ComputerSystem*, such that no intermediary managed component handles the data traffic between the endpoints. Classifying this capability using the ISO OSI model, this is layer 2 functionality, hence a logical link. Representing an existing communication path, this class stores its *LinkType*. The actual implementation of QoS for the link is realised by its endpoints.

`Class CIM-ComputerSystem`

This class from the CIM has been extended by three methods to provide information required by the procedure developed in Section 6.2.

`GetConfLocality()`: Does not take any arguments and returns the locality of configuration of a component as type *ConfigurationLocality*.

`GetSupportedLinks()`: Takes one argument of type *QoS Category* and returns a list of *LinkType*s, which are all *LinkType*s for which this particular component can implement the provided QoS Category.

`GetQoSCapacity()`: Takes one argument of type *QoS Category* and returns a value quantifying this components capacity to provide this particular *QoS Category*. Defining the meaning and range of the value is part of and specific to the *QoS Category*.

Bibliography

[AAC+ 03] Akyildiz, I. F., T. Anjali, L. Chen, J. C. de Oliveira, C. Scoglio, A. Sciuto, J. A. Smith and G. Uhl: *A new traffic engineering manager for DiffServ/MPLS networks: design and implementation on an IP QoS Testbed*. Computer Communications, 26(4):388 -- 403, 2003.

[APST 05a] Anderson, T., L. Peterson, S. Shenker and J. Turner: *Overcoming the Internet impasse through virtualization*. Computer, 38(4):34--41, 2005.

[APST 05b] Anderson, T., L. Peterson, S. Shenker and J. Turner: *Overcoming the Internet impasse through virtualization*. Computer, 38(4):34--41, April 2005.

[BCL+ 14] Berman, M., J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci and I. Seskar: *GENI: A federated testbed for innovative network experiments*. Computer Networks, 61(0):5 -- 23, 2014. Special issue on Future Internet Testbeds - Part I.

[Bell 57] Bellman, Richard E.: *Dynamic Programming*. Univ. Press, Princeton, NJ, USA, 1957.

[BFH+ 06] Bavier, A., N. Feamster, M. Huang, L. Peterson and J. Rexford: *In VINI veritas: realistic and controlled network experimentation*. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 3--14. ACM, 2006.

[BGRS 11] Burd, S. D., G. Gaillard, E. Rooney and A. F. Seazzu: *Virtual Computing Laboratories Using VMware Lab Manager*. In *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*, HICSS '11, pages 1--9, Washington, DC, USA, 2011. IEEE Computer Society.

[BKFS 07] Bradford, R., E. Kotsovinos, A. Feldmann and H. Schiöberg: *Live wide-area migration of virtual machines including local persistent state.* In *Proceedings of the 3rd international conference on Virtual execution environments*, pages 169--179. ACM, 2007.

[BLG+ 07] Boucadair, M., P. Levis, D. Griffin, N. Wang, M. Howarth, G. Pavlou, E. Mykoniati, P. Georgatsos, B. Quoitin and J. et. al. Rodriguez Sanchez: *A framework for end-to-end service differentiation: Network planes and parallel Internets.* Communications Magazine, IEEE, 45(9):134--143, 2007.

[CaJi 09] Carapinha, J. and J. Jiménez: *Network Virtualization: A View from the Bottom.* In *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, pages 73--80, New York, NY, USA, 2009. ACM.

[ChBo 09] Chowdhury, N. M. M. K. and R. Boutaba: *Network virtualization: state of the art and research challenges.* Communications Magazine, IEEE, 47(7):20--26, 2009.

[CXB+ 12] Cui, Z., L. Xia, P. G. Bridges, P. A. Dinda and J. R. Lange: *Optimizing overlay-based virtual networking through optimistic interrupts and cut-through forwarding.* In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 99. IEEE Computer Society Press, 2012.

[DSP 1041] Distributed Management Taskforce: *CIM Resource Allocation Profile.* DMTF Standard 1041, DMTF, June 2009.

[dSYF 01] Silva, S. da, Y. Yemini and D. Florissi: *The NetScript active network system.* Selected Areas in Communications, IEEE Journal on, 19(3):538--551, 2001.

[FAB+ 11] Fajjari, I., M. Ayari, O. Braham, G. Pujolle and H. Zimmermann: *Towards an Autonomic Piloting Virtual Network Architecture.* In *NTMS*, pages 1--5. IEEE, IEEE, 2011.

[FAP 10] Fajjari, I., M. Ayari and G. Pujolle: *VN-SLA: A Virtual Network Specification Schema for Virtual Network Provisioning.* In *Networks (ICN), 2010 Ninth International Conference on*, pages 337--342, April 2010.

[FBA+ 11] Fajjari, I., O. Braham, M. Ayari, G. Pujolle and H. Zimmermann: *AAVP: An Innovative Autonomic Architecture for Virtual network Piloting.* International Journal of Next-Generation Computing, 2(3), 2011.

[FeHu 98] Ferguson, P. and G. Huston: *Quality of service: delivering QoS on the Internet and in corporate networks.* Wiley New York, NY, 1998.

[FGR 07] Feamster, N., L. Gao and J. Rexford: *How to lease the Internet in your spare time.* ACM SIGCOMM Computer Communication Review, 37(1):61--64, 2007.

[FHTG 10] Fiedler, M., T. Hossfeld and P. Tran-Gia: *A generic quantitative relationship between quality of experience and quality of service.* Network, IEEE, 24(2):36--41, 2010.

[FSBR 04] Freeman, E., K. Sierra, B. Bates and E. Robson: *Head First design patterns.* O'Reilly, Sebastopol, CA, 2004.

[GHHK 01] Garschhammer, M., R. Hauck, H.-G. Hegering, B. Kempter, M. Langer, M. Nerb, I. Radisic, H. Roelle and H. Schmidt: *Towards generic Service Management Concepts -- A Service Model Based Approach.* In *Proceedings of the 7th International IFIP/IEEE Symposium on Integrated Management (IM 2001)*, pages 719--732, Seattle, Washington, USA, May 2001. IFIP/IEEE, IEEE Publishing.

[GHKR 01] Garschhammer, M., R. Hauck, B. Kempter, I. Radisic, H. Roelle and H. Schmidt: *The MNM Service Model -- Refined Views on Generic Service Management.* Journal of Communications and Networks, 3(4):297--306, December 2001.

[HAN 99] Hegering, H.-G., S. Abeck and B. Neumair: *Integrated Management of Networked Systems -- Concepts, Architectures and their Operational Application.* Morgan Kaufmann Publishers, 1999.

[HZSL+ 08] He, J., R. Zhang-Shen, Y. Li, C.-Y. Lee, J. Rexford and M. Chiang: *DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet.* In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 15:1--15:12, New York, NY, USA, 2008. ACM.

[JBR 99] Jacobson, I., G. Booch and J. Rumbaugh: *The Unified Software Development Process.* Addison--Wesley, January 1999.

[JiNa 04] Jin, J. and K. Nahrstedt: *QoS specification languages for distributed multimedia applications: A survey and taxonomy.* Multimedia, IEEE, 11(3):74--87, 2004.

[KCC+ 01] Kounavis, M. E., A. T. Campbell, S. Chou, F. Modoux, J. Vicente and H. Zhuang: *The genesis kernel: A programming system for spawning network architectures.* Selected Areas in Communications, IEEE Journal on, 19(3):511--526, 2001.

[KeEi 06] Kemper, A. and A. Eickler: *Datenbanksysteme: eine Einführung.* Oldenbourg, 2006.

[LRZ 12] LRZ press release: *SuperMUC No 4 of the Top500 List*, June 2012. [online], Accessed: Oct, 16, 2012.

[LSD 05] Lange, J. R., A. I. Sundararaj and P. A. Dinda: *Automatic dynamic run-time optical network reservations.* In *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 255--264. IEEE, 2005.

[MAB+ 08] McKeown, N., T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker

and J. Turner: *OpenFlow: Enabling Innovation in Campus Networks*. SIGCOMM Comput. Commun. Rev., 38(2):69--74, March 2008.

[MaTo 06] Martello, S. and P. Toth: *Knapsack problems: algorithms and computer implementations*. UMI Books on Demand, Ann Arbor, Mich., 2006.

[Metz 09] Danciu, V. A. and Martin G. Metzker: *On I/O virtualization management*. In *Proceedings of the 3rd International DMTF Workshop on Systems and Virtualization Management*, volume 2009, Wuhan, China, September 2009. Distributed Management Task Force.

[Metz 10] Metzker, Martin G. and V. A. Danciu: *Towards end-to-end management of network QoS in virtualized infrastructures*. In *Systems and Virtualization Management (SVM), 2010 4th International DMTF Academic Alliance Workshop on*, pages 21--24, October 2010.

[Metz 11] Danciu, V., N. gentschen Felde, M. Kasch and Martin G. Metzker: *Bottom--up harmonisation of management attributes describing hypervisors and virtual machines*. In *Proceedings of the 5th International DMTF Workshop on Systems and Virtualization Management: Standards and the Cloud (SVM 2011)*, volume 2011, Paris, France, October 2011. Distributed Management Task Force (DMTF), IEEE Xplore.

[Metz 12] Yampolskiy, M., W. Hommel, F. Liu, R. König, Martin G. Metzker and M. Schiffers: *Link Repair in Managed Multi-domain Connections with End-to-end Quality Guarantees*. Int. J. Netw. Manag., 22(6):494--507, November 2012.

[Metz 14a] Metzker, Martin G. and D. Kranzlmüller: *Exploring Virtuality -- Virtualität im interdisziplinären Diskurs*, chapter Dienstgütemanagement für Netze in Virtualisierungsumgebungen. Springer Fachmedien Wiesbaden, Wiesbaden, Germany, 2014.

[Metz 14b] Metzker, Martin G. and D. Kranzlmüller: *Mapping virtual paths in virtualization environments*. In Müller, P., B. Neumair, H. Reiser and G. Dreo Rodosek (editors): *7. DFN-Forum Kommunikationstechnologien: Verteilte Systeme im Wissenschaftsbereich, Beiträge der Fachtagung, 16.-17. Juni 2014, Fulda*, volume 2014 of *Lecture Notes in Informatics*, Bonn, Germany, June 2014. Gesellschaft für Informatik e.V., Köllen Druck+Verlag GmbH.

[MNM Dreo 02] Dreo Rodosek, G.: *A Framework for IT Service Management*. Habilitation, Ludwig--Maximilians--Universität München, June 2002.

[MNM Gars 04] Garschhammer, M.: *Dienstgütebehandlung im Dienstlebenszyklus --- von der formalen Spezifikation zur rechnergestützten Umsetzung*. Dissertation, Ludwig--Maximilians--Universität München, July 2004.

[MNM Lind 10] Lindinger, T.: *Optimierung des Wirkungsgrades virtueller Infrastrukturen*. Dissertation, Ludwig--Maximilians--Universität München, February 2010.

[MNM Marc 11] Marcu, P.: *Architekturkonzepte für interorganisationales Fehlermanagement*. Dissertation, Ludwig--Maximilians--Universität München, May 2011.

[MNM Roel 05] Roelle, H.: *Eine dienstorientierte Methodik zur Koppelung von Netz--QoS--Architekturen*. Dissertation, Ludwig--Maximilians--Universität München, July 2005.

[MNM Scha 08] Schaaf, T.: *IT--gestütztes Service--Level--Management --- Anforderungen und Spezifikation einer Managementarchitektur*. Dissertation, Ludwig--Maximilians--Universität München, December 2008.

[MNM Schi 07] Schiffers, M.: *Management dynamischer Virtueller Organisationen in Grids*. Dissertation, Ludwig--Maximilians--Universität München, July 2007.

[MNM Yamp 09] Yampolskiy, M.: *Maßnahmen zur Sicherung von E2E--QoS bei Verketteten Diensten*. Dissertation, Ludwig--Maximilians--Universität München, December 2009.

[MVKK 12] Mann, V., A. Vishnoi, K. Kannan and S. Kalyanaraman: *CrossRoads: Seamless VM mobility across data centers through software defined networking*. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 88--96, April 2012.

[NJC+ 99] Ng, W. F., D. S. Jun, H. K. Chow, R. Boutaba and A. Leon-Garcia: *MIBlets: a practical approach to virtual network management*. In *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on*, pages 201--215, 1999.

[PACR 03] Peterson, L., T. Anderson, D. Culler and T. Roscoe: *A blueprint for introducing disruptive technology into the Internet*. ACM SIGCOMM Computer Communication Review, 33(1):59--64, 2003.

[PGP+ 10] Pettit, J., J. Gross, B. Pfaff, M. Casado and S. Crosby: *Virtual switching in an era of advanced edges*. In *2nd Workshop on Data Center--Converged and Virtual Ethernet Switching (DC-CAVES)*, 2010.

[PPK+ 09] Pfaff, B., J. Pettit, T. Koponen, K. Amidon, M. Casado and S. Shenker: *Extending Networking into the Virtualization Layer*. In *ACM workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[Pujo 08] Pujolle, G.: *An Autonomic Architecture for Network Management and Control*. UPGRADE, 2008(VI), December 2008.

[RJXG 05] Ruth, P., X. Jiang, D. Xu and S. Goasguen: *Virtual distributed environments in a shared infrastructure*. Computer, IEEE, 38(5):63--69, 2005.

[RoRo 06] Robertson, S. and J. Robertson: *Mastering the Requirements Process (2Nd Edition)*. Addison-Wesley Professional, March 2006.

[RyKo 13] Rygielski, P. and S. Kounev: *Network Virtualization for QoS-Aware Resource Management in Cloud Data Centers: A Survey*. PIK --- Praxis der Informationsverarbeitung und Kommunikation, 36(1):55--64, February 2013.

[Scha 12] Schaffrath, G.: *Virtual Network Management*. Dissertation, Technischen Universität, Berlin, Germany, December 2012.

[Schö 03] Schöning, U.: *Theoretische Informatik - kurzgefasst*. Spektrum Akademischer Verlag, 4. A. (korrig. Nachdruck 2003) edition, 2003.

[SGD 04] Sundararaj, A. I., A. Gupta and P. A. Dinda: *Dynamic topology adaptation of virtual networks of virtual machines*. In *Proceedings of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems*, ACM International Conference Proceeding Series, pages 1--8, New York, NY, USA, 2004. ACM, ACM.

[Shan 09] Shankar, S.: *Amazon elastic compute cloud*, 2009. [online], Accessed: Oct, 16, 2011.

[SMEVH 10] Stanley-Marbell, P., N. P. Evans and E. Van Hensbergen: *A unified execution model for cloud computing*. ACM SIGOPS Operating Systems Review, 44(2):12--17, 2010.

[SoAn 10] Soundararajan, V. and J. M. Anderson: *The Impact of Management Operations on the Virtualized Datacenter*. SIGARCH Comput. Archit. News, 38(3):326--337, June 2010.

[SuDi 04] Sundararaj, A. I. and P. A. Dinda: *Towards Virtual Networks for Virtual Machine Grid Computing*. In *Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*, pages 177--190, 2004.

[SUN 09] SUN MICROSYSTEMS: *Crossbow: Network Virtualization and Resource Control*, October 2009. [online], Accessed: Mar, 17, 2010.

[SWP+ 09] SCHAFFRATH, G., C. WERLE, P. PAPADIMITRIOU, A. FELDMANN, R. BLESS, A. GREENHALGH, A. WUNDSAM, M. KIND, O. MAENNEL and L. MATHY: *Network Virtualization Architecture: Proposal and Initial Prototype.* In *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, pages 63--72, New York, NY, USA, 2009. ACM.

[TaWe 10] TANENBAUM, A. S. and D. WETHERALL: *Computer Networks.* Pearson Education, Limited, New York, 5th Revised edition. edition, 2010.

[TaYa 10] TAI, J. PIAO and J. YAN: *A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing.* In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 87--92, Nov 2010.

[TDS+ 09] TRIPATHI, S., N. DROUX, T. SRINIVASAN, K. BELGAIED and V. IYER: *Crossbow: a vertically integrated QoS stack.* In *ACM workshop on Research on enterprise networking*, pages 45--54, New York, NY, USA, 2009. ACM.

[TDSB 09] TRIPATHI, S., N. DROUX, T. SRINIVASAN and K. BELGAIED: *Crossbow: from hardware virtualized NICs to virtualized networks.* In *ACM workshop on Virtualized infrastructure systems and architectures*, pages 53--62, New York, NY, USA, 2009. ACM.

[Touc 01] TOUCH, J.: *Dynamic Internet overlay deployment and management using the X-Bone.* Computer Networks, 36(2):117--135, 2001.

[VdMR+98] MERWE, J. E. VAN DER, S. ROONEY, I. LESLIE and S. CROSBY: *The tempest-a practical framework for network programmability.* Network, IEEE, 12(3):20--28, 1998.

[VKO+ 09] Voith, T., M. Kessler, K. Oberle, D. Lamp, A. Cuevas, P. Mandic and A. Reifert: *Interactive Real-time Multimedia Applications on Service Oriented Infrastructures*. Technical Report, IRMOS Consortium 2008, July 2009.

[Walt 88] Walton, M.: *The Deming Management Method*. Perigee, New York, 1988.

[WaNg 10] Wang, G. and TS E. Ng: *The impact of virtualization on network performance of amazon ec2 data center*. In *INFOCOM, 2010 Proceedings IEEE*, pages 1--9. IEEE, 2010.

[WCS+ 03] Wu, J., S. Campbell, J. M. Savoie, H. Zhang, G. v. Bochmann and B. St. Arnaud: *User-managed end-to-end lightpath provisioning over CA* net 4*. In *Proceedings of the National Fiber Optic Engineers Conference (NFOEC)*, pages 275--282, 2003.

[WKB+ 08] Wang, Y., E. Keller, B. Biskeborn, J. van der Merwe and J. Rexford: *Virtual Routers on the Move: Live Router Migration As a Network-management Primitive*. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 231--242, New York, NY, USA, 2008. ACM.

[XCL+ 12] Xia, L., Z. Cui, J. R. Lange, Y. Tang, P. A. Dinda and P. G. Bridges: *VNET/P: Bridging the cloud and high performance computing through fast overlay networking*. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pages 259--270. ACM, 2012.

[YHSH 10] Yampolskiy, M., W. Hommel, D. Schmitz and M. K. Hamm: *Generic Function Schema for Operations on Multiple Network QoS Parameters*. November 2010.

[Zhan 87] Zhang, L.: *Designing a new architecture for packet switching communication networks*. IEEE Communications Magazine, 25(9):5--12, 1987.