Master Thesis

Model-Based Network Analysis and Optimization

Stefan Herrnleben
Department of Computer Science
Chair for Computer Science II (Software Engineering)

Prof. Dr.-Ing Samuel Kounev
First Reviewer

Prof. Dr.-Ing Phuoc Tran-Gia
Second Reviewer

M. Sc. Johannes Grohmann
Advisor

Dr. rer. nat. Piotr Rygielski
Second Advisor

Submission
23. October 2017

www.uni-wuerzburg.de
I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Würzburg, 23. October 2017

---------------------------------------------

Stefan Herrnleben
Abstract

Emerging technologies like cloud computing have changed the usage of data centers in the last years. While a few decades ago, often dedicated use-case specific hardware has been deployed, today’s data centers are faced with dynamic allocation of shared resources. Beside the availability of the resources, a data center provider has to ensure Quality of Service (QoS) criteria like bandwidth or response time, contracted with customers in Service Level Agreements (SLAs).

The dynamic, high frequent resource allocation and redistribution requires continuous monitoring and observation of the agreed objectives. A proactive approach predicts SLA violations before bottlenecks occur, in a reactive scenario the current system state is used to assess the SLA fulfillments. As violations are forecasted or occur, the resources in the data center have to be reconfigured, to prevent or solve a broken agreement.

Self-aware systems adapt themselves automatically, based on the current workload and specified degrees of freedom, without any human interaction. Existing approaches for dynamic resource management often focus on computing resources like auto-scaling or placement of Virtual Machines (VMs). The network part is commonly abstracted or modeled in a simplified manner. Other existing works deal with network performance analysis, usually via a simulative approach, but these analyses do not consider the adaptation of networks. Especially modeling the degrees of freedom is not supported by such simulation frameworks.

This thesis focuses on the model-based adaptation and optimization of networks. Based on observed SLAs, adaptations for the network are suggested, if a SLA violation is detected. The model-based approach introduces two benefits. Firstly, the suggested solution can be evaluated through an analysis to ensure that the violation is solved and especially no other violations are generated through reconfiguration. At second, the model-based approach allows executions of what-if analyses on workloads, predicted by forecasters, or planned reconfigurations.

One contribution of this work is the development of an adaptation points model as extension to the Descartes Network Infrastructures Modeling (DNI) language. Additionally, other models for the adaptation process are developed. A second contribution is a target-oriented, runtime-optimized adaptation approach to discover the cost-optimal solutions for network adaptations, which solve SLA violations. The third contribution is the implementation of the developed adaptation models and adaptation process in an adaptation framework for DNI.

The target-oriented adaptation process has to fulfill several requirements, related to cost-optimality and runtime-optimization. These requirements are assessed in a qualitative and quantitative evaluation. It is shown that the size of the network has no direct impact on the runtime of the adaptation process, which makes the approach suitable for large networks. In comparison to a brute force approach, the developed adaptation process discovers the cost-optimal solutions up to 250 times faster.
Aufstrebende Technologien wie die Berechnung und Speicherung von Daten in der Cloud veränderten die Nutzung von Rechenzentren in den letzten Jahren. Vor wenigen Jahrzehnten wurde oftmals noch dedizierte, meist für einen bestimmten Anwendungsfall vorgehene Hardware eingesetzt. Heutige Rechenzentren hingegen werden mit der dynamischen Zuweisung von Ressourcen konfrontiert, die von mehreren Kunden gemeinsam genutzt werden. Neben der Verfügbarkeit der Ressourcen, müssen die Betreiber von Rechenzentren auch Qualitätskriterien \( (QoS) \) wie die bereitgestellte Bandbreite oder die Antwortzeit eines Dienstes garantieren. Diese Qualitätskriterien werden in einem Vertrag \( (SLA) \) festgehalten.


Ein Beitrag dieser Arbeit ist die Entwicklung eines Modells für alternative Konfigurationen als Erweiterung für die Netzwerkmodellierungssprache \( (DNI) \) der Descartes Forschergruppe. Des Weiteren werden andere Modelle für einen Anpassungsprozess entwickelt. Ein zweiter Beitrag ist ein zielgerichteter, laufzeitoptimierter Anpassungsprozess, der kostenoptimale Lösungen für die Behebung von Vertragsverletzungen in Netzwerken bietet.

6. **Implementation** 83
   6.1. Dependencies ........................................... 83
   6.2. Inputs ..................................................... 85
   6.3. Outputs ................................................... 87
   6.4. Structure of Java Project ............................ 89
   6.5. Integration of Additional Tactics .................. 91
   6.6. Example Execution of Adaptation Framework .... 92
   6.7. Licensing .................................................. 93

7. **Evaluation** 95
   7.1. Requirements and Assessment for Adaption Models .. 95
   7.2. Experimental Setup .................................... 100
   7.3. Validation of Adaptation Process Requirements ... 102
   7.4. Quantitative Assessments (Case Studies) .......... 115

8. **Conclusion** 125
   8.1. Summary .................................................. 125
   8.2. Limitations ............................................... 127
   8.3. Future Work .............................................. 128

List of Figures 130

List of Tables 131

Listings 133

Acronyms 135

Bibliography 139

Appendix 143
   A. Adaptation Framework Input Examples ............... 143
   B. Adaptation Framework Output Examples ............ 144
1. Introduction

Emerging technologies like virtualization and cloud computing changed the usage of data centers in the last years. Originally, dedicated hardware, often intended for a fixed use case, has been located in data centers. Virtualization technologies introduced a new era by improving the utilization of physical servers. Nowadays, the idea of virtualization is expanded to entire data center infrastructures to build dynamic, agile, service-oriented architectures [1]. These infrastructures include servers, storage and network components [2]. Such dynamic approaches allow customers to allocate resources on demand to provide scalable services, without considering the underlying hardware.

For customers it is not only relevant that the booked services are up and running, also qualitative characteristics are important. The so called Quality of Service (QoS) includes response time, transmission delay, throughput, and reliability. Thresholds for these metrics are usually specified as objectives in a Service Level Agreement (SLA) between provider and customer. The provider is obligated to fulfill these requirements, as the customer - usually - pays for it.

A data center provider should rate the performance demands and reconfigure the provided resources if required, before deploying new customers or services. This approach suits for traditional data centers, where dedicated hardware was used. However, in dynamic environments, where customers scale their resources up and down or allocate additional resources, an initial performance analysis is not sufficient. Instead of this, the agreed SLAs have to be continuously observed and the data center operator has to react, if a SLA is violated in order to fulfill the customers performance expectations.

1.1. Motivation

Before adapting a data center infrastructure, the required changes have to be identified, which would solve the SLA violation. For the decision process, two methods can be named for how to find a solution. One method is an educated guess, where network experts analyze the network, determine bottlenecks and suggest adaptations. In practice, this approach is unsuitable for cloud environments, where appropriate solutions have to be discovered and applied automatically. Additionally, it could be difficult to make educated guesses for complex networks, where an adaptation possibly influences another parameter and therefore violates other SLAs.
A second approach is to automatically check for SLA violations and automatically adapt the network to ensure that the agreed expectations are fulfilled. This enables a self-aware network. But even for an adaptation, discovered by computers, it is difficult to identify mutual influencing adaptations and impede them. To predict the influence of an adaptation, the resulting performance should be determined through an analytical method and the result should be validated against the SLAs. This approach requires to model the network including the configuration and workload and run an analysis on the network model. The impact of possible adaptations to SLAs can be investigated on the model, before deploying them on the real world network.

A model-based approach can be used in two different ways. In a reactive way, the network is continuously or periodically monitored for SLA violations. As soon as SLA violations are detected, appropriate solutions are discovered through an adaptation process and validated by an analysis, before the changes are directly applied to the network or provided as possible solutions. In a proactive way, the workload on the system is predicted, e.g., by a forecaster, and SLA violations can be predicted on the network model, before they occur on the real network [3]. Adaptations to prevent such violations could be applied before bottlenecks occur.

In addition to the SLA monitoring for already deployed systems, a model-based approach could also be used for what-if analyses. The workload model could be artificially increased to investigate the behavior of the network under an increased traffic assumption. Hardware changes, e.g., replacing specific devices, could be simulated. New customers with additional hardware can be integrated into a model, to decide if they could be affiliated to the data center without impact on the agreed SLAs of other customers.

Current approaches for automatic resource management focus on computing resources. For computing resources often vertical scaling, for adjusting system properties, and horizontal scaling, for adding additional instances, is used. Migration of resources like the relocation of VMs to another hypervisor is a further valid optimization for computing resources.

But these approaches often skip optimizations on the network or provide only limited adaptation suggestions for networks. Like computing resources, also networks could be adapted. The possible adaptations could be short-term software optimizations, like changing a route, up to long-term hardware changes, like purchasing a network device with more performance. Especially emerging technologies like SDN and NFV introduce several new possibilities of how networks could be adapted, even at runtime without hardware changes.

Currently, no approach for such an automatic adaptation is known. Some tools provide optimization of networks, but they are often limited to specific features. This could result in optimizing one objective, while another objective neglected. An approach with a global view over the network, considering all agreed SLAs for a self-aware network optimization and adaptation is desired.

Scope of this thesis is the model-based analysis and optimization of networks. This includes the definition of the available adaptations as well as a sophisticated adaptation process. For the modeling approach, the Descartes Network Infrastructures Modeling (DNI) language is selected, as it provides the required fine-grained modeling features for networks and supports analyzers with extensive measurement results. Before an adapted model is suggested as solution, an analysis validates the performance of the complete model and checks, if any SLA is violated. The solutions are output from the adaptation process as cost-optimal solutions to the user. An automatic selection of a single best solution through weighting between different parameters is out of scope of this work. The technical part of applying changes to a network is also not part of this work.
1.2. Problem

The problem of a model-based network adaptation can be defined as follows. A network is described in a [DNI] network infrastructure model. Beside structural information of the network, this model also contains the network configuration and the specification of workload. In addition to the network infrastructure, the valid adaptations, annotated by costs, have to be specified in a model. The agreed [SLAs] representing the [QoS] expectations on a network, also have to be defined.

An adaptation process has to detect [SLA] violations on the network infrastructure model, and has to search for possible adaptations which would solve the occurred violations. The output is a list of required adaptation actions to the given input model in order to solve the [SLA] violation. If there is not valid adaptation, none is returned. If multiple solutions solve the problem, a set of solving adaptation actions is returned. These adaptations are cost-optimal.

1.3. Approach

The approach for the model-based analysis and optimization of networks can be divided into two parts. One part deals with the modeling of network, available adaptations and [SLAs]. The existing network model of [DNI] is used, which benefits in a fine-grained degree of details and the availability of solvers, used to analyze the performance of the modeled network. A [DNI] network infrastructure model includes the network structure, the configuration and the modeled traffic. In order to determine possible adaptations, the valid and available adaptations have to be defined for the different entities and parameters of the network in a so called adaptation points model. This adaptation points model also includes a multi-dimensional cost model, which is necessary to determine cost-optimal solutions. To validate the performance prediction from an analysis against the agreed [SLAs] the analysis results have to be available as a preferably generic analysis result model and also the [SLAs] are required to be defined in a model. The scheduled adaptation actions are provided in an adaptation plan model, which is a further part of the modeling approach of this thesis.

The second part of the approach describes the adaptation process. This includes an algorithmic approach of how to detect [SLA] violations, find alternatives and analyze the solution candidates, before outputting them. For the adaptation process, an iterative [MAPE-K] adaptation control loop with branch and bound will be used, to adapt the network model step by step. Each iteration starts with an analysis and the results are validated against the provided [SLAs]. Based on the violated objectives possibly several target-oriented adaptation algorithms are used, to discover valid alternatives. The target orientation means that only such algorithms are executed, which will solve the violation, and facilitates to reduce the runtime. Each adaptation algorithm can output multiple solutions as branches from the input model. Subsequent filter modules bound branches, which are useless for the following adaptation process. For example, branches exceeding the number of maximum allowed adaptation actions can be bounded, as all subsequent adaptations on these branches will increase the number of adaptation actions even more. After traversing the filter modules, the branches from this iteration are used independently for the next iteration, which again starts with an analysis. The adaptation process terminates, as soon as all adaptation branches are bounded or are recognized as solving model after an analysis.
1.4. Contributions and Goals

The contributions of this thesis can be summarized in three parts. At first, an adaptation points model, including a cost model, is developed as an extension to DNI. Additionally, other models for SLAs generic analysis result and adaptation plan are generated. A second contribution of this work is a sophisticated adaptation process. Target-oriented adaptation for a short runtime, output of all cost-optimal solutions, and extensibility are the major aspects of the adaptation process. At third, the models as well as the adaptation process are implemented and published as Java application, the so called DNI adaptation framework.

The requirements to the described models, adaptation process and implementation could be formalized as goals in research questions (RQs). These research questions will be answered in the related sections in this thesis. The following goals with their corresponding research questions are identified.

**Goal 1:** Develop an adaptation point model for alternative parameters. The model should decorate or extend DNI.

- **RQ 1.1:** How could the alternative parameters be modeled?
- **RQ 1.2:** On which level can the adaptation point model extend DNI?

**Goal 2:** Develop an adaptation point model for network structure. The model should decorate or extend DNI (optional)

- **RQ 2.1:** How could the alternative configurations and the relation between them be modeled? (optional)
- **RQ 2.2:** On which level can the adaptation point model extend DNI? (optional)

**Goal 3:** Develop a cost model for adaptations. This model has to collaborate with the adaptation point model, described in Goal 1.

- **RQ 3.1:** Which different types of costs are meaningful?
- **RQ 3.2:** How are the costs calculated for transition from one alternative configuration into another?

**Goal 4:** Develop a component for SLA violation detection. The component has to detect SLA violations based on a previous analysis (simulation, or analytical method).

- **RQ 4.1:** How can SLAs be modeled?
- **RQ 4.2:** How can SLA violations be detected?

**Goal 5:** Develop a tactic to suggest adaptations.

- **RQ 5.1:** Is the tactic optimal?
- **RQ 5.2:** What is the runtime of the tactic and which parameters influence it? (optional)

**Goal 6:** Develop a process for the adaptation of DNI models.

- **RQ 6.1:** How can the workflow be modeled?
- **RQ 6.2:** How could the amount of model candidates be limited?
Goal 7: Develop a framework for the adaptation of DNI models.

RQ 7.1: How could the framework be separated into multiple modules?

RQ 7.2: Which interfaces should the modules provide and require?

Goal 8: Experimental evaluation of sample DNI model instances. (optional)

RQ 8.1: Does the suggested solutions improve the original configuration? (optional)

RQ 8.2: How can the suggested solutions be improved with regard to runtime? (optional)

1.5. Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces the background knowledge, which is required for the modeling and adaptation process. Chapter 3 deals with works, which are related to modeling or self-adaptive systems. In Chapter 4 the developed models for the adaptation process are described. The sophisticated, target-oriented adaptation process is explained in Chapter 5. Chapter 6 introduces the developed Java application, which implements the introduced models and adaptation process. In Chapter 7 the requirements for the adaptation process are assessed in a qualitative and quantitative evaluation. Chapter 8 concludes this thesis and shows how this work could be extended in future.
2. Background

Network optimization involves several technologies and models. This chapter deals with the foundations for a model-based network analysis and optimization and imparts the required background knowledge for an adaptation process.

The term Service Level Agreement (SLA), defining the performance agreements between network operator and customer, is introduced in Section 2.1. The main characteristics of performance modeling are described in Section 2.2. The concept of Model Driven Engineering (MDE) is briefly explained in Section 2.3. Section 2.4 depicts, how simulation techniques can be used for performance prediction. Queueing Petri Net (QPN) and SimQPN as one simulation tool, are introduced in Section 2.5. Section 2.6 deals with the major principles of Software Defined Networking (SDN). The Descartes Network Infrastructures Modeling (DNI) language, which is the basis for the model-based approach, and is used for network and traffic modeling, is described in Section 2.7. The key features of the Branch and Bound (BnB) algorithm are explained in Section 2.8. Section 2.9 introduces the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) adaptation control loop, which is a common approach for self-adaptive computing systems.

2.1. Service Level Agreement

It is an important aspect when using IT services to specify the quality expectations against the service provider. These specifications of Quality of Service (QoS) are negotiated as a contract, commonly referred to as Service Level Agreement (SLA) between the service provider and the customer [4]. The service provider is obligated to achieve the service promises, as - usually - the customer pays for the subscribed service [5]. SLAs between organizations are used in different areas of IT services like housing, communication services, and web services. Especially for emerging cloud services, the specification of SLAs are essential. Sometimes SLAs are also defined for help desks and problem resolution services [4].

A SLA contains service level attributes and Service Level Objectives (SLOs). The SLOs define the quality objectives, which every involved partner is obligated to provide [6]. Metrics are required to measure the compliance of agreed objectives. The used metrics strongly depend on the context of the contract. Some example metrics from online services are response time, throughput, and latency.
2. Background

2.2. Performance Modeling

Several QoS factors like availability, reliability, serviceability, security, and performance are crucial for the evaluation of computer systems and distributed component-based systems [7, 8]. Beside the availability of the provided services, it is also important that the services fulfill the performance expectations in forms of throughput, utilization and response time, which are usually specified in the SLAs for each customer. For example, a video conference system has to meet several high expectations related to bandwidth or delay, and it is not sufficient for the video conference provider to only ensure the availability of the assured service. In contrast, other services like a SMTP server for E-Mail delivery, typically has not so high demands on latency performance metrics. These performance factors are important throughout the lifecycle of a system, including design phase, development, configuration and maintenance. The performance characteristic should be analyzed to avoid pitfalls caused of unmet expectations on QoS. Performance engineering means to estimate the level of the expected performance of a system and provides configuration recommendations for the optimal performance level [8].

To measure performance metrics on a system, often the availability of the investigated system is required. This is not always possible, like for example in the design and development phase. In that case, either an educated guess or a performance model can be used to estimate the performance of a system [7]. Often the interactions within or between the involved systems are so complex that some forms of modeling are necessary. Performance modeling is emerging throughout the different phases of software engineering life-cycles [8]. Beside the performance prediction, another benefit of performance modeling is the insight into the structure and the behavior of the investigated system [7]. Additionally, performance models allow to estimate the performance considering an expected workload, maybe provided by a forecaster [3]. Often it is challenging to build models which accurately capture the different aspects of system behavior, especially when the model-based approach is applied to large and complex real-world systems [8].

Mathematical approaches or simulation techniques are well known methodologies to estimate the expected performance of models. Mathematical analyses benefit in calculation time, but often the given models are too complex or no analytical method exist, so that mathematical approaches cannot be used. In such cases, the performance models can be simulated through a simulation framework to estimate the expected performance.

2.3. Model Driven Engineering

The software development process has evolved over the time. While the first programs have been hard-wired computers, they emerged from stored-program-machines and assembly languages to high-level programming languages [9]. On the one side, higher levels of abstractions facilitate the development of complex software, but on the other side, the requirements and the complexity of software also increased over the time. Therefore, software development is still a complex, difficult, time consuming, money consuming, and error-prone task. The model-driven software development enables programmers to increase their productivity and decrease the cost of software construction. Additionally, the model-based approach also improves the software quality in form of better software reusability and increased maintainability.

The form of a model is not fixed. The commonly used models are defined in graphical, mathematical, or textual representation [9]. A graphical definition usually uses lines, shapes, symbols and text descriptions. An example for a mathematical model is $A = \pi r^2$ for a circle. Also text descriptions can be used to represent a model. Independently from the used representation, all the forms have in common to represent some aspects of a
system \cite{10}. Such a model abstracts some aspects of an underlying system and facilitates the modeler to focus on specific aspects without considering the complexity of the full system. The model-based approach lets software developers build their products much closer to the application domain. One example for a widely used modeling language is Unified Modeling Language (UML). UML was standardized by the Object Management Group (OMG) and describes all kinds of object-oriented software artifacts \cite{11}.

UML as a modeling language allows the instantiation of objects from a defined model and follows well defined specifications. For UML this specification is called UML metamodel. This meta-model language is defined by a definition language, the Meta-Object Facility (MOF). The Model Driven Architecture (MDA) is a software design approach, launched in 2001 by OMG for the development of software systems. MDA defines a modeling stack consisting of four different layers, depicted in Figure 2.1 \cite{9}. The lowest layer, the M0 data instance layer, contains instantiated objects, records, data, and related artifacts of the real world. The M1 layer, the user model layer, contains class diagrams, statecharts and other such artifacts. The M2 meta-model layer describes concepts for modeling languages like the UML meta-model. The M3 meta-meta-model defines concepts, which are used in the meta-model. The Meta-Object Facility (MOF) is the meta-meta-model for UML for example. The meta-meta-model can be compared to the self-defined, extended Backus Normal Form (BNF) formalism.

![MDA modeling and meta-modeling layers](image)

The Eclipse Modeling Framework (EMF) \cite{http://www.eclipse.org/modeling/emf/} is a modeling framework and code generating facility, based on Eclipse. EMF provides tools and runtime support to generate a set of Java classes for described models. Viewers, command-based editing of the model, and a basic editor are also provided by EMF. Annotated Java, UML, and XML documents can be used so specify models. Ecore \cite{https://wiki.eclipse.org/Ecore} is a meta-model language to describe models. In particular, Ecore is also its own meta-model. That means that Ecore is defined in terms of itself. Xcore \cite{https://wiki.eclipse.org/Xcore} is an extended concrete textual syntax for Ecore and enables in combination with Xbase \cite{https://wiki.eclipse.org/Xbase} the transformation into a full programming language.

---

\cite{10} \cite{9} \cite{http://www.eclipse.org/modeling/emf/} \cite{https://wiki.eclipse.org/Ecore} \cite{https://wiki.eclipse.org/Xcore}
2.4. Simulation Techniques

Performance evaluation of a system as well as its functionality, efficiency and capacity is a typical process in system development and maintenance process. Often models are used to investigate complex systems of the real world. In traffic theory, models facilitate a quantitative and qualitative description of events and assessment of system responses. This enables to determine if the expectations off a system are fulfilled. The following well known methods can be used to investigate the performance of a system [12]. They differ in abstraction level, granularity of result, effort, availability of hardware and software, the capabilities of the system to be investigated, and in computing time for the results. The methods in relation to the abstraction level of the system are depicted in Figure 2.2.

**Performance measurement.** A performance measurement can be performed on an already deployed system. Performance and functionality are investigated by measurement methods under real-world usage profiles. Additionally, the impacts from and to other connected systems are considered.

**Hardware simulation.** Hardware simulations run on a prototype of the system, which should be investigated. Using this method requires the availability of the used hardware, e.g., in form of a testbed. The load is typically generated by hardware simulation load generators.

**Simulation.** The simulation approach if often used in the concept and development phase, as it is independent of the availability of the hardware. Simulations are executed based on detailed system or abstract traffic models. This enables simulations on different abstraction levels. Simulation techniques are well suited for complex models, for which exact mathematical analysis are not possible and no reasonable approximate analysis techniques exist [7].

**Analytical approach.** Analytical approaches work on higher abstraction layers and provide exact or approximate results after typically short calculation times. Using analytical methods require the availability of analytical calculation methods for the given system, which do not exist in any case.

![Figure 2.2: Methods for performance assessment [12].](image)

Using simulation techniques provides several benefits. A simulation model describes the system structure, and is often easy to understand and to communicate. Animations and graphical representations facilitate demonstration, verification and debugging. Global knowledge is accessible at every time. Additional measurements and statistics can be added to the simuler at a later stage. But using a simulation technique also involves
some disadvantages and risks. Creating a simulation may require high effort. Possible pitfalls are programming errors, logical model errors, numerical measurement errors, random independence of results, and bad input models [13, 7]. Additionally, simulations use simplified assumptions and hypotheses, which have to be considered when the results are analyzed.

Simulation models can be classified into different dimensions [13]. A static model investigates the system at an arbitrary point in time and especially the time has no impact on the result. A dynamic model represents the behavior of a system during a certain time period. A simulation with a deterministic model has no random dependent components, while in a stochastic model, random events might have an effect on the system behavior. In a continuous model the system states change continuously, in contrast to a discrete model, where the states only change on discrete points of time.

A Discrete Event Simulation (DES) is one type of discrete simulations. A sequence of events is executed, whereby new events can be inserted into the so called event chain dynamically at runtime. Usually every event causes a change in the system state. As between consecutive events no change occurs, the simulation clock can jump directly to the next event. This enables to simulate real-world long time periods in a short simulation time. A DES consist of multiple components [13]. The system state is temporal mutable and is represented through one or multiple variables. The previously mentioned simulation clock contains the current system time. An event is associated to a discrete simulation point of time and executes the event routine, to possibly effect the system. The events are managed in an event chain. Statistical counters store statistical information of the behavior of the system. Library routines for example contain random number generators. The main program initializes the simulation, supervises and process the event chain and controls the simulation time. The simulation results are prepared and delivered through a report generator.

2.5. Queueing Petri Nets (QPNs)

Petri nets (PNs) describe discrete, usually distributed, systems and are used in system modeling. This section briefly introduces the central idea of Petri nets, illustrates the advantages of Queueing Petri Nets (QPNs) and introduces Simulator for Queueing Petri Nets (SimQPN), as an discrete event simulation framework for QPNs. A more detailed explanation of Petri nets and QPN especially their formal definition, can be found in [14, 15, 16].

Petri nets consist of a bipartite directed graph composed of places, transitions and arcs. An arc connects a place with a transition or vice versa, but never two of the same type. Input places are places from which an arc points to a transition and output places are places, which are target of an arc. In graphical representation Petri nets contain a discrete number of tokens. If all preconditions of a transition are fulfilled, i.e., a required number of tokens are in its input places, the transition fires. The tokens are consumed then and new tokens are created in the output place.

There have been several extensions to Petri nets like Colored Petri Nets (CPNs). While in classical Petri nets the tokens cannot be distinguished, the tokens in a CPN are from a specific type (“color”). Stochastic Petri Nets (SPNs) add an exponentially distributed firing delay to transitions. Two types of transitions, immediate and timed, are introduced by Generalized Stochastic Petri Nets (GSPNs). The ideas of two different transition types from GSPNs and colored tokens from CPN are combined in Colored Generalized Stochastic Petri Nets (CGSPNs).
Another enhanced extension to Petri nets are Queueing Petri Nets (QPNs), introduced by Bause in 1993 [15]. QPNs are the result of the combination of CGSPNs and the introduction of queues (queueing stations) to Petri nets. A queueing place, as depicted in Figure 2.3, consists of a queue and a depository for tokens, which are processed and have completed their service. Tokens from input transitions are inserted to the queue of an input place, according to the queue’s scheduling strategy. Output transitions of the place have no access to these tokens. A token is moved from the queue to the depository immediately after the token is processed, i.e., the service is completed. After moving the token to the depository, it is available for output transitions of the place. QPNs support two different queueing places. For timed queueing places the previously explained service completion delay applies, while immediate queueing places have no service time and can be used to model scheduling aspects.

![Figure 2.3.: A queueing place and its shorthand notation](image)

QPNs have a greater expressiveness than Petri nets, extended Petri nets and Stochastic Petri Nets (SPNs). Hardware as well as software aspects of the system behavior can be integrated into the same model via QPNs. QPNs are well suited to model hardware contention, scheduling strategies, simultaneous resource possession, synchronization, blocking and software contention. Modeling distributed e-business applications is one example for the usage of QPNs.

One aim of a model is to get analysis result, e.g., performance predictions, through an appropriate solver. Simulator for Queueing Petri Nets (SimQPN) is a discrete event simulator for QPN models, written in Java [16]. SimQPN leverages the knowledge of the structure and behavior of QPNs to solve models, even of realistic size and complexity, efficiently. General purpose simulations, which are not designed for QPNs, cannot process such models directly, and do not work as fast and efficient as specialized simulators. SimQPN provide different statistic outputs. The PlaceStats and QueueStats contain statistics collected during simulation. Statistics of multiple simulation runs can be found in the AggregatorStats. SimQPN supports different modes for data collection, which differ in detail granularity and simulation processing time. DML solvers and DNI solvers are two example applications, using SimQPN.
2.6. Software Defined Networking (SDN)

Traditional networks are build up on fixed topology with use case specific network elements. This approach hampers the adaptation of new protocols and the implementation of additional network functions. The capabilities of networks, e.g., regarding dynamic adaptation, is often limited by vendor specific proprietary firmware, running on the devices. Consequently, the most deployed networks are use case specific, for example for web services, for business applications or for [High-performance Computing] ([HPC]). Emerging technologies like cloud computing and network virtualization require new concepts for future networks. Networks, consisting of devices, functions and a topology should be programmable like software. Additionally, the limitations by the fixed functionality of network devices, implemented in silicon, should be removed. This facilitates fast adaptation and short release cycles of new features. To achieve this, the functionality from hardware is abstracted into more flexible and adaptable software, running on commodity hardware. This abstraction is one of the key features of [Software Defined Networking] ([SDN]). The following four basic principles are mandatory for classifying a technology as SDN [17]:

**Separation of Control and Data Plane** is one of the most significant principles in SDN. In traditional closed box network elements, the controlling logic and the forwarding plane are implemented in the same device. SDN extracts the intelligence of the devices into a separate control plane as shown in Figure 2.4.

**Logically Centralized Control** means to configure and manage the network by one controller instance with global network information. The actual controller setup can be distributed due to reliability aspects. Figure 2.4a depicts the control plane in a device in traditional networks, while Figure 2.4b shows the centralized control plane in SDN.

**Open Interfaces** are required for vendor independent data plane programming. For interoperability, this protocol should be standardized. This avoids vendor lock-in by vendor-specific configuration utilities and enables adoption of new packet handling paradigms without the need of a firmware upgrade. A SDN controller provides different interfaces for interaction with switches, other controllers, applications and traditional network devices. One of the most popular protocols for the communication between control plane and data plane is OpenFlow. This protocol is especially used to configure the forwarding behavior of the SDN enabled network elements.

**Programmability** of SDN is used by external software or applications, running on top of the network, to influence the network topology and the forwarding behavior of packets. The forwarding behavior can be programmed as a part of software with the advantage of flexibility and short release-cycles.

![Figure 2.4.: Control plane in traditional network devices and SDN](image)
2.7. Descartes Network Infrastructures Modeling (DNI)

Today’s data centers are faced with an increasing complexity. This complexity is not only caused by software architectures, middlewares, and resources but also technologies like network virtualization, network protocols, network services and configuration are accountable for increasing complexity. Customers’ expectations on QoSs are typically specified in SLAs which should not only be considered in design phase, but also at runtime to ensure the compliance with the agreements on deployed networks. If the requirements are not fulfilled, the network should be adapted to meet the expectations.

It is often difficult to predict the performance of possible adaptation actions on complex networks, especially as experimenting with different adaptation actions could cause service outages on productive running systems. The meta-modeling approach is well suited to predict the impact of changes on productive networks. Performance modeling provides a powerful mechanism to evaluate, predict and analyze the performance of modern data centers. Several existing coarse-grained analytical models are not useful for a meaningful performance prediction. Existing highly-detailed simulation models require a great duration for simulation and are therefore not suitable for runtime prediction.

The Descartes Network Infrastructures Modeling (DNI) language introduces a sophisticated meta-model for data-center networks, focusing on performance aspects [18]. It provides several solvers for a flexible performance prediction on runtime. DNI as a descriptive meta-modeling language for data center networks, supports traditional as well as SDN-based networks, with different modeling granularities. The generated model is well suited for what-if analysis without impact on the productive network. DNI does not separate between the computing and software context and can be used for end-to-end performance analysis. The system can be observed at runtime and a model can be build based on the captured behavior. The DNI network meta-model and its solvers are described in this section.

2.7.1. Network Modeling

The DNI meta-model consists of several entities. The entities, related to this work are introduced in this subsection. A more detailed description of all entities can be found at [18]. Firstly, the Dependency, used for numeric values of parameters is explained. At second the Entity, as a generic definition of entities is described, and afterwards the individual entities are introduced.

2.7.1.1. Dependency

Numeric values are used to describe performance parameters like throughput, packet processing time, or switching capacity. These values could be fixed numeric, e.g., 10 Mbps, or functions like exponentially distributed with a specific rate parameter $\lambda$. To support fixed values as well as functions, DNI introduces the Dependency interface. This interface is inherited by the Variable and the Function interface. The Variable interface is implemented in the ConstantLongVariable class for integer values, in the ConstantDoubleVariable for floating point numbers, and in the RandomVariable, containing a Cumulative Distribution Function (CDF). The Function interface is implemented by the DiscreteFunction, containing two double values, and the ExponentialFunction, containing a floating point constant $\lambda$.

In addition to the supported numeric value, also an DNIUnit can be attached to a dependency. The DNIUnit is an interface, implemented by the following three entities. The SpeedUnit contains a speed classifier for BytesPerSec, bitsPerSec, and packetsPerSec, and an optional prefix for kilo, mega, and giga. The TimeUnit contains a descriptor
for the granularity of time for seconds, microseconds and milliseconds. The \texttt{DataUnit}, as the third implementation of \texttt{DNIUnit}, contains only the previous introduced kilo, mega, and giga prefix. The \texttt{Dependency} interface and its inheritance structure is depicted in Figure \ref{fig:dependency-structure}. This structure enables a flexible definition for numeric values, including measurement units.

![Figure 2.5.: Dependency structure in DNI meta-model][18]

### 2.7.1.2. Entity

The \texttt{Entity} interface is implemented usually for physically and logical entities in \texttt{DNI}. The \texttt{Entity} interface itself implements the interfaces \texttt{NamedElement}, which adds a name and a description to every entity, and the \texttt{Identifier} interface, to assign an auto-generated or manually specified \texttt{UID}. The \texttt{Entity} class with its inheritance structure is illustrated in Figure \ref{fig:entity-structure}.

![Figure 2.6.: Entity structure in DNI meta-model][18]
2.7.1.3. Network Infrastructure

The NetworkInfrastructure is the root element of a DNI model instance. The root element contains three logical parts, NetworkStructure, NetworkTraffic, and NetworkConfiguration, as depicted in Figure 2.7. The NetworkStructure describes the physical structure of the network, the NetworkTraffic is used for the definition of the workload on the network, and the NetworkConfiguration contains the specified configuration of the network.

![Diagram of Network Infrastructure](image)

Figure 2.7.: Network infrastructure as root element in DNI meta-model.

2.7.1.4. Network Structure

The NetworkStructure entity is used to describe the Nodes in the network and the Links, connecting them. It can be considered as an undirected graph, where the nodes are the vertices and edges. The NetworkStructure with its contained Node and Link entities is depicted in Figure 2.8.

![Diagram of Network Structure](image)

Figure 2.8.: Network structure of DNI meta-model.

2.7.1.5. Node

A Node in DNI describes entities producing, forwarding, or consuming data. Nodes could be an end node, like a server, an intermediate node, like a switch, or both. The structure of Node and the related entities is shown in Figure 2.8. Acting as end and intermediate device both is a special use case for SDN devices, as they forward traffic like an intermediate device and consume and produce traffic for the used SDN control protocol such
4.1 Modeling Classical Networks

Network Interface

DNI Nodes are connected using NetworkInterfaces and Links. The performance of a Link is usually described by the parameters forwardingLatency, forwardingBandwidthBPS, and switchingCapacityPPS. A Link is virtual if it connects NetworkInterfaces that belong to at least one SDN node.

Depending on the type of node, end or intermediate, the entity also contains a performance descriptor. An end node contains an EndPerformance, with a protocolRef that specifies the protocols used. The node also contains performance descriptions for softwareSwitchingPerformance and hardwareSwitchingPerformance, which have the same parameters as an intermediate node performance descriptor. Additionally, a SDN node contains a reference to a SDN controller and a CommunicatingApplication as openFlowEndPoint.

2.7.1.6. Network Interface

Network Interfaces are contained in nodes, to get connected with other nodes. A referenced ProtocolStack, as illustrated in Figure 2.9, describes the used protocols defining the additional payload which is attached or detached to the payload while encapsulation or decapsulation of the packet. A NetworkInterface also contains a PerformanceNetworkInterface as performance descriptor. This performance is specified by the three parameters packetProcessingTime and interfaceThroughput.

![Network interface structure in DNI meta-model](image)

2.7.1.7. Link

Links are used to connect network interfaces of two different nodes, as depicted in Figure 2.8. A Link could either be a physical or virtual medium over which the traffic is transferred. To describe the performance of a Link, a contained PerformanceNetworkInterface specifies one parameter for packetProcessingTime and one parameter for interfaceThroughput.

2.7.1.8. Network Traffic

The NetworkTraffic is the second entity in the NetworkInfrastructure root element, which is illustrated in Figure 2.10 in detail. The most traffic is generated by deployed
applications, other traffic could be generated by control flows. The traffic is modeled in Flows, and each Flow has one source and possibly multiple destinations, which are referenced by CommunicationApplications. Currently [DNI] only supports open workloads. Each workload contains a list of actions like wait, transmit, start, stop, loop branch or sequence. Generating one message of size 2.47 MB on application A and transmitting it to application B is an example definition for an example network traffic. The workload represents a time span, while the network structure as well as the network configuration represent a snapshot of the system state at single moment of time.

![Network traffic structure in DNI meta-model](image)

**2.7.1.9. Network Configuration**

NetworkConfiguration is the third children of the DNI root element NetworkInfrastructure. The network configuration contains routes for the packets and specifications of the used protocols. The NetworkConfiguration represents a snapshot of the current configuration. Dynamic changes through a period of time, i.e., dynamic routing protocols, are currently not supported by [DNI].

Routes for packets traversing the network are specified flow-based on each hop via a so called Direction, contained in a RoutesRepository. A Direction references the three entities Node, NetworkInterface and Flow. This definition can be supplemented by an optional priority, to define different routes for a flow, to model load balancing mechanisms. The approach of route specification via Direction allows rules like “On node n1 the flow f1 should be forwarded over the network interface eth0 with the probability of 0.5”. Such rules have to be installed over the complete path on which the packets should traverse the network. As the destination for a packet, specified as destination CommunicationApplication is included in the Flow, it has not to be defined in the Direction entity. The diagram of the route specification via Directions is depicted in Figure 2.11.

In addition to the RouteRepository, the NetworkConfiguration also specifies the protocols and protocol stacks in a ProtocolsRepository. Each NetworkProtocol, like Ethernet, IP or TCP, has to be defined with its Maximum Transmission Unit (MTU).
and header length. Beside the size of the packet header, the header length must also contain the generated overhead by packet retransmissions. The NetworkProtocols can be layered to a ProtocolStack, to model different protocol stacks like Ethernet/IP/TCP. This modeling approach enables flexible and especially exchangeable protocols which could be referenced by NetworkInterfaces. For example the Fiber Distributed Data Interface (FDDI) protocol can be used on a connection between two network interfaces, by replacing the Layer 2 Ethernet protocol through FDDI. The relation between protocols and the protocol stack is shown in Figure 2.11.

Figure 2.11.: Network configuration structure in DNI meta-model [18].

Beside the RoutesRepository and the ProtocolsRepository the NetworkConfiguration also contains a set of SdnFlowRules. They specify per node and per flow the probability if the packet should be processed in the hardware or the software table of the switch, or if the packet should be sent to the SDN controller.

2.7.2. MiniDNI

The DNI meta-model, described in the previous sections, facilitates a fine-grained definition of data center networks. But sometimes not all the required data are available to build such a detailed network model. A reduced version of DNI the so called Mini, Descartes Network Infrastructures Modeling (MiniDNI), abstracts selected parts of the meta-model and enables a lightweight definition of networks. MiniDNI especially requires less input data. As MiniDNI is out of scope of this thesis, it is not discussed here more in detail. Further information about MiniDNI can be found in [18].

2.7.3. Transformations and Solving

To evaluate the runtime performance of a given model instance, the model has to be analyzed. This is done by solvers, which include simulation approaches as well as analytical methods. Currently, only simulative solvers for DNI are available. These discrete event simulations enable fine-grained performance modeling, up to protocol-level details and traffic at the packet level. Additionally, simulation models capture the internal structure of a modeled system explicitly. In contrast, analytical performance models are often defined at a high level of abstraction. For the completion of the model instances, DNI provides model extraction methods, to prefill the DNI model automatically with the network traffic data.
DNI is a **descriptive model**, which organizes information about a domain, but does not provide information for the performance prediction process, which are required for solvers. Therefore DNI models have to be transformed into **predictive models** that can be solved. A model transformation is defined as follows: “A transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.” [19]. Model transformations are developed for a pair of meta-model languages, consume an arbitrary instance of the meta-model and produce an equivalent representation in an output meta-model language.

DNI provides transformations for all supported analyzer, so that all predictive output models can be solved directly by a proper solver. Currently, DNI supports transformations for the following solvers: SimQPN, OMNeT++ INET, OMNeT++ generic, LINE, LQNS and LQSIM [20, 21]. Not all of these solvers support all features of DNI, more details about this can be found in [18]. The model transformations in DNI are structured in three phases, to solve a model. At first, a descriptive DNI model from an existing network infrastructure is built. Secondly, an predictive model is generated through model transformation. At third, the predictive model is analyzed by a compatible solver to get the performance metric values as result of the executed analysis.

Multiple solvers are supported by DNI and there are some criteria for selecting an appropriate one. The solvers differ in runtime for the simulation, so a first criterion is the time frame, in which the performance results are needed. The required solving time depends on the size of the input model, scalability of the solver, performance of the transformation program, required accuracy, and size of the confidence interval for a given metric. At second, the solvers provide different accuracy of the performance prediction. A selected solver has to support the required performance metrics and the necessary resolution of the results. Thirdly, model-relevant factors like model size and the supported features like SDN, custom protocols, or load balancing should be included in the decision process for a suitable solver.

### 2.7.4. Integration with DML

**Descartes Modeling Language** (DML) is a sister language of DNI with focus on self-aware performance and resource management for modern dynamic IT systems, infrastructures and services [22]. Both meta-models, DNI and DML, represent complementary parts of data centers. DNI is used to model complex network structures and describes physical and virtual nodes and intermediate nodes, and the connections between them. DML represents the computing and software parts and uses a resource landscape to model computing nodes, storage nodes and virtual containers. Both language support different solvers including the required model transformations, differing in granularity of result and the required solving time.

### 2.8. Branch and Bound Algorithm

Combinatorial optimization problems are very frequent in theoretical computer science as well as in real-world. Problems of these type have a finite, but usually very large number of feasible solutions [23]. The **traveling salesman problem** is one example for such an optimization problem, classified as \( \mathcal{NP} \)-hard. The **Branch and Bound** (BnB) **algorithm** is a paradigm to solve \( \mathcal{NP} \)-hard discrete combinatorial optimization problems,
introduced by A. H. Land and A. G. Doig [24]. The pseudocode of the algorithm has to be customized for a specific problem. The [Branch and Bound] algorithm uses a bound function to only investigate parts of the solution space, possibly containing an optimal solution.

The [Branch and Bound (BnB)] algorithm consists of three main components. The first component, the node selection, chooses the next node, which should be explored. There exist different methods to build a initial state, the so called root node. Which next node is selected from the unexplored subspace pool depends on the used search strategy.

The best-first search is widely used for branch and bound, but [Breadth First Search (BFS)] as well as [Depth First Search (DFS)] are alternatives in some scenarios. The second component is the bounding function, as one key component of the branch and bound algorithm. If the calculated value for a node is between the lower and the upper bound, the node is explored. Selecting the right bounding function is often a trade off between quality and calculation time. As branch and bound is typically used for NP-hard problems, the bounding function is sometimes complex too. The third component is branching. The rules for branching are often the addition of constraints in form of assigning values to variables. The branch component dynamically builds a search tree by exploding the next best node. The search terminates if all parts of the solution space are explored. The solution, marked as current best, is the optimal solution.

### 2.9. MAPE-K Adaptation Control Loop

Self-adaptation is a part of autonomic computing and describes the capabilities of an IT system, to adapt itself, with minimal human intervention. Design decisions are moved towards runtime and an individual system controls dynamically its behavior with knowledge of its state and environment [25]. Feedback loops are common control mechanisms, which iteratively adapt a system and involve observations from their previous adaptations in their decision process. In software engineering, [Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K)] is a commonly used feedback control loop for self-adaptive systems, which is described in this section.

A [MAPE-K] control loop is often implemented as an autonomic manager, which is a component managing other software and hardware components [26]. The autonomic manager uses sensors to get information about the state and state transitions of the managed resource, i.e., the managed system. Effectors are interfaces to apply changes on a managed systems. The interaction between an autonomic manager the sensor and effector interfaces and the managed system is depicted in Figure 2.12. In practical, multiple separate control loops are often involved to adapt a complex system. The responsibilities of a generic control loop are collecting information about the current system state, analyze these information to diagnose performance problems or to detect failures, decide how to resolve actual problems, plan the adaptation and finally execute them. These steps are classified in the four phases of monitor, analyze, plan, and execute. These phases are often supplemented by a knowledge component, as illustrated in Figure 2.12. The four phases and the additional knowledge component are described in the following more in detail [26]:

**Monitor.** The monitor phase is responsible collect and aggregate information from the managed resource. This includes topology information, metrics, and configuration settings. The data are usually requested through sensor interfaces on the managed system. Additionally, the information can be filtered and correlated to a format, which can be analyzed.
The acronym MAPE-K reflects the five main constituent phases of autonomic loops, i.e., Monitor, Analyze, Plan, Execute, and Knowledge, as depicted in Figure 2.4. Basically, the Monitor phase collects information from the sensors provided by the managed artifacts and its context. The Analyze phase uses the data of the Monitor phase to assess the situation and determine any anomalies or problems. The Plan phase generates an adaptation plan to solve a detected problem. The Execute phase finally applies the generated adaptation plan on the actual system. A cross-cutting aspect shared among all phases of the loop is the Knowledge about the system and its context, capturing aspects like the software architecture, execution environment, and hardware infrastructure on which the system is running. The knowledge may also explicitly capture the operational goals of the system, e.g., the target QoS level the managed system should provide. The representation of the knowledge can take any form, e.g., a performance model describing the performance behavior of the system. As this thesis follows a model-based approach, we are particularly interested in the different types of models that can be applied to represent the knowledge about the system. Such concepts will be introduced in the following Section 2.1.3.

The software engineering community uses a similar feedback loop concept, distinguishing the four phases Collect, Analyze, Decide, and Act (Cheng et al., 2009). Conceptually, the behavior of these phases is similar to the phases in the MAPE-K loop, however, this concept does not explicitly consider the Knowledge part.

More details about the use of feedback loops in self-adaptive systems, such as the use of multiple, multi-level, positive, or negative feedback loops, are given by Brun et al. (2009).

Figure 2.12.: Reference architecture of the autonomic control loop implemented by autonomic managers [27, 28].

**Analyze.** In the analyze phase the observed data are analyzed and the autonomic manager learns about the environment. For a proactive approach, future situations can be predicted using a forecaster, based on existing time series from the knowledge component and current observations. The analyze phase decides, if all goals are fulfilled and generates a change request to initiate an adaptation.

**Plan.** The planning phase decides what should be adapted, to achieve the specified goals and objectives for the managed system. The planning mechanism can be implemented as a single command, an algorithm or a complex workflow. A change plan is created, which includes all necessary changes in an abstract form.

**Execute.** The execute phase performs and controls the execution of the previously scheduled changes. The state of one or more managed resources are modified, according to the change plan. The execute phase should additionally update the knowledge component by injecting the executed adaptation actions.

**Knowledge.** The knowledge component extends the knowledge capabilities of an autonomic manager and is not an iteratively passed phase, like the previous fourth ones. This component should be accessible for read and write operations, for and from the four phases. It could be implemented as a registry, dictionary, database, or repository. The knowledge component could contain topology information, historical logs, metrics, symptoms, policies, change request and change plans.
3. Related Work

This chapter introduces model-based optimization approaches, simulation frameworks, and other works, related to this thesis. Section 3.1 describes the S/T/A adaptation approach, which is used for autonomic performance-aware resource management in dynamic service infrastructures. In Section 3.2, PerOpteryx as a similar approach is introduced, which differs especially in the used model and the time of appliance. While S/T/A focus on optimization at run-time, PerOpteryx is intended to be used on design-time. Section 3.3 describes NS-3 Network Simulator, a sophisticated network simulator with extensive analysis results. CloudSim, a toolkit for modeling and simulation of cloud environments is outlined in Section 3.4. In Section 3.5, REWIRE, an optimization-based framework for data center network design is introduced. REWIRE focus on optimization of wiring challenges. Section 3.6 deals with requirements and assessment of languages and frameworks for adaptation models.

3.1. S/T/A Adaptation Approach

Beside DNI for modeling network infrastructures, computing resources can be modeled using the Descartes Modeling Language (DML) [22]. Several analytical and simulative methods are provided for DML to execute performance predictions and what-if analysis for DML model instances including a defined workload. While the analytical methods enable detection of bottlenecks in the system environment, native DML does not support to suggest adaptations which will remedy detected problems. Nikolaus Huber et al. proposed an adaptation framework for DML to dynamically allocate resources and suggest changes to the system environment at runtime, if any problems have been identified [27]. The approach of this adaptation framework is introduced in Section 3.1.1. The adaptation points meta-model, an extension to DML is used to define the degrees of freedom, i.e., the adaptable parameters on a model instance. The adaptation points meta-model is briefly explained in Section 3.1.2. The adaptation process meta-model, described in Section 3.1.3, enables a model-based specification of the adaptation process. A conclusion of the S/T/A adaptation approach is given in Section 3.1.4.

3.1.1. Approach

Scope of the Strategies/Tactics/Actions (S/T/A) adaptation framework is a continuous observation and adaptation during runtime through a generic adaptation control loop concept. A schematic representation of the model-based adaptation process is depicted
in Figure 3.1. In the first step, problems on the current configuration are anticipated and detected under a given workload, which is also part of the DML model instance. If any problems are found, the performance model is adapted and the violations are tried to be remedied. After the adaptation of the performance model, the impact of the adaptation is again predicted through an analysis. If the problems is solved, the suggested adaptation is a solution and can be applied to the real system. Otherwise, the adaptation process is repeated until the operational goals are fulfilled.

Figure 3.1.: Schematic representation of the model-based adaptation process and the involved artifacts [22].

The adaptation process uses a MAPE-K feedback control loop, as introduced by Brun et al., to build a self-adaptive system as described in Section 2.9 [25]. How the four phases monitor, analyze, plan and execute and the knowledge component are mapped to the S/T/A approach, is depicted in Figure 3.2

Figure 3.2.: Mapping between MAPE-K adaptation control loop and the model-based S/T/A adaptation process [22].

In the monitor phase, the system configuration, Quality of Service (QoS) metrics, architectural data and (changed) SLAs are observed. The monitored data are persisted in the knowledge base. The analyze phase detects and anticipates problems like SLA violations or inefficient resource usage. The system state is analyzed in this phase to identify the cause of a problem. For analysis a reactive approach based on obtained monitoring data, as well as a proactive approach, based on forecasted workloads can be used. In the plan phase, feasible solutions for the problems, identified in the analyze phase, are searched. This phase is executed iteratively in two steps. In the first step, a
new system configuration is generated on model level through an adaptation strategy, chosen in the analysis. The possible changes are defined in the adaptation points model, which is explained in Section 3.1.2 more in detail. In the second step, the impact of the adaptation is analyzed, using online performance techniques. If the problem is solved, the process will continue with the execution phase, otherwise the plan phase will be rerun iteratively. The execute phase applies the generated adaptation on the system environment. For this, the previous, on the model applied, adaptation actions will be replayed on the real system, using reconfiguration interfaces. The knowledge component stores QoS metrics, monitoring data, topology information, configuration settings, and the DML model instance, and the adaptation points.

3.1.2. Adaptation Points Meta-Model

The adaptation points meta-model specifies the adaptable parameters for an architecture-level performance model by annotating the existing entities at design time. There are two types of adaptation points. One type is the adaptation of attribute values like the passive resource capacity or number of vCPUs assigned to a VM. The other type is changing the number of instances of a model entity, like starting or reducing the number of VMs. The adaptable entities have to be annotated manually and cannot be automatically detected, because some entities cannot be adapted through technical limitations or an adaptation is unintended by operators. The landscape meta-model as well as the application architecture meta-model require adaptation points. To avoid introducing adaptation points to both meta-models, the alternative configurations are implemented in a separate meta-model. The usage profile does not support the annotation with adaptation points, because the workload cannot be controlled.

3.1.3. Adaptation Process Meta-Model

In addition to the adaptation point meta-model, which defines what can be adapted, the adaptation process meta-model specifies how an alternative configuration is selected. The adaptation point meta-model consists of the three main elements, strategies (S), tactics (T), and actions (A), which are explained in the following more in detail.

Strategy

Strategies represent the highest level of abstraction. They work on the logical goal-oriented aspect of the adaptation process and describe the possible ways to achieve the objectives. If an objective is violated during system operation, the associated strategy is triggered through an event. On strategies, metrics are compared to a threshold, e.g., response time. A strategy specifies one or multiple tactics, which are selected by a weighting function, considering observations of the current system state.

Tactic

A tactic is more system specific and pursues a short-term goal for a certain purpose, e.g., scaling-up a specific resource, by executing one or multiple adaptation actions. Tactics execute a specified sequence of actions, without explicitly considering the effect. These actions are defined in the adaptation plan by a deterministic control flow, consisting of the elements start, stop, loop, and branch. Tactics work conservative and optimize values iteratively, step by step. This means that no more than one resource unit is removed at a time, for example. A selection mechanism, based on the impact of the technical aspect of the current tactic, can trigger a subsequent tactic.
Related Work

**Action**

Actions are atomic elements at the lowest level of the adaptation process hierarchy. They are a technical part of the adaptation process and encapsulate system specific details. Actions implement actual adaptation operation on the system model or on the real system, but they do not describe how the operation is executed. The implementation of an action is part of the adaptation framework. Actions are specified by directions like increase, decrease, or migrate to add or remove resources.

**3.1.4. Conclusion**

The S/T/A adaptation process of DML works on a generic level. The adaptation bases on the adaptation process meta-model, which allows to define an adaptation process on model level by predefined control flow elements. This model-based approach facilitates extensibility and a clear separation between implementation and logic of the adaptation process.

However, this approach limits the expressiveness of the adaptation process to the predefined control flow elements. Some optimization algorithms, e.g., novel insights from graph theory, would not be able to be expressed by the current meta-model and could therefore not be included into the adaptation framework. The adaptation process contains no domain knowledge about the adapted landscape and therefore the whole complex adaptation logic has to be modeled by a system engineer.

**3.2. PerOpteryx - Automated Improvement of Software Architecture Models**

For software architects it is a time consuming, error-prone and difficult task, to find an optimal configuration for a software architecture manually. Often there is not optimal solution related to all quality attributes, instead of this, it is a trade-off between multiple quality attributes. *PerOpteryx* introduces a novel multi-objective evolutionary optimization framework for software architectures [29]. It operates on a software architecture, modeled with *Palladio Component Model* (PCM), and focuses on optimization at design-time. The alternative parameters and parameter ranges like processing rates, hardware costs, and component costs for software, are annotated in the PCM model.

PerOpteryx uses a hybrid optimization approach, based on rules, containing domain specific knowledge, and a meta-heuristic for finding alternative configurations. This meta-heuristic applies a general-purpose, problem-independent optimization strategy. *Non-dominated Sorting Genetic Algorithm-II* (NSGA-II) is used as multi-objective evolutionary algorithm, which is well suited for multi-objective combinatorial problems, as found in the software architecture design with attributes, influencing each other [30].

Alternative configurations are evaluated, i.e., tested, by an analysis. Currently a *Layered Queueing Networks* (LQN) solver is used, which is connected through a transformation adapter from PCM to LQN. PerOpteryx can be extended by additional analysis and simulation tools. Scope of the optimization process is to provide multiple solutions, the Pareto-optimal candidates, to software architects. As the candidates possibly contain conflicting quality attributes, there would not be a single solution.

The optimization process could be summarized as follows. PerOpteryx generates randomly several model instances as initial input. These candidate models are analyzed through a LQN simulation. An investigation of the analysis result selects the most Pareto-optimal candidate models for further optimization and throws the other randomly generated models away. Through crossover mutation and tactics, encoding design
knowledge and rules of thumb, new candidate models are generated, which pass the optimization process iteratively until the Pareto-optimal candidate models, fulfilling the QoS metric, are found.

PerOpterys pursues an optimization of software architectures on design-time, based on PCM. Although it is also an optimization framework, there are some obstacles to using it for network optimization. The network adaptations should be discovered on runtime, while PerOpteryx focus on design-time. Furthermore PerOpteryx bases on PCM with focus on software architectures and PCM does not provide the desired granularity for network modeling.

3.3. NS-3 Network Simulator

The NS-3 Network Simulator in a network simulation environment, classified as Discrete Event Simulation (DES) [31, 32]. Focus of NS-3 are research and educational experiments. The core of NS-3 can be extended and reused by several applications. NS-3 provides support to generate a complete simulation workflow, from configuration to trace collection and analysis. Different network protocols, IP as well as non-IP, are supported. Simulations including models for Wi-Fi, Worldwide Interoperability for Microwave Access (WiMAX) or Long Term Evolution (LTE) can be executed. Furthermore NS-3 can be used with static and dynamic routing protocols.

One focus of NS-3 is the interaction with real networks. Packets, generated by NS-3, can be emitted and received on real network devices. Additionally, virtual machines can be connected through NS-3 and effects can be added to links between them. Real world applications can be used in interaction with NS-3.

NS-3 provides a sophisticated network simulation framework. The analysis result could be used for bottleneck detection. However, NS-3 like other simulation frameworks, e.g., OMNeT++, does not support the definition of degrees of freedom and does not contain any adaptation logic.

3.4. CloudSim - Toolkit for Cloud Modeling and Simulation

CloudSim is a toolkit for modeling and simulation of cloud computing systems and application provisioning environments [33]. In clouds, virtualized services as IT infrastructures and applications are provided on the fly to end-users. Theses services are usually offered by an usage-based payment model. The Quality of Service (QoS) expectations are agreed in a Service Level Agreement (SLA) between the cloud provider and end-user.

Cloud providers force to comply with the assured performance agreements, though it is difficult to evaluate the performance of a cloud. It is challenging to run benchmark experiments in a repeatable, dependable, and scalable real-world cloud environment, as the cloud is shared by other users [33]. Additionally, an application service developer has usually limited control over the cloud environment.

CloudSim provides modeling the system and behavior of cloud system components like data centers and virtual machines. The modeled system can be simulated to get performance predictions about the current configuration. The simulative approach enables testing, tuning and experimenting on an experimental environment, with transferable results from the simulation to the cloud ins real world.

CloudSim focuses on modeling and simulation of VMs allocation, applications, provisioning, and cloud market. Several more enhanced features like power consumption are also supported. But unfortunately currently no network entities like routers or
switches can be modeled with CloudSim. The network topology is stored in as latency matrix of nodes. This approach is unsuitable for a network analysis and adaptation, which requires fine-grained details of the network infrastructure.

3.5. REWIRE - Optimization-based Framework for Data Center Networks

Designing data center networks is a challenging task. This is true for new data center networks as well as for expanding existing networks. REWIRE, an optimization-based framework for data center network design, facilitates finding appropriate network setups. REWIRE uses a local search-based algorithm, to find a network with maximal bisection bandwidth and minimal end-to-end latency [34]. User defined constraints will be considered and the predicted costs for the network are accurately modeled.

It has been shown that overloaded links in data center networks often constrain server utilization [35]. Additionally, sometimes network links are unused, which could be utilized through a more sophisticated traffic routing. Through an optimization, the network performance should be maximized, which includes bisection bandwidth, end-to-end latency, and reliability. In parallel, the costs should be minimized and a large number of constraints have to be considered. The REWIRE algorithm addresses this problem for designing new, upgraded, or expanded data center networks using a local search approach.

REWIRE suggests an optimal wiring of a given set of switches. An optimization, in form of adding or replacing hardware, is currently not supported through an automatic approach. REWIRE therefore covers only the part of optimizing the wiring in data center networks, while other optimization options are left untouched. However, the REWIRE algorithm could be included into a network adaptation framework, to suggest improvements of data center network wiring.

3.6. Requirements and Assessment of Languages and Frameworks for Adaptation Models

Several self-adaptive software systems use a model-based adaptation process. These models are often used in the monitoring, analysis, plan and execute phase of a MAPE adaptation control loop, as introduced in Section 2.9 [28]. Vogel and Giese investigated requirements for a modeling language for adaptation models and introduce patterns for using these models within a feedback loop [36].

The modeling approach differs according to the used adaptation process. In general, the analyzing and planning phase decide if and how the system should be adapted. While rule-based mechanisms work efficient and support early validation, search-based techniques benefit in scalability. The monitor phase updates the reflection model, which describes the running system and its environment, usually in an abstracted level. The analyze phase decides on this updated model, if the system fulfills its goals and if an adaptation is required. An evaluation model consists of multiple constraints, which are validated against the reflection model. The planning phase decides, based on degrees of freedom which are defined in a change model, how the system should be adapted. In the execution phase the adaptations are applied to the running system. Evaluation and change models are generally considered as adaptation models.

The requirements for adaptation models to analyze models, decide if an adaptation is required, plan adaptations, and decide how a system should be adapted can be segmented.
into two parts. The *language requirements* describe functional as well as non-functional requirements for adaptation model languages. *Framework requirements* specify the demands for the execution environment on adaptation models. This is limited to the process determining how adaptation models are employed and executed in the feedback loop.

The specification of each requirement is explained in the work of Vogel and Giese in detail \[36\]. Adaptation models as well as adaptation frameworks, implementing a specific adaptation process, can be evaluated against these assessments and requirements.
4. Modeling

To enable a model-based adaptation and optimization of networks, existing models should be used and additional models have to be developed, to meet the challenges for a model-based adaptation process. This chapter introduces the input and output models of the adaptation process as well as models used within the adaptation procedure.

The required extensions to DNI core meta-model are described in Section 4.1. The available adaptations should be annotated by costs, so a cost model is introduced in Section 4.2. The adaptation points, defining the degree of freedom of a DNI network infrastructure, are explained in Section 4.3. The Service Level Agreements (SLAs), specifying the performance requirements on the network, are introduced in Section 4.4. Section 4.5 describes the adaptation plan model, containing all scheduled adaptation actions which are associated with an adaptation branch. In Section 4.6 a generic analysis result model is presented, which is used to generalize the analysis results of different solvers to a uniform model. To apply the scheduled adaptation actions of an adaptation plan to an instance of a DNI network infrastructure model, a transformation, described in Section 4.7, is used. The transformation to convert SimQPN specific analysis results to generic analysis results is introduced in Section 4.8.

4.1. Extensions on DNI Core Meta-Model

The adaptation process should be modeled as an optional extension to the existing DNI core meta-model. Therefore all adaptation related specifications, like definitions of adaptation points (degrees of freedom) or costs are a modeled in a separate meta-model besides the DNI core. However, some extensions on DNI core meta-model would facilitate the definition of adaptation models, so that DNI is extended by a few features. It is important that the changes are backward compatible, meaning that all current existing DNI model instances can still be loaded and analyzed. The extensions to DNI core meta-model are not directly related to the adaptation models and would possible be useful for other future works on DNI.

This section describes and depicts the introduced changes on the DNI core meta-model. The extension to make some entities unique identifiable is described in Section 4.1.1. Section 4.1.2 introduces entity types for defining model types on entities.
4.1.1. Identifiable Entities

Addressing entities in a model can either be done by specifying the position of the entity in the model or by defining an Unique Identifier (UID) for each entity. As the first approach requires holding the same sequence of entities and especially removals are difficult to handle, the second approach by an unique identifier is easier to handle. For the adaptation process it is required to address individual entities, like interfaces, nodes, or performance descriptions in a the DNI model instance.

The DNI core meta-model version, which was used as basis for the adaptation process, already includes the abstract class Identifier which assigns every instantiated object an auto-generated identifier, which can be manually overridden.

The most of entities which should be addressed already extend this class and contain a unique identifier, excepting the performance descriptions of links, network interfaces, end nodes, intermediate nodes, and SDN nodes. To tackle this absence of the identifier on those entities, they are also extended by the abstract class Identifier. As all performance descriptions implement the interface PerformanceSpecification, it is sufficient to extend this interface by Identifier to assign an UID to the previously listed entities.

Figure 4.1 depicts the part of the new inheritance structure of the interface PerformanceSpecification. As this interface now extends the abstract class Identifier, the children PerformanceLink, PerformanceNetworkInterface, EndPerformance, IntermediatePerformance, and PerformanceSdnNode get an UID assigned by the inheritance.

![Figure 4.1: Extension of DNI PerformanceSpecification entity by abstract class Identifier.](image)

4.1.2. Annotate Entities by Entity Types

In the actual DNI core meta-model the parameters and nested entities are set individually for each entity. Nested entities are defined as entities, which are included in other entities, like network interfaces in an intermediate node. For the adaptation process two different adaptation actions could be identified. At first parameters, i.e., attributes of an entity could be changed. Increasing the throughput of an interface from 1 Gbps to 10 Gbps is one example for an parameter adaptation. Secondly, an entity could be replaced by another entity. This could either be a nested entity, e.g., replacing a SFP module in a switch, or a top level entity, e.g., the switch itself.
One approach to specify adaptable parameters and replacement entities is to define the alternatives for each entity individually. For this approach no changes on the DNI core meta-model are necessary, but as each entity requires its own specification of alternatives, such an approach does not scale well in huge networks with many alternative configurations. Another approach, inspired from the practical side, is to specify an entity type for each entity, which could be interpreted as a vendor specific model type. Alternative parameters and alternative entities have to be specified only once for each entity type and can be applied to all entities, related to the corresponding type.

The concrete adaptation model, the so called adaptation points, are described in Section 4.3. As the entity types should be independent from the adaptation points, the annotation is implemented in the DNI core meta-model. This enables using the entity types for other extensions on DNI in future. As entity types are required for all entities, the required attributes are specified in the interface ITypedEntity. The interface contains three attributes, a required name, an optional description, both extended from existing interface NamedElement, and an unique identifier UID, extended from abstract class Identifier. The inheritance structure of the interface ITypedEntity is depicted in Figure 4.2.

![Inheritance structure of ITypedEntity](image)

The entity type instances are stored in a new container element EntityTypes, which is directly attached to the DNI root element NetworkInfrastructure, as shown in Figure 4.3. As the entity types of the different entities have to be distinguishable, each of are modeled in a separate class, which all implement the ITypedEntity interface and are referred to and from a specific entity, as depicted in Figure 4.3. Table 4.1 shows the already existing entity classes and their new corresponding entity type class. As referencing an entity type from an entity is optional the backward compatibility to existing DNI model instances is retained.

![Inheritance structure of ITypedEntity](image)

<table>
<thead>
<tr>
<th>Entity Class</th>
<th>Corresponding Entity Type Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>NodeType</td>
</tr>
<tr>
<td>EndPerformance</td>
<td>EndPerformanceType</td>
</tr>
<tr>
<td>IntermediatePerformance</td>
<td>IntermediatePerformanceType</td>
</tr>
<tr>
<td>PerformanceSdnNode</td>
<td>SdnNodePerformanceType</td>
</tr>
<tr>
<td>NetworkInterface</td>
<td>NetworkInterfaceType</td>
</tr>
<tr>
<td>NetworkInterfacePerformance</td>
<td>NetworkInterfacePerformanceType</td>
</tr>
<tr>
<td>Link</td>
<td>LinkType</td>
</tr>
<tr>
<td>LinkPerformance</td>
<td>LinkPerformanceType</td>
</tr>
</tbody>
</table>

Table 4.1.: Entity classes of DNI with their corresponding entity type classes.
4.2. Adaptation Costs

Adaptation on existing network structures could introduce costs. A cost model is used to calculate the overall costs for an adaptation process, maybe consisting of multiple adaptation actions. The overall costs are useful for network operators to weight between different solutions. Additionally, some potentially solving models could be excluded from analysis or further adaptation processes, if they exceed the predefined cost constraints. As adaptations on different entities introduce different costs, they are modeled for each entity. A software parameter change would typically arise lower costs than replacing a hardware switch, for example.

4.2.1. Different Types of Costs

In this thesis the term “costs” is not limited to the financial costs, i.e., investment. An adaptation would have more effects than only financial. For example several changes would require handling time of a network operator for reconfiguring and hardware operations. A sophisticated cost model should be extendable by additional cost types, which could be introduced in future. The expected downtime for applying an adaptation, the Total Cost of Ownership (TCO), the energy consumption, the realization time, and the shipping time of new hardware, are examples for additional cost types.

The current model would support the cost types investment and handling time, which interpretations are explained in the following:
**Investment.** The investment is the financial expense to realize the adaptation on the current network. The investment value should include the cost for additional hardware, software, and licenses to deploy the changes.

**Handling time.** The handling time defines the amount of time for human interaction to apply a specific adaptation on the network. Setting the handling time to zero has a special meaning. Some adaptations can be automatically deployed via APIs from a network management software. Such changes do no require human interaction and are suitable for an automatic network adaptation process. Restricting the handling time to zero by user constraints, only such adaptations will be suggested, which would require no human interaction.

Table 4.2 shows three examples of adaptations, annotated by costs. The packet processing time is defined as software change, which requires 600 seconds handling time by the network operator for reconfiguring, which results in 10 $ for the working time. Changing the switching capacity can be applied fully automatically via an API without any human interaction and investment costs in this example. This adaptation is especially suitable for automatic network adaptations. Replacing a SFP module is expensive because new hardware has to be purchased, but the handling time to replace such a module is relatively low.

<table>
<thead>
<tr>
<th>Adaptation</th>
<th>Investment</th>
<th>Handling Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>decrease packet processing time</td>
<td>10 $</td>
<td>600 sec.</td>
</tr>
<tr>
<td>increase switching capacity</td>
<td>0 $</td>
<td>0 sec.</td>
</tr>
<tr>
<td>replace SFP module</td>
<td>800 $</td>
<td>60 sec.</td>
</tr>
</tbody>
</table>

**4.2.2. Calculation of Costs**

For the calculation of costs, only the desired target state could be considered. Table 4.3 shows an example of such a cost transition matrix, where the costs for a target state are independent from the current state. The investment for purchasing a 40 Gbps SFP module is fixed at 450 $, regardless if currently a 1 Gbps or 10 Gbps module is used. In the cost model such costs are modeled as fixed cost function, which only takes one value for each desired target state.

<table>
<thead>
<tr>
<th>Current / Desired</th>
<th>1 Gbps</th>
<th>10 Gbps</th>
<th>40 Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Gbps</td>
<td>-</td>
<td>300 $</td>
<td>450 $</td>
</tr>
<tr>
<td>10 Gbps</td>
<td>100 $</td>
<td>-</td>
<td>450 $</td>
</tr>
<tr>
<td>40 Gbps</td>
<td>100 $</td>
<td>300 $</td>
<td>-</td>
</tr>
</tbody>
</table>

Beside the costs, which only depend on the target state, there are costs also depending to the current state. Table 4.4 shows an example cost transition matrix for licenses. If overall four licenses are required, the costs depend on the number of currently used licenses. If currently two licenses are used, only two additional licenses have to be purchased. If instead currently no licenses are used, all of four licenses have to be purchased. Modeling these costs, requires to specify in addition to the target state an additional source state for each transition. This type of cost function is currently not implemented in the framework, but can be easily extended in the cost model of Section 4.2.3 as required.
Table 4.4.: Example costs, depending on source and target state.

<table>
<thead>
<tr>
<th>Current / Desired</th>
<th>1 license</th>
<th>2 licenses</th>
<th>3 licenses</th>
<th>4 licenses</th>
<th>5 licenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 license</td>
<td>-</td>
<td>10 $</td>
<td>20 $</td>
<td>30 $</td>
<td>40 $</td>
</tr>
<tr>
<td>2 licenses</td>
<td>0 $</td>
<td>-</td>
<td>10 $</td>
<td>20 $</td>
<td>30 $</td>
</tr>
<tr>
<td>3 licenses</td>
<td>0 $</td>
<td>0 $</td>
<td>-</td>
<td>10 $</td>
<td>20 $</td>
</tr>
<tr>
<td>4 licenses</td>
<td>0 $</td>
<td>0 $</td>
<td>0 $</td>
<td>-</td>
<td>10 $</td>
</tr>
<tr>
<td>5 licenses</td>
<td>0 $</td>
<td>0 $</td>
<td>0 $</td>
<td>0 $</td>
<td>-</td>
</tr>
</tbody>
</table>

Each of the above described cost functions can be applied for every cost type, specified in Section 4.2.1.

4.2.3. Modeling Costs

The cost model is a part of the adaptation points model, described in Section 4.3. The cost modeling approach is introduced in this separate section, as the modeling concept is also applicable to other descriptive meta-models. The cost model is depicted in Figure 4.4. All adaptable entities and adaptable dependencies have to implement the interface ICostable, which extends them by a list of abstract Costs. The classes Investment and HandlingTime are implementations of the abstract class Cost. As described in Section 4.2.1, these additional implementation can be added to support other costs types like downtime or TCO in future. Each cost implementation contains a cost function, implementing the interface ICostFunction. The FixedCostFunction is the currently only supported cost function, which takes a single double argument for the cost value. This cost function is used to model costs, depending on target state.

Figure 4.4.: Meta-model of Cost.

Figure 4.5 depicts an example of an alternative entity, representing a replacement for a SFP module. The dependency contains two different cost types. The Investment with fixed costs of 450.0 $ describes the amount of money, required for an additional SFP module with a throughput of 10 Gbps. The HandlingTime with a value of 600.0 seconds specifies the working time of the network operator, required to execute the adaptation. Beside the adaptation to 10 Gbps there could be other alternatives with other investment values and maybe the same handling time.

This cost model fulfills three main aspects. Firstly, entities and parameters which should be annotated by costs, only have to implement the ICostable interface. This enables an easy reuse of the cost model. At second, the model can be easily extended to additional cost types, by adding additional classes. Lastly, the definition of cost functions enable more complex cost calculation scenarios, which are out of scope of this work. How this cost model is used in the DNI adaptation points model is described in Section 4.3.
 Assessment of Goals

In Section 1.4 the following goal and two research questions have been defined.

**Goal 3:** Develop a cost model for adaptations. This model has to collaborate with the adaptation point model, described in Goal 1.

**RQ 3.1:** Which different types of costs are meaningful?

**RQ 3.2:** How are the costs calculated for transition from one alternative configuration into another?

Research question RQ 3.1 is answered in Section 4.2.1. The term “costs” is not limited to the financial aspect only. Instead of this, also the handling time, expected downtime for applying an adaptation, the TCO, the energy consumption, the realization time, and the shipping time of new hardware could be considered. As the definition of meaningful cost types requires several preliminary considerations which are out of scope of this work, only the cost types “investment” and “handling time” are selected for the current model. The model can be easily extended by additional cost types as required.

The calculation of costs is described in Section 4.2.2 which answers research question RQ 3.2. Two types of cost calculations have been identified. The first cost type, only depends on the target state, without considering the previous configuration. This is useful for hardware purchases, as the previous used hardware is commonly irrelevant. The second cost type depends on source and on target state. The calculated costs vary for a target state, from the current configuration value. This type of cost is useful, e.g., for licenses. If additional licenses are required, the already existing licenses could usually used further on.

Goal 3 is fulfilled, as both research questions RQ 3.1 and RQ 3.2 are answered. The cost model will be integrated into the in Section 4.3 introduced adaptation points model and will therefore collaborate with it.

### 4.3. Adaptation Points

The main focus of network optimization is the adaptation of a network, to met the predefined performance expectations. Beside the adaptation process itself, the *degrees of freedom*, the so called *adaptation points*, have to been specified. The adaptation points are fundamental for the adaptation process and define, which adaptations on the network are allowed and which alternative configurations are available and valid.

This section is outlined as follows. The preliminary considerations for modeling approach are introduced in Section 4.3.1. The identified types of adaptations for parameters and entities are described in Section 4.3.2 and Section 4.3.3. Section 4.3.4 summarizes and concludes this section.
4.3.1. Modeling Approach

The model-based adaptation process for DNI network infrastructure is implemented as extension to existing DNI core meta-model. This enables backwards compatibility of existing DNI model instances and does not overload networking infrastructure models with the optional adaptation features. A DNI adaptation points model references an existing DNI network infrastructure model, to get access to the entities and entity types of the network.

The major relation between adaptation points models and network infrastructure models are entity types, developed as extension to DNI and introduced in Section 4.1. The entity types classify the entities by their model type, e.g., a specific hardware type of a vendor. This enables a single specification of alternative parameters and alternative entities for each entity type. Instead of this, the degrees of freedom could have also been defined for each entity, but such an approach would require high effort for the user, creating an adaptation points model instance.

The adaptations can be categorized into two classes. Firstly, parameters should be adaptable. Parameters are attributes within entities and are usually configuration settings or single performance metrics. The second category for adaptations are entities. While parameter adaptation allows fine tuning on specific settings, entity adaptation replaces whole entities. The replacement of an entity has no direct impact on the network performance, but the type of entity defines the configuration space for the included parameters and alternatives of nested entities. The adaptations should be assessable regarding to costs, whereby the in Section 4.2 introduced cost model is used, which allows multi-dimensional costs like investment and handling time.

The root element AdaptationPoints of the DNI adaptation points model is shown in Figure 4.7. The adaptation points model references a network infrastructure model. The link to NetworkInfrastructure is depicted by dashed line the figure. The AdaptationPoints root element contains an AdaptationRepository and a GroupRepository, which are explained in the following sections in detail.

4.3.2. Adaptation of Parameter

Adapting parameters is one opportunity to optimize networks, i.e., to increase the performance. The parameter adaptation is defined as changing attributes of entities used in DNI models. In real-world, parameters are often configuration settings within a component, may be hardware or software. Changing the forwardingBandwidthBPS parameter in an IntermediatePerformance entity is an example for a parameter adaptation.

DNI uses different types for parameters, supplemented by different units. The inheritance structure of the so called Dependency was introduced in Section 2.7 and is depicted in Figure ???. For the definition of alternative parameters it is not sufficient to change only the numeric value of an existing dependency, also the data type as well as the unit should be adaptable. This allows to change a ConstantLongVariable of 1 second to a ConstantDoubleVariable of 0.5 millisecond for example. The definition of the alternative parameters has to respect the supported data type for each entity as specified in DNI. If a parameter in DNI network infrastructure requires a data type implementing Variable, the alternatives could not be defined as the most abstract data type Dependency, instead of this only alternatives of data type Variables have to be allowed here.

The data types for adaptable parameters AdaptableDependency and AdaptableVariable, corresponding to the data types Dependency and Variable are depicted in Fig-
4.3. Adaptation Points

To support the sophisticated definition of units within a dependency the `AdaptableDependency` implements the interface `Dependency` from [DNI] core meta-model. Inherit from the abstract class `Identifier` adds a unique ID to each dependency, which is useful within the adaptation process to distinguish between different parameter alternatives. The implemented interface `ICostable` adds a list of the in Section 4.2 introduced costs to each dependency. For each cost type, currently investment and handling time, a cost function, commonly fixed costs, can be added to the dependency. The `AdaptableVariable` is the corresponding data type for `Variable` in [DNI] and implements the interfaces `AdaptableDependency` and `Variable`.

![Diagram showing the meta-model of AdaptableDependency and its inheritance structure.](image)

The classes `AdaptableConstantLongVariable` and `AdaptableConstantDoubleVariable` implement the interface `AdaptableVariable` to be used as alternative for variable or dependency attribute. They also inherit from their corresponding DNI class `ConstantLongVariable` and `ConstantDoubleVariable` to inherit the attribute `value` containing the `Long` or `Double` value.

For the adaptation points it is essential, which of the alternative parameters are valid for an attribute. A generalized approach, e.g., a global set for the maximal supported bandwidths for links does not suit well, as virtual links, copper connections and fiber optic cables does not support the same alternatives. In [DNI] the parameters are contained in an entity and these entities can be annotated by a specific entity type, as introduced as [DNI] extension in Section 4.1.2. This approach enables to define the alternative parameters for each entity types, by creating an corresponding adaptable entity, like a template or blueprint, for each entity type in the adaptation points.

The adaptation points model, depicted in Figure 4.7, contains an `AdaptationRepository` where the adaptable entities for the entity types are defined. Within each adaptable entity the alternatives for each parameter are specified. Instead of directly put a set of alternatives for each attribute in the adaptable entity, an interface of `AlternativeDependency` and `AlternativeVariable`, as shown in Figure 4.6, is used. The
implementations of these interfaces are AlternativeDependencySet and Alternative-VariableSet. This approach enables other definitions beside of sets, which could be ranges of values in future for example. The adaptable entities are explained in following in detail.

Figure 4.7.: Meta-model of AdaptationPoints.
Adaptable Node, Adaptable Network Interface and Adaptable Link

These entities do not contain any parameters, so a parameter adaptation is not relevant and possible on these entities.

Adaptable Intermediate Performance

An AdaptableIntermediatePerformance contains the parameters forwardingLatency, forwardingBandwidthBPS, and switchingCapacityPPS. These parameters comply with the definition of the IntermediatePerformance of DNI. All of these parameters are defined as Dependency in DNI and are specified as the corresponding data type AlternativeDependency in the adaptation points.

Adaptable Network Interface Performance

The NetworkInterfacePerformance in DNI defines the parameters packetProcessingTime of type Dependency and interfaceThroughput of type Variable. These parameters are represented by the corresponding attributes of AdaptableNetworkInterfacePerformance, whereby for packetProcessingTime the data type AlternativeDependency and for interfaceThroughput the data type AlternativeVariable is used.

Adaptable Link Performance

The AdaptableLinkPerformance is the corresponding adaptable entity of LinkPerformance in DNI. The parameters forwardingLatency and forwardingBandwidthBPS define their AlternativeDependencies for their corresponding parameters, specified in DNI.

Example of Parameter Adaptation

The usage of the alternative dependencies is illustrated in Figure 4.8 by the object IP1 of type IntermediatePerformance. This entity is part of the network infrastructure specification of DNI. The intermediate performance references the entity type IP-A which could be interpreted as a type of backplane. An additional entity type IP-B is also visible, but currently not referenced by any intermediate performance entity. The forwardingBandwidthBPS of IP1 is initially set to 1 Gbps.

The adaptation process needs to know the alternative values for this parameter. The adaptable intermediate performance AdapIP-A also references the entity type IP-A and contains a set of alternative values for the parameter forwardingBandwidthBPS. Beside the currently used 1 Gbps, also a forwarding bandwidth of 10 Gbps is available. The forwarding bandwidth of 10 Gbps is also annotated by a cost model. Changing the forwarding bandwidth of the intermediate performance to 10 Gbps arises investment costs of 200 $ and requires a handling time of 600 seconds. This could be interpreted as installing a license for increased performance. The forwarding bandwidth of 40 Gbps, as defined in the adaptable intermediate performance AdapIP-B, is not a valid alternative, as the entity type IP-A of the intermediate performance IP1 does not match the entity type IP-B of AdapIP-B.
Figure 4.8.: Example model instance of adaptable dependencies.
Assessment of Goals

In Section 1.4 the following goal and two research questions has been defined.

**Goal 1:** Develop an adaptation point model for alternative configurations. The model should decorate or extend DNI.

**RQ 1.1:** How could the alternative configurations be modeled?

**RQ 1.2:** On which level can the adaptation point model extend DNI?

Section 4.3.2 answers research question RQ 1.1. The introduced model specifies the alternative parameters as abstract entities `AdaptableDependency` and `AdaptableVariable`. Each of these abstract entities are implemented as a set, containing the corresponding adaptable parameters. By this approach, the dependency structure of DNI explained in Section 2.7.1.1 is retained. Instead of the set, also other implementations, e.g., ranges, might be possible in future work.

Also, research question RQ 1.2 is answered by Section 4.3.2. The adaptable parameters are specified by the introduced entity type for each entity. This approach extends DNI on the model level (M1). However, the entity type enables a definition of alternatives on a meta-level instead of specifying them for each entity.

As research questions RQ 1.1 and RQ 1.2 are answered, goal 1 is fulfilled. The developed adaptation points model extends the existing DNI network infrastructure model.

**4.3.3. Adaptation of Entity**

Adapting, i.e., replacing, entities is another opportunity to optimize network performance. Entity adaptation means to add, remove or replace entities in the network infrastructure, whereby currently only replacements are supported. The replacement of an entity is changing the entity type. As already mentioned, the entity type itself, as referenced in the DNI network infrastructure, has no direct impact on the performance of the entity. The entity type defines the alternatives for parameters, explained in Section 4.3.2, and the possible replacements for the entity itself. To enable adaptation of an entity, the entity has to contain a unique ID, which is ensured by an auto generated `UID` through inheritance. Additionally, the entity has to reference an entity type and this entity type must have a corresponding adaptable entity, defined in the `AdaptationRepository` in the adaptation points.

The adaptation process needs to know which entities are allowed as replacement for the current entities. The simplest approach is that an entity can be replaced by any entity of the same class. This is impractical, as replacing the link performance of a copper link through a link performance of a fiber optical cable will not work in real world. Therefore, the allowed replacements should be defined for each entity type, i.e., adaptable entity. The entities `IntermediatePerformance`, `NetworkInterface`, `NetworkInterfacePerformance`, and `LinkPerformance` are nested within a parent entity. For these entities the alternatives are defined on the level of the parent entity. This corresponds to real world where, for example, the used switch model specifies the available backplanes for intermediate performance and the supported interfaces.

The entities `Node` and `Link` are not contained by any structural entity, so there is no parent entity which could define the valid replacements. To allow the replacement of these entity by any other entity of this class would result in invalid substitutions, like the replacement of a firewall through a switch, which are both of type intermediate node. In this example, the class (Node) would be obtained and a performance gain would be achieved, but replacing a firewall through a switch would introduce a massive security
issue and would not be a valid replacement in real world. Replacing each link by any other link is similar impractical, as either virtual links nor fiber optical links are suitable alternatives for a copper connection.

To handle the alternatives for Nodes and Links the adaptation points model contains a set of NodeAdaptationGroups and LinkAdaptationGroups within AdaptationGroup. There could be multiple NodeAdaptationGroups and LinkAdaptationGroups defined, referencing entity types of nodes and links, which could be replaced by each other. Vice versa, each adaptable node and adaptable link should reference such a group, specifying all allowed alternatives for the entity.

If an entity is replaced, it is not sufficient to change the reference to the entity type, also the parameters have to be adapted to the allowed alternatives by the new entity type. Additionally, an entity could also contain nested entities, like NetworkInterfacePerformance in NetworkInterface. If the type of network interface is changed, also the corresponding performance has to be adapted to a valid type. A default entity could be specified for nested entities, which should be selected for entity replacements. If no default is set, the first alternative would be used. The selection of valid contained parameters as well as replacing nested entities is scope of the adaptation process.

Any adaptable entity can be annotated by the cost model introduced in Section 4.2. If a node is replaced by an alternative, the costs of the new entity arise. Costs can be modeled in multiple dimensions like investment and handling time, as described in parameter adaptation in Section 4.3.2. In addition to the costs for the replacement, also costs for contained parameter adaptation and nested entities have to be considered. This is enabled through annotating the nested entities as well as alternative parameters by costs. Calculation of costs and respecting cost constraints is scope of the adaptation process.

The adaptable entities as depicted in Figure 4.7 are explained in the following:

Adaptable Node

An AdaptableNode could be either an intermediate node, end node or both. A node could contain several network interfaces and an intermediate performance description, in case of an intermediate node. The alternative intermediate performance as well as the alternative network interfaces are referenced as collection in AdaptableNode. The intermediatePerformanceDefault and networkInterfaceDefault reference the default for the nested entities. As node is not contained by any structural parent entity, an adaptable node could also reference a NodeAdaptationGroup which contains all replacement candidates for the specific node.

Adaptable Intermediate Performance

The AdaptableIntermediatePerformance does not contain any nested entities, instead only alternative parameters could be specified, as described in Section 4.3.2. If an intermediate performance should be replaced by another performance descriptor, the containing AdaptableNode entity has to be considered for valid, alternative AdaptableIntermediatePerformances.

Adaptable Network Interface

An AdaptableNetworkInterface has no parameters, which could be adapted. Instead a network interface contains a nested performance descriptor the AdaptableNetworkInterfacePerformance. This nested entity is referenced as collection in AdaptableNetworkInterface. The default AdaptableNetworkInterfacePerformance is referenced
by `networkInterfacePerformanceDefault`, which is used first, if a network interface is replaced. The alternatives for network interfaces are specified by the corresponding `AdaptableNode`.

**Adaptable Network Interface Performance**

The `AdaptableNetworkInterfacePerformance` is the performance descriptor for a network interface. This entity contains the performance parameters for the network interface and no further nested entities. The valid replacements for a network interface performance are specified by the parent `AdaptableNetworkInterface`.

**Adaptable Link**

An `AdaptableLink` represents an alternative for a network connection, which could either be physical or virtual. The `AdaptableLink` references a list of `AdaptableLinkPerformance` and an `AdaptableLinkPerformanceDefault` which are used in the same way as described for the intermediate performance of a node. As a link has no parent in the network structure, the alternatives are specified by a referenced `LinkAdaptationGroup`. All `AdaptableLinks`, referenced by the group, are valid alternatives for the current used link.

**Adaptable Link Performance**

The `AdaptableLinkPerformance` describes the alternative performances for links. This entity contains the alternative parameters and does not reference any further nested entities. The alternatives for an `AdaptableLinkPerformance` are specified by the corresponding `AdaptableLink`.

**Example of Entity Replacement**

Modeling of adaptable entities is explained in the following by an example. The corresponding object diagram is illustrated in Figure [4.9]. The network contains a link `L1` between two nodes, which are omitted in the figure for the sake of clarity. The link references the type `LT-A` and contains the performance descriptor `LP-1`. The adaptation plan contains three adaptable links `AdapL-A`, `AdapL-B`, and `AdapL-C`. The adaptable links `AdapL-A` and `AdapL-B` references the same link adaptation group `Fiber`, the adaptable link `AdapL-C` references the group `Copper`. For replacing the link `L1` only the links of the same group (`Fiber`) are valid alternatives. As `AdapL-A` references the same entity type `LT-A` as the current used link `L1`, the only valid alternative is `AdapL-B` referencing the entity type `LT-B`. If this entity is used, also the link performance has to be adapted, if the performance descriptors of `AdapL-A` and `AdapL-B` do not match. If such nested entities are replaced, also the parameters have to be changed, if the current values are not compatible with the parameters of the new entity type.

**Assessment of Goals**

In Section [4.4] the following optional goal and two optional research questions have been defined.

**Goal 2:** Develop an adaptation point model for network structure. The model should decorate or extend `DNI` *(optional)*

**RQ 2.1:** How could the alternative configurations and the relation between them be modeled? *(optional)*

**RQ 2.2:** On which level can the adaptation point model extend `DNI` *(optional)*
Section 4.3.3 answers research question RQ 2.1. The alternative configurations, related to entities, are also modeled by the entity type. If an entity should be replaced, not the related entity type, but the entity type of the parent, containing entity has to be considered. In case of nested entities, each entity defines its valid children, which can be replaced by each other. In case of the two top level entities Node and Link the valid replacements are managed in groups, as they are not contained by a parent entity.

The answer of research question RQ 2.2 is similar to the answer of research question RQ 1.2. The introduced entity type specifies the alternative entities on model level (M1). The alternative entities are specified on a meta-level, grouped by the entity type, instead of specifying them for each entity.

As research questions RQ 2.1 and RQ 2.2 are answered, goal 2 is fulfilled. The adaptation points model for entity adaptation extends DNI network infrastructure model.

4.3.4. Summary

The introduced adaptation points are an extension to existing DNI network infrastructure. This facilitates backward compatibility for existing DNI models and does not overload them, if no adaptation is desired. The adaptation points only require a minimal intervention in current DNI core meta-model as described in Section 4.1. The extension is at least required to introduce entity types as basis for defining alternatives for all entities of same type, instead of specifying them for each entity individually. The adaptation points model support parameter adaptation and entity adaptation. In future, the adaptation points model can be extended by further adaptations like additional links, alternative routes or changes in the network structure.

4.4. Service Level Agreement

The definition of Service Level Agreements (SLAs), as described in Section 2.1, is fundamental for the adaptation process. The agreements specify the performance expectations, i.e., the Quality of Service (QoS) parameters, on the data center network. Aim of the
adaptation process is to detect SLA violations and suggest adaptations which will solve these violations. The objectives of the contract have to be validated against the analysis result to decide about SLA violations. Detailed knowledge about which objective is violated and where the violation occurs, e.g., identification of node or link, facilitates efficient and target-oriented adaptation methods.

To observe a performance aspect in the adaptation process, it has to be implemented in three components. Firstly, the used analysis method, e.g., simulation technique, must observe and extract the corresponding measurement value. Secondly, the SLA model has to formalize the performance expectations as constraint. Thirdly, at least one adaptation tactic needs provided adaptation methods to tackle the violated objective.

This section introduces a SLA meta-model which is used for the adaptation process and supports the objectives ObjectiveLinkThroughput, ObjectiveNetworkInterfaceThroughput, and ObjectiveMinSwitchingThroughput. Measurement values for these objectives can be extracted from the analysis result, introduced in Section 4.6. Adaptation tactics for each objective are available and are explained in Section 5.3. The predefined SLA model can be easily extended in future, if additional objectives and metrics are required to be observed. Each objective contains a threshold and multiple attributes for a fine grained definition which flow, node, or link the objective should observe.

The SLA meta-model is depicted in Figure 4.10. The three objectives with their thresholds and attributes are described in the following. Each of the objectives can be defined multiple times within a SLA model for different scopes, e.g., observing switching throughputs of multiple intermediate nodes.

![Figure 4.10.: Meta-model for SLA definition.](image-url)
ObjectiveLinkThroughput  The objective for link throughput is used to define a minimum bandwidth on connections between nodes. The objective requires to set a minimumThroughput, to define the minimal bandwidth in bits per second the link has to provide. If the optional attributes flowUid, applicationUid, and linkUid are set, the objective applies only to the specified flow, destination application or link. Omitting for example the definition of a specific link and destination, than all nodes traversed by this flow have to fulfill this objective.

ObjectiveSwitchingThroughput  To specify the minimum switching throughput within an intermediate node, the objective for switching throughput is used. The minimumThroughput defines the minimal switching bandwidth of the node. The optional attributes flowUid, applicationUid, and nodeUid enable a fine grained specification of the objective, to only observe a specific flow, destination application or node.

ObjectiveNetworkInterfaceThroughput  The minimum throughput of a network interface can be observed by the objective for network interface throughput. The parameter minimumThroughput defines the minimum throughput, which has to be achieved for sending and receiving over a network interface. To apply the objective to a specific flow, destination application, node, or network interface, the optional parameters flowUid, applicationUid, nodeUid, and interfaceUid can be set. If for example no interface UID is set and a node is specified, the objective is applied on all interfaces on the node.

An example definition of a minimum switching throughput objective, including its attributes and metrics, is shown in Figure 4.11. The specified flowUid “Flow 1” binds the objective only on traffic for Flow 1. The unspecified (“<unset>”) applicationUid does not limit the objective to a specific application, so it is applied to all destination applications. The nodeUid “Switch 1” restricts the objective only to traffic traversing Switch 1. The metric for the minimum throughput of 104857600 represents a minimum throughput of 100 Mbps (1024 · 1024 · 100 = 104857600 bits per second). The effect of the objective is illustrated in Figure 4.12 as a black filled circle. The only observed value is the switching throughput in Switch 1 of Flow 1 (green). The other flows as well as the other throughputs are not monitored by this objective. Additional objectives could be defined, to observe the same metric or other metrics on other places.

<table>
<thead>
<tr>
<th>Objective 1: ObjectiveSwitchingThroughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowUid = &quot;Flow 1&quot;</td>
</tr>
<tr>
<td>applicationUid = &lt;unset&gt;</td>
</tr>
<tr>
<td>nodeUid = &quot;Switch 1&quot;</td>
</tr>
<tr>
<td>minimumThroughput = 104857600</td>
</tr>
</tbody>
</table>

Figure 4.11.: Example object of an ObjectiveSwitchingThroughput, defining the minimum throughput for a specific flow and node.

Assessment of Goals

In Section 1.4 the following goal and two research questions have been defined.

Goal 4: Develop a component for SLA violation detection. The component has to detect SLA violations based on a previous analysis (simulation, or analytical method).

RQ 4.1: How can SLAs be modeled?
RQ 4.2: How can SLA violations be detected?

Section 4.4 answers research question RQ 4.1. As defined, a SLA could consist of multiple objectives. The current SLA model supports the objectives ObjectiveLinkThroughput, ObjectiveNetworkInterfaceThroughput, and ObjectiveMinSwitchingThroughput. Each of them contains an attribute specifying the minimum throughput, which has to be observed. The other optional attributes flowUid, applicationUid, linkUid, nodeUid, and interfaceUid allows a fine grained definition, where the objective should be applied.

As research question RQ 4.2 is not answered in this context, goal 4 is not yet fulfilled.

4.5. Adaptation Plan

A DNI network infrastructure model defines a network structure, network configuration and traffic workload while an adaptation points model specifies the degrees of freedom, i.e., the interchangeable entities and alternative configuration parameters. Scope of an adaptation process is to adapt a DNI network infrastructure through multiple iterations. Applying the desired adaptation actions could be handled in two different ways.

The adaptation actions could be applied to the network model immediately as soon as the adaptation logic decides to change an entity or parameter. Such an approach eases the iterative adaptation, because the working model includes all previous changes and represents the current adaptation state. Additionally, the model can be directly analyzed by solvers without the need of any preprocessing.

Nevertheless applying the adaptations directly to the model arises some challenges. Firstly, the executed adaptations could be applied to a real network, if an appropriate solution was found. In order to do so, an operator is primarily not interested in how the final network looks like, but in a description of the changes between the original state and the desired configuration. Secondly, the costs for the adaptation are essential. Determining the costs for a list of adaptations is much easier than comparing two model instances. Lastly, holding a copy of the DNI model instance would require more memory, instead of only storing a list of adaptation actions.

Therefore the adaptation actions are collected in a separate model, the so called adaptation plan. The adaptation plan contains a list of scheduled adaptation actions which have to be performed on the origin DNI model instance to get the desired adapted network. The adaptation process starts on an empty adaptation plan, representing that
no adaptations have to be performed. Through the adaptation process, the discovered adaptation actions are added to the adaptation plan. The adapted DNI model instance, as required for analysis purposes, can be generated through a transformation on the initial DNI model and the adaptation plan as input models. Beside a collection of adaptation actions, the adaptation plan could also be classified as update language for DNI network infrastructure models.

4.5.1. Requirements for Adaptation Plan

Implementing the adaptation actions in an adaptation plan instead of directly applying them directly to the network model introduces some disadvantages. For each analysis, the adaptation plan has to be merged into a temporary copy of the DNI network infrastructure model to run the analysis on the adapted network. Such merging step is also required when searching for adaptable entities and parameters within an iterative adaptation run. Furthermore, the adaptation plan has to be copied into new adaptation branches, which could be a memory consuming process, if the adaptation plan contains a huge amount of adaptation actions. For the assumption that adaptation plans contain a manageable amount adaptation actions, these disadvantages can be neglected. Merging an adaptation plan into a DNI network infrastructure model can be implemented efficiently and copying an adaptation plan with only a few adaptation actions does not require so many resources. This is especially a more lightweight process than copying the whole DNI network infrastructure model to every new branch.

As mentioned above, the adaptation plan plays an significant role for the adaptation process. To support all desired features, the model of the adaptation plan has to fulfill the following requirements:

**Identifiable** Each adaptation plan has to be unique identifiable. As each adaptation branch has exactly one adaptation plan and each adaptation branch already contains a unique identifier, the identifier from the adaptation branch can be transferred to the adaptation plan.

**Mergeable** New adaptation actions should be merged into an existing adaptation plan instead of only appending them. If an entity or parameter has already been adapted, the existing entry in the adaptation plan should be replaced with the latest value. This facilitates the calculation of costs, as the adaptation plan does not contain any temporary selected transition states. The merging operation is part of the method implementation for the adaptation plan.

**Comparable** Adaptation plans from different adaptation branches should be comparable to detect duplicates. This feature is especially required for the in Section 5.2.6 described history of the redundancy elimination. The unique identifier must not take into account for the comparison, because this would always result in a non equality.

**Sortable** The entities and parameters in the adaptation plan should be sorted in a predeterminated manner. This eases a performant implementation of the merging methods and facilitates comparison of multiple adaptation plans.

4.5.2. Dependency in Adaptation Plan

In Section 4.3 the adaptation points have been classified into parameter adaptation and entity adaptation. The parameter adaptation describes changes on the attributes of entities, e.g., replacing a throughput variable of 100 Mbps to 1 Gbps. An entity adaptation means, inserting or replacing an entity like end nodes, intermediate nodes or links.
4.5. Adaptation Plan

The DNI core meta-model introduces an enhanced definition of parameter values, by an inheritance structure via dependency, variables and functions. Each dependency can additionally supplemented by different units for data, speed and time, partially including a prefix. In order to provide the same functionality for the adaptation process, the dependency structure has to be available in the adaptation plan.

Related to the dependency structure of DNI, the adaptation plan meta-model contains a similar structure of dependency, as shown in Figure 4.13. On top is the abstract Dependency with an unique ID and an abstract method getNormalizedValue() which has to be implemented by its children. This method normalizes the values to be comparable, which is especially achieved by converting them into values without unit prefixes. The unit to a dependency is specified as a DataUnit, SpeedUnit, or TimeUnit. Each of them inherit from the abstract class Unit, which contains the method getFactor() to determine the normalizing factor for the corresponding method in the dependency class. The abstract class Variable inherit from Dependency and is the parent class of ConstantLongVariable and ConstantDoubleVariable. These two implementations contain a value for their respective numeric data type.

![Dependency structure used in adaptation plan.](image)

4.5.3. Structure of Adaptation Plan Model

All adaptation actions are defined in the adaptation plan, which class structure is depicted in Figure 4.14. The root element AdaptationPlan contains a adaptationBranch-Id to provide an identifier for the adaptation plan, as described as a requirement for the
adaption plan meta-model. Furthermore, the adaptation plan contains different entity adaptations, which inherit from \texttt{AbstractAdaptationEntity} including an \texttt{originUid} and \texttt{typeUid} attribute and references an \texttt{AdaptationAction}. The adaptation action specifies by its enumeration value if an entity should be updated or replaced. The \texttt{originUid} attribute is the reference to the corresponding entity in the DNI network infrastructure model, which should be adapted. For all adaptation actions the specification of the \texttt{originUid} is mandatory. The \texttt{typeUid} must be set, if an entity should be replaced by an entity of another type. Replacing a switch from one vendor to a switch form another vendor is an example for such a replacement. The \texttt{typeUid} refers to an entity type, as introduced as DNI meta-model extension in Section 4.1. The specification of the entity type does not directly impact the performance, but the entity type is crucial to determine further alternatives in subsequent iterations of the adaptation process.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{structure_of_adaptation_plan.png}
\caption{Structure of adaptation plan.}
\end{figure}
Entity adaptations can be described for nodes, intermediate performances, network interfaces, network interface performances, links, and link performances, as shown in Figure 4.14. As nodes, network interfaces and links contain no parameters, these entities can only be replaced by other entity types. Although a replacement has no direct impact on the performance, the entity types define the allowed nested entities. For example when replacing an intermediate node by another intermediate node, also the nested entities intermediate performance and network interfaces have to be changed.

The adaptation plan is designed in a flat hierarchy, so that adaptations of nested entities have to be specified as separate entity adaptations. This enables to only adapt a nested entity without specification of the containing parent entity. The originUid in each entity enables directly addressing them. The IntermediatePerformanceAdaptation contains dependencies for forwardingLatency, forwardingBandwidthBPS, and switchingCapacityPPS, corresponding to the intermediate performance definition in DNI. If one of these parameters is set, the parameter has to be changed. Otherwise the parameter of the adapted entity has to be left unchanged. The NetworkInterfacePerformanceAdaptation contains one parameter for interfaceThroughput of type Variable. By the LinkPerformanceAdaptation the propagationDelay and maximalSupportedBandwidth, both dependencies, of a link performance can be adapted.

4.5.4. Summary

The introduced adaptation plan describes all current available adaptation actions. Entities can be updated and replaced and parameters can be set to other types and values. How an adaptation plan can be transformed by a DNI network structure to an adapted DNI model is described in Section 4.7.

4.6. Generic Analysis Result

Solvers are used to analyze the performance of network models, such as DNI. These model instances are descriptive models in a model specific language, i.e., DNI core metamodel. These models have to be transformed to predictive input models for solvers like SimQPN [16] or OMNeT++ [20]. The solvers analyze the predictive model and return an output model, containing throughput predictions and other metrics. In combination with the network infrastructure model, the output model is the basis for further evaluations, like bottleneck detection. In the adaptation process, the metrics of the analysis result are extracted and used for different purposes. The metrics are required to detect violations on the agreements. Additionally, the analysis result facilitates a smart adaptation process, by selecting only meaningful adaptations in order to solve the violations. This includes the choice of effective algorithms and focus the adaption only on violating entities.

4.6.1. Motivation for Generic Analysis Result

By the variety of available solvers also the data format of the analysis output varies between them. This hampers the usage of different solvers, as the adaptation process needs to handle different output formats. Especially new solvers cannot be integrated without extending each component, accessing the analysis result. It is desirable to receive the analysis result in a single generic data format from all solvers, to implement the adaptation process independently from the used analysis method.

Such generic analysis result has to fulfill the some criteria. As the analysis result is DNI specific, it should be compatible with the DNI network infrastructure model. For example, the nested structure of nodes, interfaces and interface performances, as introduced in Section 2.7, should be represented by the generic analysis result and the UIDs.
specified in DNI for entities, should be used. Furthermore, the generic analysis result should be extendable to include further observed entities and metrics. Additionally, the generic analysis result should support different abstraction levels, since not all solvers provide their result in the same granularity.

The usage of solvers should be transparent, which means that a generic interface for calling them and a generic data format for the observations should be used. As a DNI specific customization of solvers is sometimes not possible and particularly not desired, these should be attached by adapters. Such adapter is responsible to implement a common interface to call the specific solver, translate a request into the predictive solver-specific input model, and execute the analysis. After the analysis has finished, the solver-specific observations have to be transformed into the generic analysis result, which is returned to the calling application. This approach enables the interchangeability of solvers and makes the DNI application independent of the used solvers. Software upgrades with impacts on the output format of solvers would only require modifications of the transformations within the adapters, without changing the calling applications.

4.6.2. Generic Analysis Result Model

The analysis result is related to flows, as explained in Section 2.7, which define the traffic between one source and maybe multiple destination applications. These applications are deployed on end nodes, which are connected by one or multiple network interfaces to the network. The traffic usually traverses multiple intermediate nodes, which also have network interfaces. A traversed node is also called hop. Flow, as specified in DNI, can be outlined in the following hierarchy. The traffic definition in a DNI network infrastructure model consists of multiple flows. Each flow could have multiple destination applications, which traverse multiple hops. From one hop to the next hop, the traffic traverses an outgoing network interface on source node, a link, and an incoming network interface on destination node.

This hierarchical model is used as basis for modeling flow-based metrics in the generic analysis result. Each flow, destination application, and network interface can be identified via its UID. Specifying the UID of a node is not necessary, as each application is deployed on exactly one node, and therefore the node can be derived from the location of the application. The model of the generic analysis result is depicted in Figure 4.15 and described in the following.

The root element of the generic analysis result model is AnalysisResult. The currently only available collection of FlowResults enables extending the generic analysis result by non-flow related metrics like power consumption or CPU load in future. Each FlowResult is specified by a unique flowUid. A flow could have multiple destination applications, which are modeled as a collection of ApplicationResult identified by an unique applicationUid. The traffic to an application has to traverse multiple nodes, modeled as NodeResult with an unique nodeUid, and multiple links, represented as LinkResult. As nodes, especially intermediate nodes, have usually multiple network interfaces, a collection of NetworkInterfaceResults is contained in NodeResult. Each network interface is specified by an unique interfaceUid. The BandwidthResult within NodeResult, NetworkInterfaceResult, and LinkResult specify the observations. The attributes in and out of data type long specify the input and output traffic in bits per second. If the output bandwidth differs from the input bandwidth, packets are lost and the entity, containing the bandwidth result, has a throughput bottleneck. In this case the method isLossy() returns the boolean value true, otherwise false.
4.6.3. Summary

The generic analysis result model contains the observation metrics required for the adaptation process. The analysis result can be extended in future if other performance metrics, like delay, or other observations should be included. The introduced generic analysis result model is used in the adaptation process as common data input format for analysis results. Each used solver has to provide the observations in this data format, which could be done via an appropriate adapter.

4.7. Adaptation Plan to DNI Network Infrastructure Transformation

The adaptation plan introduced in Section 4.5 contains all scheduled adaptation actions, which have been discovered through the adaptation process. Instead of applying the adaptations to the DNI network infrastructure model directly, the adaptation actions are collected in a separate adaptation plan model. This model is well suited to execute the adaptation on a real system, as all required adaptations are explicitly listed, and also the costs for an adaptation branch can be determined efficiently by the model. However for analysis purposes and for finding adaptable entities the adapted state of the DNI network infrastructure is required. This section describes a transformation, which generates the adapted DNI network infrastructure by the adaptation plan and the original DNI network infrastructure.
The adaptation plan is structured in a flat hierarchy and every entity adaptation is attached directly to the root element `AdaptationPlan` as depicted in Figure 4.14. To apply the scheduled adaptations from the adaptation plan, the transformation has to iterate through each entity in the adaptation plan.

Each scheduled entity adaptation in the adaptation plan references a corresponding entity in the DNI network infrastructure, identified by a unique ID, specified in the `originUid` attribute. Each entity adaptation defines either an `UPDATE` or `REPLACE` action. For an update action the entity type is left untouched, while for a replacement action the entity type has to be changed. In this case, the entity type of the entity, as described in Section 4.1.2 and depicted in Figure 4.3, has to reference the new entity type, as specified in the respective adaptation plan.

Beside the replacement of entities, which have no directly impact on the performance, also the parameters of entities can be adapted. The entity adaptations `IntermediatePerformanceAdaptation`, `NetworkInterfacePerformanceAdaptation`, and `LinkPerformanceAdaptation` contain parameters like `forwardingBandwidthBPS` in the adaptation for the intermediate performance. All parameters in the adaptation plan are optional. The absence of a parameter means that no adaptation on this parameter has to be executed. If the parameter is set, the dependency of the adaptation plan has to be transformed to the corresponding dependency structure of DNI core meta-model, which replaces the current parameter in the DNI network model.

The described adaptations do not verify the validity of the adaptation, corresponding to the adaptation points. Ensuring the validity of the adaptation actions, e.g., setting only parameters which are allowed by definition in adaptation points, is task of the adaptation process. The transformation works on low level and applies the given adaptation actions without any validity checks.

### 4.8. SimQPN to Generic Analysis Result Transformation

As described in Section 4.6 the adaptation process as well as other DNI related future applications should work on a generic analysis result instead of using solver specific data formats. This enables interchangeability of solvers and decouples the development of DNI related applications from the development of solvers. For the current adaptation process SimQPN is selected as solvers, as especially the Java-based implementation of SimQPN eases the development of an appropriate adapter \[16\]. As soon as adapters for other solvers are available in future, these solvers can also be used.

This section explains the transformation from the SimQPN specific analysis result to the generic analysis result, introduced in Section 4.6. The adapter, containing the transformation, is implemented as a separate artifact beside the adaptation process. The data format of SimQPN result is only briefly introduced, further information can be found in the work of Kounev \[16\].

The existing SimQPN simulation for DNI network infrastructure observes nodes, links, network interface and their corresponding performance descriptors. These entities are modeled as places, queueing places and subnet places in QPN as explained in detail in \[18\]. The traffic is modeled in DNI as flows and each flow could have multiple destination applications. In QPN the traffic for each destination application within a flow is represented by distinguishable types of tokens, annotated by colors. The structure of the SimQPN analysis is shown in Figure 4.16. The SimQPN analysis of DNI models provide metrics about arrival throughput, departure throughput, minimum number of tokens in queue/depository, maximum number of tokens in queue/depository, mean number of
tokens in queue/depository, and the probability that a token of the corresponding color is in queue/depository.

The current adaptation approach requires measurement values for switching throughput, network interface throughput and link throughput. How these measurement values are extracted and transformed in the generic analysis result as described in Section 4.6 is described in the following paragraphs.

The traffic, described by destination applications per flow, is modeled as colors within the observed entities. The regular expression pattern, depicted in (4.1) enables extraction of the flow from the simulation result. The first matching group contains the name of the flow, and the second matching group the application name. Matching groups three, four and five describe the token size as floating point numbers supplemented by an optional unit descriptor like kilo, mega or giga. As the provided measurement values, stated as “tokens per second”, are an inappropriate metric for throughput description, they are converted to the well known metric “bits per second”. Each throughput measurement contains two relevant values for the current adaptation process, which are required to determine throughput bottlenecks. The arrival throughput describes the input throughput of the observed entity, while the output throughput denotes the output throughput of the entity.

\[(\w*)_to_(\w*)_size_(\d+)\.(\d+)(([KMG]?)_[\d]+)\]  

(4.1)

The switching throughput is described in an observed element with the regular expression name pattern depicted in (4.2). The first matching group contains the node name. The type of the observed element is “place”. The observed element contains

\[\text{SimQPNResults}\]

\begin{verbatim}
configurationName: String
date: String
qpmVersion: String
\end{verbatim}

\[\text{ObservedElement}\]

\begin{verbatim}
id: Integer
name: String
statsLevel: Integer
type: String
\end{verbatim}

\[\text{Color}\]

\begin{verbatim}
id: Integer
name: String
realColor: String
\end{verbatim}

\[\text{Metric}\]

\begin{verbatim}
type: String
value: Double
\end{verbatim}

Figure 4.16.: Class-diagramm of SimQPN result.
different colors, representing the different destination applications per flow. This measurements for each nodes and flows are transformed to appropriate generic analysis result by creating the related flow, application, and node element. The arrival and departure throughput token metrics are converted into throughput metrics in “bits per second” for each color and set as input and output values for the bandwidth.

\[
\text{Node}_{(\text{\w*)}_[\text{\d}]+.\text{Switching_of}_{(\text{\w*)}_[\text{\d}]+\text{input-place} \quad (4.2)}
\]

The **interface throughput** can be extracted form the observed element of type “qplace: queue” with the regular expression pattern depicted in (4.3). The first matching group describes the node name and the second matching group the network interface name. The throughput metrics within colors in the network interfaces, corresponding to the different destination applications and flows, are transformed in the same manner as described for switching throughput. The interface throughput results are added as nested network interface elements to the corresponding node in the generic analysis result.

\[
\text{Node}_{(\text{\w*)}_[\text{\d}]+.(.*)_[\text{rt}x_of_\text{(\w*)}_[\text{\d}]+} \quad (4.3)
\]

The **link throughput** is represented in an observed element of type “qplace:queue” with the regular expression pattern depicted in (4.4). The first matching group contains the link name of the corresponding link. The link also contains different colors, representing the destination applications per flow. The link result is attached as nested element of a flow element in the generic analysis result. The throughput metrics and traffic descriptions for input and output bandwidth are extracted and converted as previously described for switching and interface throughput.

\[
\text{Link}_{(.*}_[\text{\d}]+.\text{maximum_bandwidth}_[\text{\d}]+ \quad (4.4)
\]
5. Adaptation Process

The models described in Chapter 4 are fundamental for adapting and optimizing networks. Section 4.3 introduced the adaptation points, which defines the degrees of freedom and what could be adapted. However, the model do no specify how the adaptation should be executed, i.e., an algorithmic approach to find alternative configurations. Detecting SLA violations, finding proper adaptations and evaluating the resulting models is scope of the adaptation process. An abstract overview about the adaptation process approach, which is inspired by several concepts, is depicted in Figure 5.1.

![Diagram of adaptation process](image-url)

Figure 5.1.: Overview of adaptation process.
The iterative MAPE-K design enables a responsive optimization of the network infrastructure. The network model is monitored and analyzed by an Analysis. SLA violation detection and Solving model collection are responsible to detect appropriate solutions. If no SLA violations can be detected in the SLA violation detection, no (further) adaptation are required and the model is identified by the solving model collector as an appropriate solution. Otherwise multiple target-oriented adaptation tactics are selected through a strategy selection. This branching S/T/A paradigm enables tracking multiple alternative solutions. The generated solutions have to pass multiple filter modules to bound useless branches, which should be excluded from further adaptations. After filtering, the candidate models repeat the iterative MAPE-K adaptation process for analysis and possible further adaptations, until they solve the SLA violation or are bounded.

These concepts including techniques, algorithms and paradigms are described in Section 5.1 more in detail. In the adaptation process these concepts are implemented in different modules, which are introduced in Section 5.2. Finding alternative configurations for parameters and entities is handled in tactics, which are explained in Section 5.3. Section 5.4 summarizes the adaptation process and concludes this chapter.

5.1. Concepts

Several existing adaptation processes, like the adaptation framework for DML, use existing paradigms on how to plan adaptations. Search algorithms play also an important role, as sometime multiple adaptation candidates are available, which create a search space of possible solutions. Additionally, design patterns from software development should be considered when designing an adaptation process.

This section describes the adaptation process with its used concepts. In Section 5.1.1 the iterative MAPE architecture is introduced as basis for the adaptation process. How the idea of S/T/A from the DML adaptation framework for a target oriented adaptation influences the adaptation process is explained in Section 5.1.2. The selected branch and bound algorithm to track multiple candidate solutions is described in Section 5.1.3. Section 5.1.4 introduces a modularization and pipelining concept, on how the adaptation process can be separated into meaningful modules and connecting these together. Section 5.3 deals with the black box modules, called tactics, for the adaptation algorithms. Section 5.4 concludes this section.

5.1.1. MAPE-K Adaptation Control Loop

Scope of the adaptation process is to find alternative configurations which do not violate the agreed SLAs and respect the valid alternative parameters and entities, specified in the adaptation points defined in Section 4.3. If an adaptation is required it is sometimes difficult to decide which alternative configuration should be chosen if multiple exist. The most parameters can be ordered by their performance description, like a sequence of different throughputs of 10 Mbps, 100 Mbps, and 1 Gbps. Also descending sequences like latency exist, where smaller values improves the performance more than higher values.

With knowledge of the order of the available parameters the value with the highest performance gain could be chosen. But this will often result in over-provisioning with unnecessarily high costs. Instead of choosing the parameter with the highest performance gain, the parameter with the lowest cost - considering the multi-dimensional cost model - should be selected. Although the performance gain can be interpreted from the parameter value, it is difficult to predict the overall performance gain in complex scenarios and filter parameters, which will solve the SLA violation.
A better approach is to filter the alternatives for parameters, which will result in a possible unknown performance gain and try the parameter with the lowest cost. Afterwards, the overall system performance is evaluated again and if the same SLA violation still occurs, the next parameter from the list is chosen. This process is iteratively repeated until a solution is found or there are no more parameters which will improve the performance. One major characteristic of this approach is that the parameter adaptation is not executed on the real system, instead only the model will be changed and evaluated. This prevents disruption of the real network operation.

The described approach by analyzing and iteratively selecting the next value from a filtered and sorted list can be described as MAPE-K adaptation control loop. The MAPE-K adaptation control loop with its phases monitoring, analyzing, planing, and executing supplemented by the additional knowledge component is depicted in Figure 5.2. The general characteristics of MAPE-K are introduced in Section 2.9. In addition to parameter adaptation, MAPE-K control loop facilitates entity adaptation, i.e., replacing entities. Sorting alternative entities by their performance impact is a challenge, as they usually contain sets of alternative parameters, which influence the performance of the entity. How multiple alternative entities are handled will be explained later.

![Figure 5.2: MAPE-K adaptation control loop](image)

The MAPE-K adaptation control loop is used in a model-based environment within the adaptation process. An additional outer control loop can be used to observe a real system and react on structure, traffic and configuration changes on runtime. The inner MAPE-K control loop for discovering valid alternative configurations, represented by the introduced adaptation approach, is part of the plan phase of the outer observing control loop. The interaction between these two control loops is depicted in Figure 5.3.

Discovering valid, cost optimal alternative configuration through a model-based simulation approach is contribution of this thesis. An outer control loop for observing and interacting with a real network is out of scope of this work. This would especially include techniques to detect structural and configuration changes, extract the current workload profile and executing the suggested adaptations through an API.

How the four phases and the knowledge component of the MAPE-K adaptation control loop are mapped to the DNI adaptation process is described in the following.

### 5.1.1.1. Monitor

Monitoring is the first phase in the iterative adaptation control loop. This phase is responsible to acquire topology information, the used entities, current parameter assignments, configuration settings, routes and the current workload. This information
are contained in a DNI network infrastructure model, which is required as input model for the adaptation process. Generation of the model instance through automatic or semi-automatic approaches like reading configuration settings through network APIs or extraction the workload profile is out of scope of this work.

The input model has not necessarily be in accordance with the real network. Using different network components, alternative parameter allocations, and altered configuration settings can be used for what-if analysis. Especially including a modified traffic description, e.g., as a result from a forecaster, enables a proactive approach to react before a SLA violation occurs. The origin of the modeled information is immaterial for the adaptation process and enables a flexible usage of the adaptation process, with a DNI network infrastructure as input model.

5.1.1.2. Analyze

The analyze phase is responsible for network performance evaluation, detection of SLA violations and collection of solving models, which contains no violations. DNI models can be analyzed by different solvers to get performance predictions, i.e., about existing bottlenecks per traffic flow and traversed entities. The in Section 2.3 introduced SimQPN is an example for such a solver. Interpreting analysis result and detection of bottlenecks is not sufficient to become aware of SLA violations. The SLAs have to be accessible for analysis to evaluate the simulation results against the specified objectives. Every transmission is limited by a bottleneck, which could be beside the network also the sender or receiver. Only if the performance of the bottleneck is below the agreed value, defined in the objective, a SLA violation occurs. A detailed output of the violated objectives facilitates a sophisticated and target-oriented adaptation process.

If no violations are detected the model is declared as a solving model, which does not contain any SLA violations, evaluated by a previously executed simulation. In this case the iteratively created adaptation plan is put into the solving model repository. If the initial model does not violate any SLAs, an empty adaptation plan without any adaptation action is output to claim that no SLA violations could be detected and no adaptation is required.
5.1.1.3. Plan

Scope of the plan phase is to suggest network adaptations which would possibly solve existing SLA violations. The planning phase requires several input models. The DNI network infrastructure represents the original state of the network without any adaptations. The adaptation plan contains the already discovered adaptation actions. Through a transformation the adapted network infrastructure can be generated by the original network infrastructure and the adaptation plan as input. The adaptation points specify, which parameters and entities can be adapted and defines the valid alternative configurations. Providing the violated SLAs and the analysis result to the planning phase, enables selection of target-oriented adaptations related to the bottleneck, instead of disoriented adaptations.

In the planning phase, different adaptations, possibly in different dimensions, can be generated through one or multiple algorithms. These algorithms are called tactics, explained later more in detail. An example for a planning process is a link bottleneck in a network. The violated objective is detected in the analysis phase and is passed to the planning phase. By knowledge of the violated objective and the exact analysis result, the link, representing the bottleneck, can be identified. The adaptation algorithms are able to search for alternatives to solve the violation. Upgrading the link or redirecting the traffic over another path are two possible solutions to reduce link load. These adaptation actions are specified as concrete adaptation instructions and are added each to adaptation plan.

Furthermore the planning phase contains several filtering mechanisms to bound candidates as early as possible if, for example, predefined maximal costs are exceeded. The filter modules enables canceling the adaptation process for models, which are currently and in future always senseless.

Output of the plan phase are one or multiple adaptation plans, based on the adaptation plan provided as input model. Also no output is possible, if no adaptations can be discovered, e.g., if all adaptations are already applied or no appropriate adaptation points are defined.

5.1.1.4. Execute

The execution phase is not explicitly delegated to a module in the DNI adaptation process. The original scope of the execution phase is applying the scheduled adaptations to the real system, e.g., via APIs. As the adaptation process bases on a network model, the adaptations can only be executed on this. Instead of applying the adaptation actions directly to the network infrastructure, the actions are collected in the adaptation plan, as already explained in Section 4.5. If the adapted model is required on some places in the adaptation process, e.g., in analysis, the adapted model is generated on the fly through a transformation by applying the adaptation actions of the adaptation plan on the original model.

5.1.1.5. Knowledge

The knowledge component represents a data repository and is no phase in the adaptation control loop. Each phase in the control loop has read and write access to the knowledge component to write data for further adaptations or other tactics on it. In the adaptation process the original DNI network infrastructure, the adaptation points, the SLAs, the user constraints (max. number of adaptation, max. costs) and some module-depended data are stored in the knowledge component.
5.1.2. Approach of Strategy / Tactic / Action

The adaptation process changes an existing configuration, respecting the alternatives defined in the adaptation points, to remedy SLA violations. But usually there would be several possibilities of how to adapt a network, and only a few of them would solve the SLA violation. The adaptation has to effect the bottleneck directly or indirectly. Upgrading a device with a bottleneck would directly effect the violation, while redirecting the traffic over another link would indirectly effect the violation, for example. There would be would be several adaptation actions, which are useless for a specific SLA violation, e.g., improving the latency when the bandwidth is identified as bottleneck.

One approach is to try all possible adaptations via brute force. Although the adaptation process design could handle such a brute force approach and an optimal solution would be found, this approach does not scale well on large models with several alternative configurations. Especially analyzing all candidate models would take a long time, which makes the brute force approach unusable for large network models in practical.

A target-oriented adaptation, where only entities effecting the bottleneck are changed, is a better approach. The adaptation should be selected based on the violated SLAs and analysis result. Huber [27] introduced a Strategies/Tactics/Actions (S/T/A) approach as explained in Section 3.1. Inspired by this a similar approach is established for the target-oriented adaptation of DNI network infrastructure models. How the three parts strategy, tactic, and action are defined in detail and how they are used in the adaptation process is explained in the following.

5.1.2.1. Strategy

A strategy describes the origin of a SLA violation. Based on the violated objectives one or multiple strategies are selected. The strategy specifies on high-level, which parameter or entity has to be directly or indirectly effected to remedy a bottleneck. A strategy is selected based on the detected SLA violations. Currently the adaptation process supports the strategies OPTIMIZE_SWITCHING THROUGHPUT, OPTIMIZE_INTERFACE THROUGHPUT, and OPTIMIZE_LINK THROUGHPUT which are triggered by the objectives ObjectiveLinkThroughput, ObjectiveSwitchingThroughput, and ObjectiveNetworkInterfaceThroughput. The mapping between the objectives and the strategies is illustrated in Figure 5.4. The current mapping references a single strategy by each objectives, but also multiple strategies can be triggered by a violated objective.

Figure 5.4.: Objective to strategy mapping in DNI adaptation process.

A strategy itself contains no algorithm to find alternative configurations, instead a strategy maps to one or multiple algorithms, the so called tactics. No knowledge about the tactics, especially no weighting between different tactics to select the potentially best tactic, is required. The tactics have to implement and provide some interfaces, but the implementation of the algorithm is independent from the remaining adaptation process. As a strategy is triggered, it simply invokes all tactics, mapped by it.
For performance reasons currently only one violated objective is considered for strategy selection. This feature addresses the violations step by step instead of triggering several strategies in parallel. As each violated objective has to be remedied, it is irrelevant if the violations are removed in parallel or subsequently. The parallel approach would also work, but without any benefits.

5.1.2.2. Tactic

A tactic contains an algorithm to find alternative configurations. A tactic is triggered by one or multiple strategies and should have a single scope. While strategies define high-level goals, e.g., optimizing a bottleneck on a network interface, tactics contain implementations of how this goal could be achieved. This approach enables a separation between goals and adaptation algorithms. In the example bottleneck on a network interface, one tactic could try to improve the interface performance, while another tactic could be used to replace the interface. A tactic could be invoked by multiple strategies, if the scope of the tactic is useful for multiple violations and strategies. Replacing a switch for example is a valid operation to change the switching performance and could also be helpful to adapt an interface, as the interfaces are related to the switch.

The mapping between strategies and tactics should be customizable, e.g., through a configuration file. An example mapping is depicted in Figure 5.5. By the modular design, the adaptation process can be extended by additional, maybe more sophisticated, tactics. Current tactics are implemented in Java, but as the tactics are seen as black-box algorithms, also tactics in other programming languages or webservices can be invoked via adapters.

As the tactics are triggered in parallel no weighting function to select the potentially best tactic is required. This multi-tactic and extensible approach is one of the strongest aspect of the adaptation process. Each tactic could output one or multiple adaptation plans.

5.1.2.3. Action

An adaptation action is defined as single atomic instruction in the adaptation plan. It describes which entity or parameter has to be changed, typically via UID reference and specification of the new value. By an APIs to the real network, the adaptation actions can be applied to an existing network structure. This technical part is out of scope of this work. The in Section 4.7 described transformation applies the adaptation actions of an adaptation plan to a DNI network infrastructure.

5.1.2.4. Comparison to S/T/A approach of DML

The DML adaptation process also uses a S/T/A approach, however there are some differences between the approaches of DNI and DML. Both adaptation processes use
strategies to achieve objectives. In DML, an event triggers a strategy, while in DNI a violated SLA invokes it. In DNI, an event could be generated for each violated objective as well, so the triggering mechanism is similar. In the DNI adaptation process, strategies are lightweight mappings without any complex functionality, which simply invoke the associated tactics. In the DML adaptation process a strategy selects one tactic via a weighting function to select the probably best tactic. The tactics are used for a certain purpose in DNI as well as in DML. The implementation of the tactics differ.

While tactics in the DNI adaptation process are seen as black-box algorithms, tactics in DML are also described by models. The DNI approach enables defining additional tactics model-based, but this limits the capabilities of tactics to the expressiveness of the model. It could be challenging to specify all algorithms, e.g., routing algorithms from graph theory, on model level. The DNI adaptation process is independent from the used programming language and implementation of adaptation algorithms. DNI as well as DML uses the same definition of an adaptation action. In both adaptation processes an action describes an atomic element on the lowest level of the adaptation process hierarchy and represents the execution of an adaptation operation on model or real system. Finally, both adaptation processes do not implement technical details.

5.1.2.5. Summary

The S/T/A approach enables a target-oriented adaptation, by only adapting entities and parameters, effecting the violated objective directly or indirectly. The DNI adaptation process is inspired by the S/T/A approach used in DML, but the approaches differ especially on how the tactics are implemented and invoked. The S/T/A approach enables a clear separation of concerns and responsibilities.

5.1.3. Branch and Bound Optimization Algorithm

The in Section 5.1.2 introduced S/T/A approach triggers multiple strategies and multiple tactics. Additionally, each tactic can provide multiple candidate models, each represented by an adaptation plan. The introduced multiplicity and diversity of solution candidates has to be handled. One approach is using a search algorithm with backtracking. But even if a solution was found, other search branches have to be tracked, as they could possible result in a more optimal solution as the already found one.

The DNI adaptation process therefore uses a branch and bound algorithm, as described in Section 2.8, to track all branches in parallel. The candidate models, usually adapted in different dimensions, are tracked in parallel. This enables an efficient multi-threaded implementation, avoids complex backtracking and facilitates finding solutions with less adaptation operations earlier. Beside the branching, the bounding component is the opposite key feature of branch and bound. A branch can be bound if it can be ensured, that all future from this branch established branches do not generate a solution or do not generate a better solution than an already existing one. How the two key features, branching and bounding, are used in the DNI adaptation process is described in the following.

5.1.3.1. Branching

The adaptation process starts on a single branch with an empty adaptation plan, the so called root branch. Every time a new branch is forked off, the adaptation plan from the parent branch is copied into the new branch. The S/T/A approach allows branching on the following levels.
5.1. Concepts

**Strategy selection** The strategy selection bases on violated SLAs as described in Section 5.1.2.1. As multiple objectives are violated, or if one objective is mapped to multiple strategies, new branches are established from the input branch of the strategy selection.

**Tactic execution** Each strategy can trigger multiple tactics, usually with different adaptation scopes as introduced in Section 5.1.2.2. Each triggered tactic establishes a new branch, with a copy of the adaptation plan of the parent branch.

**Tactic** Also a tactic itself can establish multiple branches. For example if an entity has multiple parameters and it could not be predicted which parameter adaptation would solve the SLA violation, the tactic is allowed to generate multiple branches, each with another parameter adaptation.

Figure 5.6 illustrates the branching approach of the DNI adaptation process. The input branch is branched through the strategy selection to Strategy A and Strategy C. These strategies map to Tactic 1, Tactic 3, and Tactic 4. It should be noticed that Tactic 3 is triggered twice, but through a smart invocation of the tactic modules by the tactic executor, this tactic is only called once. The tactics could branch again, i.e., to adapt multiple dimensions like independent parameters or multiple alternative entities.

![Figure 5.6: Example branching by strategy and action in DNI adaptation process.](image)

**5.1.3.2. Bounding**

Establishing multiple branches facilitates tracking several possible solutions in different dimensions. But tracking each branch until it results in a solving model or all alternative configurations are exhausted, results in a large amount of branches and requires a large number of time consuming simulations. The bounding is responsible of cutting of useless branches to avoid unnecessary simulations and restricting the number of overall branches.

Bounding a branch is only valid if either the adaptation branch would in future never generate a valid solution, e.g., by violating user constraints, or all future adaptations on this branch would result in higher costs than an already found solution. Bounding is realized in several cases, explained in the following.
No violations This represents the trivial case. If no SLA violations could be detected the corresponding adaptation plan is identified as solving model and the adaptation branch is bounded as all further adaptations would result in higher costs. However, the other existing parallel tracked branches, have to be tracked furthermore as they could possible result in a better solution.

Cost constraint violation The user can specify a limit for each cost dimension. If an adaptation branch exceeds a cost constraint, all subsequent adaptations would result in higher costs, as the alternative parameters are ordered by costs. Such branches can be bounded.

Cost optimality If an adaptation branch exceeds the costs in all dimensions related to an already found solving model, this branch can be bounded. As the subsequent branches will result in higher costs, they would not cheaper.

Number of adaptations A limit for the number of adaptation operations can be defined by the user. As adaptation actions can only be modified or added and not be removed, an adaptation branch exceeding the maximum number of adaptations can be bounded, as all subsequent branches would have the same amount or more, but not less, adaptation actions.

Redundancy Through the flexible branching approach and based on some technical details, different adaptation branches could sometimes result in identical adaptation plans. As all subsequent adaptations are also identically from this point on, all redundant branches can be bounded.

No further alternatives If a tactic does not find further alternative configurations, e.g., all parameters are already set to the value with the highest performance, the branch can be bounded.

5.1.3.3. Summary

The branch and bound algorithm facilitates handling multi-dimensional adaptation operations and tracking branches for each cost dimension. Adaptation branches which are currently and in future senseless, can easily be bounded. By the bounding approach no complete solution space will be generated, but all cost-optimal solutions will be found.

5.1.4. Modularization and Pipelining

The adaptation process, introduced in previous sections of this chapter, has to deal with several tasks. For an independent development, easy maintenance and flexible extensibility these tasks should be partitioned into multiple modules. These modules and the interfaces between them, are described in the following.

5.1.4.1. Modularization

The concerns of the adaptation process like analysis, violation detection, strategy selection, tactic execution and several bounding functions can be separated into independent modules. This divides the coarse-grained phases of the MAPE-K control loop into multiple fine-grained sub-phases. Using the same input and output data structure for each module, i.e., providing and requiring the same interfaces, facilitates inserting additional modules, removing modules and reordering their sequence. The following modules can be identified for the adaptation process, which will described in detail in Section 5.2.

- Analyzer
- Agreement Violation Detector
5.1. Concepts

- Solving Model Collector
- Strategy Selector
- Tactic Executor
- Redundancy Eliminator
- Maximum Adaptation Supervisor
- Cost Constraint Bounder
- Cost Optimization Bounder

5.1.4.2. Pipelining

The modules of the adaptation process have to be connected to each other, to build a pipeline and enable passing the adaptation branch from one module to the next one. As the adaptation process works iterative, the last module has to be linked to the first one. The iterative adaptation process is depicted in Figure 5.7. To link each module to another, the chain of responsibility pattern is used, where each module knows its successor and passes the output model to the next module via a predefined interface [37].

![Pipeline of modules used in adaptation process.](image)

Figure 5.7.: Pipeline of modules used in adaptation process.

The used pipeline for the adaptation process is no single pipeline, as each module can generate multiple branches. To enable this multiplicity, the interface of the successor is
invoked once for each branch. If a branch is bounded, the successor is not called and the processing for the corresponding branch ends.

Figure 5.8 illustrates the pipeline in conjunction with the branch and bound approach. The root branch, denoted as “0” is forced into three branches, “1”, “2”, and “3”. Tactic A generates two output branches “1a” and “1b”, while tactic B and C result each in a single branch. After tactic execution these branches are tracked independently from each other and are supplied to the bounding modules, illustrated as a single abstract Filter in the figure. Branches “1a”, “1b”, and “3” pass the filter, while branch “2” is bounded. The resulting branches rerun the adaptation process iteratively and could be bounded or branched again in additional iterations.

Assessment of Goals

In Section 1.4 the following goal and two research questions have been defined.

**Goal 6:** Develop a process for the adaptation of DNI models.

- **RQ 6.1:** How can the workflow be modeled?
- **RQ 6.2:** How could the amount of model candidates be limited?

To answer research question RQ 6.1, all in Section 5.1 introduced concepts have to be considered. The workflow is modeled by a combination of several concepts. At first, a MAPE-K adaptation control loop, described in Section 5.1.1 is used for an iterative adaptation and dividing the adaptation process into multiple phases. The second included concept is the S/T/A approach, which introduces a sophisticated selection of
adaptation algorithms and enables a target-oriented adaptation, by only adapting en-
tities and parameter, which potentially could solve the SLA violation. The S/T/A ap-
proach is explained in detail in Section 5.1.2. Thirdly, a Branch and Bound algorithm is
used to track multiple possible solutions in multiple adaptation branches. This concept
also introduces a bounding approach, to cut off branches, which will never generate a
cost-optimal solution. The concept of Branch and Bound is introduced in Section 5.1.3.
At last, a modularization and pipeline, representing a chain of responsibility is used, to
separate the concerns of the adaptation process and connecting them together. In
Section 5.1.4 this concept is explained more in detail.
Research question RQ 6.2 is partially answered in Section 5.1. The S/T/A approach,
introduced in Section 5.1.2 reduces the number of candidate models, as candidate models
are only generated for adaptations, which potentially solve the SLA violation. The filter
modules of the Branch and Bound algorithm also limits the number of candidate models.
As the reasons for bounding branches is explained later, this research question is only
partially answered.
Research question RQ 6.1 was answered in this section, but the remainder of RQ 6.2 will
be answered later. Therefore goal 6 is not fulfilled yet.

5.2. Modules

The previous section introduced the used concepts of the adaptation process. One of
the key features of the MAPE-K adaptation control loop and S/T/A approach is the divi-
sion of the adaptation process into multiple phases and modules. The DNI adaptation
phase uses a more fine-grained partitioning for a stronger separation of concerns. The
responsibility and the processing of these modules is explained in this section.

5.2.1. Analyzer

Scope of the analyzer module is to execute a performance analysis on a DNI network
infrastructure model. The analysis result is the basis for the later executed SLA violation
detection. The analysis result is added to the adaptation branch to be accessible by
subsequent modules.

The existing solvers for executing the analysis consume an instance of a DNI network
infrastructure. As the already determined adaptation actions on each branch are stored
in an adaptation plan instead of changing the original network infrastructure model
directly, the adaptation plan has to be applied through a transformation to the original
infrastructure model, to get the adapted DNI model which should be analyzed. This
transformation is described in Section 4.7.

The analysis module itself needs no knowledge about the used analysis method. The anal-
ysis should be called via a common interface, so that different solvers such as SimQPN
can be used. As currently only a transformation from SimQPN result to the generic
analysis result as described in Section 4.8 exists, the analysis is currently limited to
SimQPN. As soon as adapters to the generic analysis result from other solvers exist,
these could also be integrated.

5.2.2. Agreement Violation Detector

Scope of the agreement violation detector is to validate the SLAs against the analysis
result, appended by the analyzer to the input adaptation branch. The objectives can
observe different elements of the analysis result like flows, nodes, interfaces or combi-
nations of them. The metrics defined in the objectives are compared to the analysis
result. If the predicted value of the analysis result is less than the agreed value in the objective, the objective and the corresponding SLA is violated. The violated objectives are marked in the adaptation branch, so that the information can be used in further modules to facilitate a target-oriented adaptation process.

A SLA violation can only be detected, if the violation currently occurs. If for example a bandwidth of 100 Mbps is guaranteed and only 80 percents are currently used, it cannot be predicted if the remaining 20 percents are available. To ensure the availability of a specific performance, the traffic has to be modeled like the desired performance value. However, such an approach would not work in overbooked networks, as it is similar to providing dedicated resources.

Assessment of Goals

In Section 1.4 the following goal and two research questions has been defined.

Goal 4: Develop a component for SLA violation detection. The component has to detect SLA violations based on a previous analysis (simulation, or analytical method).

RQ 4.1: How can SLAs be modeled?

RQ 4.2: How can SLA violations be detected?

Section 5.2.2 answers research question RQ 4.2, of how SLA violations can be detected. Violations could only be detected, if they occur during the simulation. Only if the resource is overloaded, e.g., packets are lost, it can be concluded that the utilization is over a threshold. These thresholds are specified in the objectives in the SLA model, introduced in Section 4.4. As a violation occurs and the output of a resource is less then the agreed threshold, a SLA violation is detected.

As research question RQ 4.1 has already been answered in Section 4.4 and the remaining research question RQ 4.2 is answered above, the related goal 4 is declared as fulfilled. The component for the SLA violation detection is developed by the in Chapter 6 described implementation.

5.2.3. Solving Model Collector

Scope of the solving model collector is collecting the adaptation plans of non SLA violating branches. This module is configured as successor of the agreement violation detector.

If an adaptation branch contains no SLA violations, determined by the previous executed agreement violation detector, the corresponding adaptation plan is identified as solving model and is suggested as solution, as illustrated in Figure 5.9. In this case, the branch is bounded, as subsequent adaptations on this branch will result in higher cost than this branch. If incoming branches in the solving model collector contain SLA violations, the branch is forwarded to the next module without any action.

5.2.4. Strategy Selector

Scope of the strategy selector is the target-oriented selection of strategies and tactics. This module uses the violated objectives to select one or multiple strategies, based on a mapping, as described in Section 5.1.2.1. The strategies classify the type of SLA violation, without detailed knowledge, of how to solve the violation.

After selected strategies are triggered, the corresponding tactics are invoked, as explained in Section 5.1.2.2. This mapping could be manually configured and is extensible for additional tactics. If a tactic is called by multiple strategies on a branch, the tactic is executed only once. Avoiding redundant tactic calls for identical adaptation plans is cause for the merged module for strategy and tactic selection.
5.2.5. Tactic Executor

Scope of the tactic executor is calling the tactics, specified by the previous strategy selection module. The tactic executor is no adaptation algorithm and does especially not contain any logic for selecting alternative configurations. Instead of this, the tactic executor is responsible to branch to multiple tactics and collect the multiple resulting adaptation branches from tactics. This process is depicted in the sequence diagram in Figure 5.10. It is also possible that a tactic does not provide a resulting adaptation plan, which means that the tactic is unable to handle the problem, e.g., by missing alternatives.

5.2.6. Redundancy Eliminator

Scope of the redundancy eliminator is bounding redundant adaptation branches. Through the nature of multiple, independent, adaptable parameters and the multi-dimensional cost model, tactics could output identical adaptation branches. Identical adaptation branches generate unnecessary load for the adaptation process and should be avoided.
Figure 5.11 illustrates such an example in an abstract manner. In the first iteration, the values of parameter A and parameter B are increased in two separate, independent adaptation branches. Each of the adaptation branches are analyzed in a second iteration and it is assumed that the previous optimization does not solve the SLA violation. Therefore, on both branches, the other one of the previously adapted parameter is changed to the increased value. This results in identical adaptation plans in the second iteration, which have been generated in reverse order. As identical adaptation plans result in identical solutions, and only produce increased, unnecessary load on the adaptation process, one of them is bounded in the redundancy eliminator module. This approach also applies to identical adaptation plans, generated in different iterations of the adaptation process.

Figure 5.11.: Example of redundancy elimination caused on identical adaptation branches.

Detection for redundant adaptation branches is done by identifying identical adaptation plans. As all branches have to pass the redundancy elimination module, a copy of the adaptation plan is globally stored in a collection of adaptation plans, called *history*. Before storing the adaptation plan in the history, the module checks if the plan with
identical actions is already contained in the history. If the adaptation plan already exists, the branch is identified as redundant branch and is bounded. If the branch contains a new adaptation plan, a copy is stored in the history and the branch is forwarded to the successor module. The decision logic and the interaction with the adaptation plan history is depicted in Figure 5.12.

![Decision logic and interaction with history in redundancy eliminator module.](image)

**Figure 5.12.:** Decision logic and interaction with history in redundancy eliminator module.

### 5.2.7. Maximum Adaptation Supervisor

Scope of the *maximum adaptation supervisor* is to bound branches, which exceed the maximum number of adaptations, specified in the configuration. The parameter of maximum adaptations is optional and can also be left unconfigured to allow for an unlimited number of adaptations. The number of adaptations on the adaptation branch could be determined from the adaptation plan. If the maximum number of adaptations is exceeded, the branch is bounded.

The number of adaptation actions is independent from the number of iterations an adaptation branch has passed. For entity adaptations, especially with nested entities and contained parameters, often multiple adaptation actions are generated in one iteration. In contrast, multiple iterations could rewrite an already contained adaptation, which would not increase the number of adaptation actions, which have finally to be executed. As no adaptation actions could be removed, the number of adaptations increases or remain unchanged in subsequent iterations. Therefore, a branch, exceeding the maximum number of adaptations, can be bounded, as all subsequent branches could only result in an increasing or equal number of adaptation actions.

### 5.2.8. Cost Constraint Bounder

Scope of the *cost constraint bounder* is to cut off branches, which exceed the maximum costs, defined in the configuration. This module works similar to the maximum adaptation supervisor and the specified limits are also optional. In contrast to the maximum number of adaptations which only contains one integer value, the cost model is multi-dimensional.

To work correctly, this module requires that each subsequent adaptation branch has equal or greater costs than the current one. The tactics therefore have to select the
next cost optimal alternative from the alternatives with improved performance, instead of selecting a arbitrary alternative, maybe based on performance gain expectations. If the alternative configurations, determined by the next cost optimal selection, differs for the different cost dimensions, a separate branch for each cost type is generated. This approach enables to find the cost optimal solution, as for all dimensions the cost optimality is ensured.

The costs can be determined by iterating over the adaptation actions and extracting the costs from the corresponding adaptable entity from the adaptation points model. As this process can be implemented with less performance effort, the costs are calculated on the fly for each adaptation branch. This avoids storing the calculated costs on each adaptation branch.

5.2.9. Cost Optimization Bounder

Scope of the cost optimization bounder is to abort the adaptation of branches, if solving models with lower costs are already found. A branch exceeding the costs in all dimensions compared to an already found solution, would never be cost optimal, as the costs will increase further more or will remain unchanged. It is important that all cost dimensions have to be considered for the cost analysis, as there could be multiple cost-optimal solutions, building the so called pareto front.

To decide if a solution with lower costs already exist, the module is linked to the solving model collector. Every time the solving model collector identifies a model as solution, the costs are calculated and are stored in a shared collection. When an adaptation branch passes the cost optimization bounder module, the costs for this branch are calculated and it is determined, if there is a solving model with lower costs in all dimensions in the collection. If this is true, the branch is bounded in the module. Otherwise, the branch continues the iterative adaptation process.

The scenario of cost optimization bounding is depicted in Figure 5.13. Adaptation plan 3 has to pass the cost optimization bounder module. The module checks, if an adaptation plan with lower costs already exists in the solving models. As adaptation plan 1 has lower costs for “investment” and “handling time” in this example, adaptation plan 3 is bounded. Assuming adaptation plan 1 does not exist, adaptation plan 3 would continue the adaptation process, as adaptation plan 2 has lower investment costs, but requires more handling time. Therefore, adaptation plan 3 could be a cost-optimal solution related to the cost type “handling time”, compared to adaptation plan 2.

Assessment of Goals

In Section 1.4 the following goals and research questions have been defined.

Goal 6: Develop a process for the adaptation of DNI models.

RQ 6.1: How can the workflow be modeled?

RQ 6.2: How could the amount of model candidates be limited?

Goal 7: Develop a framework for the adaptation of DNI models.

RQ 7.1: How could the framework separated into multiple modules?

RQ 7.2: Which interfaces should the modules provide and require?

Research question RQ 6.1 has already been answered in Section 5.1.4.2.

For research question RQ 6.2 the target-oriented adaptation of S/T/A and the filter modules of the Branch and Bound have been identified as limiting factors. Details of the
5.3. Tactics

Tactics are algorithms to discover and select adaptations, based on the violated SLAs and degrees of freedom, specified in the adaptation points model. Tactics are called by the tactic executor module as introduced in Section 5.2.5. An adaptation plan, the violated SLAs, and the analysis result are provided as inputs for a tactic. Additionally, a tactic has access to the original DNI network infrastructure model and to the adaptation points model. Output of a tactic is a single adaptation plan, if exactly one adaptation was found, multiple adaptation plans, if multiple adaptations or multiple cost optimal alternatives have been discovered, or nothing, if no valid adaptation was found. The

Figure 5.13.: Example of cost optimization bounding.

bounding modules are provided in the following. The redundancy eliminator, described in Section 5.2.6, which bounds branches with identical adaptation plans. As such branches would result in identical further candidate models, redundant branches can be bounded. The maximum adaptation supervisor, as introduced in Section 5.2.7, bounds branches which exceed the specified number of maximal adaptations. The cost constraint bounder, explained in Section 5.2.8, cuts off branches, which exceed the cost constraints, specified by user. Through the cost-optimality, all subsequent branches would result in higher costs, so that such branches can be bounded. The cost optimization bounder, introduced in Section 5.2.9, bonds branches, if they result in higher costs in all dimensions, as an already found solution. Again, as all subsequent branches would result in higher cost, the children branches of this branch cannot be cost-optimal solutions. This four bounding modules also contribute to a limitation for the candidate models. Research question RQ 6.2 is therefore fully answered.

As both research questions RQ 6.1 and RQ 6.2 are answered, goal 6 is fulfilled. The developed adaptation process in explained in detail in Chapter 5.

Research question RQ 7.1 is answered by Section 5.2. The identified modules are analyzer, agreement violation detector, strategy selector, tactic executor, redundancy eliminator, maximum adaptation supervisor, cost constraint bounder, and cost optimization bounder. The pipeline of these modules is depicted in Figure 5.7.

The research question RQ 7.2 cannot be answered here, so the question stays unanswered and goal 7 is currently not fulfilled.

5.3. Tactics

Tactics are algorithms to discover and select adaptations, based on the violated SLAs and degrees of freedom, specified in the adaptation points model. Tactics are called by the tactic executor module as introduced in Section 5.2.5. An adaptation plan, the violated SLAs, and the analysis result are provided as inputs for a tactic. Additionally, a tactic has access to the original DNI network infrastructure model and to the adaptation points model. Output of a tactic is a single adaptation plan, if exactly one adaptation was found, multiple adaptation plans, if multiple adaptations or multiple cost optimal alternatives have been discovered, or nothing, if no valid adaptation was found. The
output adaptation plan always bases on the input adaptation plan, supplemented by additional adaptation actions, whereby existing adaptation actions can be overwritten.

Tactics are significant to efficiently find a solution. They are used as black box algorithms and are called via an interface. A tactic should focus on one certain adaptation, e.g., optimizing throughput parameters of an entity, or replacing a specific entity. This enables reusability of the tactics by other strategies. For example replacing a switch would be a useful operation to increase the switching throughput as well as to optimize the latency in a switching device. In contrast, it is possible to have multiple tactics, focusing the same adaptation actions. As they are triggered by different strategies in parallel, the resulting candidate models of all invoked tactics are used in the further adaptation process.

Currently, some reference tactics are available, and the adaptation process can be extended by additional tactics in future. The current tactics focus on operations effecting the bottleneck directly, through changing parameters or entities. Adaptation operations with indirect impact on the violated objective, like alternative routes, are future work.

The scope of current tactics can be classified into parameter adaptation and entity replacement. Each type of these operations work similar within a tactic, regardless of which entity or parameter is adapted. These two types and the adaptation of nested entity, which is derived form the entity replacement, are explained in the following.

5.3.1. Parameter Adaptation

Scope of the parameter adaptation is the optimization of parameters within an entity. The parameters are modeled as attributes of an entity and represent configuration parameters or properties in common. The values for an interface throughput or the switching capacity are examples for such parameters. The alternatives for each parameter are specified in the adaptation points model. The containing entity has to reference an entity type, e.g., a specific switch model, as described in Section 4.1.2. The entity type also has to be referenced by a corresponding adaptable entity in the adaptation points model. The adaptable entity contains a set of values for each attribute, which can be used alternatively. By referencing an entity type it is ensured that only alternative values are chosen, which are compatible with the current parameter type of the entity.

Several entities contain multiple attributes, influencing a specific performance property. For example, the switching throughput parameter as well as the switching capacity parameter within an intermediate device have an impact on the switching performance. Sometimes it is unclear which parameter should be adapted to achieve a desired performance gain if multiple parameter exist. In such cases, the tactic benefits from the powerful branching capabilities of the adaptation approach. The tactic can generate multiple branches, one for each parameter optimization.

Selecting an alternative value does not simply mean to increase a value. For some attributes, like latency, smaller values are more preferable. Such logic has to be taken into account by the tactics. Beside the performance gain, the tactics also have to consider the introduced costs for an adaptation. From the alternative values improving the performance, the value with the lowest costs has to be selected. As the adaptation process supports a multidimensional cost model, the alternative parameter with the lowest cost maybe differs for the different cost dimensions. In that case, the tactic also benefits from the branching approach of the adaptation process and has to generate multiple adaptation branches, each for one cost-optimal dimension.

The algorithm for the parameter adaptation is depicted in Figure 5.14. At first, a branch for each parameter generated to adapt the parameters independently. It should be noticed that only parameters, which have impact on the bottleneck, have to be adapted.
and not necessarily all parameters within an entity. In each generated branch the alternative values for the corresponding parameter are extracted from the adaptation points, based on the entity type of the adapted entity. The received set of alternative parameters is firstly filtered to contain only values, which will improve the performance. For a throughput optimization the set will only contain higher bandwidths after filtering, for latency the filter process will only leave smaller values in the set.

After the filtering process, the set of alternative parameters is copied and sorted for each cost-dimension, to have a cost-optimal alternative value as first entry in each list. Now, the adaptation branches, corresponding to each parameter, are branched again, each for one cost type. The adaptation action to upgrade to the first entry of each list is inserted into the adaptation plan of each branch. These branches are collected by the tactic executor and are forwarded to the next module. The tactic does not need to take into account if the alternative values of the different cost-optimal branches are identically. Such duplicates will be eliminated by the in Section 5.2.6 described redundancy elimination module.

5.3.2. Entity Replacement

The previously introduced parameter adaptation is used to change parameters within an entity. Sometimes the available alternative parameters do not provide the desired performance gain so that the original entity is not able to tackle the SLA violation. Instead of changing a parameter, the entity replacement substitutes a whole entity by a compatible alternative entity. Such entity replacement is not necessarily a physical
operation. Beside physical switches also virtual switching instances could be used and also several performance descriptors are modeled as entities as example.

In contrast to the parameter adaptation, it is difficult to determine the next cost-optimal alternative entity from a set, as an alternative entity itself could contain adaptable parameters and other entities to handle this challenge, the current tactics generate a branch for each alternative entity and tracks each of them independently. This enables searching for a solution without any cost or performance predictions on the alternative entity. The cost-optimality is ensured, as solutions on all entities are considered and the cost-optimal solutions can be filtered in a preprocessing from the solving models.

The alternative entities are also specified in the adaptation points. There are two different methods to get the alternatives for an entity. The first method is used for nested entities. Nested entities are entities, which are contained in other entities, e.g., a network interface which is contained in an intermediate node. In case of a nested entity, the alternatives are specified via the entity type of the parent entity, which defines the valid alternatives for its children. In the network interface example, the switch containing the interface defines the alternative interfaces. If a nested entity should be replaced, the parent entity, i.e., the entity type of the parent entity, is determined though the hierarchy of the network infrastructure. Through the entity type the corresponding adaptable entity in the adaptation points can be identified. This adaptable entity specifies its valid nested entities, from which the alternatives could be extracted.

The second method is used for top level entities. Top level entities are not contained by any other entity. Nodes and links are currently the only entities of this type in DNI network infrastructure. A simple approach to replace these entities would be to allow replacing any node by any other node and any link by any other link. But this simple approach would allow replacing a sophisticated firewall through a less secure switch, which are both nodes, or replacing an optical link through a copper connection. To restrict such impractical adaptations, the top level entities are managed in adaptation groups, introduced in Section 4.3. Each entity type could be contained in one group and all other group members are valid alternatives. This enables for example different groups for firewalls and switches, and different groups for copper and fiber optical links.

5.3.3. Nested Entity Replacement

The nested entity replacement is no standalone adaptation type like the previous introduced parameter adaptation or entity replacement. Instead of this, the replacement of nested entities is a sometimes necessary process, when an entity is replaced, which contains other entities. For example, if a switch is replaced, it is not sufficient to only change the entity type and some parameters of the device, also the contained nested entities like the performance description and the interfaces have to be considered.

When replacing an entity, it has to be checked if this entity contains nested entities. If this is true, the nested entities have also to be replaced. Instead of branching to all alternatives, as described for the entity replacement, firstly the default alternative entity is used, as defined in the adaptation points. This avoids massive unnecessary branches, if several alternatives exist. If the default alternative is later identified as bottleneck, the adaptation process branches to all alternatives as described in the entity replacement to tackle this bottleneck.

Assessment of Goals

In Section 1.4 the following goal and two research questions, including one optional question, have been defined.
5.4. Summary

**Goal 5:** Develop a tactic to suggest adaptations.

**RQ 5.1:** Is the tactic optimal?

**RQ 5.2:** What is the runtime of the tactic and which parameters influence it? (optional)

Research question RQ 5.1 has to be considered for both previously described adaptations. Section 5.3.1 describes the adaptation of parameters. To suggest cost-optimal solutions, it is required that the parameter with the lowest costs from the performance improving parameters is selected. This is achieved by a list, ordered by costs. The challenge of sorting by multiple dimensions is managed by creating a branch for each cost type and sort the list in each branch by the corresponding cost type. In Section 5.3.2, the selection of alternative entities is explained. Selection of the cost-optimal entity is challenging, as further adaptation could be executed on the entities, which would result in other costs, as the initial configuration for an entity. To ensure cost-optimality also for entity replacements, multiple branches for each alternative entity are generated and tracked, instead of selecting a single entity. This ensures that the cost-optimal solutions are included in one branch. As the cost optimality is fulfilled for both adaptation types, the answer for research question RQ 5.1 is that the implemented reference tactics suggest cost-optimal solutions. The cost optimality is also evaluated in Section 7.3.3 and Section 7.3.7.

Research question RQ 5.2 has also to be considered for both adaptation types. The runtime for a parameter adaptation is influenced by the number of attributes and alternative parameters. If the entity contains multiple attributes, which could potentially solve the SLA violation, a separate adaptation branch for each of them is generated, which increases the runtime. The required time to filter the list of alternative parameters for performance increasing values, and sorting them by costs, increases with a higher number of alternatives. As the cost-optimal candidates for the cost types differ, additional branches have to be generated to ensure cost optimality. The runtime for an entity replacement strongly depends on the structure of the entity. As an entity replacement usually also requires a parameter adaptation, the previous statements for parameter adaptations can be applied to this type of adaptation. Additionally, the number of alternative entities has to be considered. An increasing number of alternative entities also increases the runtime. For nested entities, additional replacement steps are required. For all of these described influence factors it has to be mentioned that these are all simple algorithms with a limited complexity for the current implemented reference tactics. Compared to the used analysis, the runtime of the tactics is negligible, especially as they are executed in parallel to a simulation.

As the reference tactics are implemented as part of the framework implementation, described in Chapter 6 and research question RQ 5.1 and RQ 5.2 are answered, goal 5 is fulfilled.

**5.4. Summary**

This chapter described the adaptation process for the DNI network structure in detail. The process itself is inspired by the iterative concept of MAPE-K which divides the process in the phases of monitoring, analyzing, planing and execution, supplemented by a knowledge component. The Strategies/Tactics/Actions (S/T/A) approach enables target-oriented adaptation, through only executing algorithms, which have impact on the detected bottleneck. The flexible branch and bound algorithm enables triggering multiple tactics in parallel and allows the tactics to output various possible solutions.
The filter modules bound branches, which will never produce an optimal solution, to limit the number of branches and avoid unnecessary simulations. Modularization of the adaptation process and building a pipeline of these modules enables a clear separation of concerns and cases maintainability. The responsibility and the processing of these modules was described in detail. How tactics determine an alternative configuration, for both cases, parameter adaptation and entity replacement was also explained.

The technical details of the adaptation framework are skipped in this chapter, but are described in Chapter 6.
6. Implementation

The adaptation process for DNI network infrastructure, described in Chapter 5, and the corresponding adaptation points model, introduced in Chapter 4, are implemented as a software artifact, the so called DNI adaptation framework. The framework is written in the Java 8 programming language, as the existing model implementations of DNI network infrastructure as well as the used SimQPN solver are also implemented in Java. The source code of the adaptation framework is public available on the Git repository of the Institute of Computer Science of the University Würzburg. A ready to use JAR-artifact is published on the Descartes Maven Repository at group tools.descartes.dni with artifact ID adaptation-framework. This work references version 3.1.0-SNAPSHOT, which is derived from the version number of the used DNI network infrastructure meta-model.

This chapter describes technical details about the framework implementation and is outlined as following. Section 6.1 depicts the dependencies to other projects. The consumed inputs of the adaptation framework are specified in Section 6.2 and the outputs are explained in Section 6.3. How the Java packages and classes are structured is outlined in Section 6.4. Section 6.5 explains how additional tactics can be included into the adaptation framework. In Section 6.6 a short example of how to run an adaptation process on the framework is given.

6.1. Dependencies

The framework uses existing model implementations, existing solvers and other existing Java artifacts. This sections describes the dependencies to DNI related artifacts and how these dependencies are resolved. Figure 6.1 depicts the dependencies to other DNI related artifacts. Dependencies to other libraries are suppressed in the illustration.

6.1.1. DNI Core Meta-Model

The DNI core meta-model contains the implementation of the DNI network infrastructure model. This artifact provides methods to read a network infrastructure model from a file, convert it to a Java object, and navigating through the object hierarchy. Additionally,
6. Implementation

Figure 6.1.: Dependencies of the DNI adaptation framework to other DNI related artifacts.

Fabrics are included to generate new DNI network infrastructure instances, which are useful for JUnit test development. The DNI core meta-model implementation is published as a Maven artifact with artifact ID \textit{core-metamodel} at group \texttt{tools.descartes.dni} on the Descartes Maven Repository\footnote{https://se4.informatik.uni-wuerzburg.de/nexus/}. The used version for this work is 3.1.0-SNAPSHOT.

6.1.2. DNI Adaptation Points Meta-Model

A Java implementation of the DNI adaptation points is provided by the DNI adaptation points meta-model artifact. As the DNI core meta-model implementation, this artifact also provides method for reading model instances from a file system and converting them into Java objects, including the corresponding getter and setter methods. Fabrics for instantiating models, e.g., for JUnit tests are also included. The DNI adaptation points artifact is a contribution of this work. Like the DNI core meta-model implementation, the DNI adaptation points meta-model implementation is also published on the Descartes Maven Repository\footnote{https://se4.informatik.uni-wuerzburg.de/nexus/} as artifact \textit{adaptation-metamodel} at group \texttt{tools.descartes.dni}. This work bases on version 3.1.0-SNAPSHOT.

6.1.3. SimQPN as Solver for DNI Network Infrastructure

For the analysis of a DNI network infrastructure model, SimQPN as solver is used. As SimQPN consumes a generic input file for analysis and can not handle a DNI network infrastructure model directly, a transformation from DNI model to SimQPN input model is required. This transformation was developed by Rygielski\footnote{https://se4.informatik.uni-wuerzburg.de/nexus/}. The output of a SimQPN analysis is a file in a SimQPN specific structure. To facilitate the usage of different solvers, the SimQPN specific analysis has to be converted to the in Section 4.6 introduced generic analysis result by a transformation as described in Section 4.8. As the adaptation framework should be unaware from solver specific characteristics and data structures, the transformation to SimQPN input file, the simulation call and the transformation to the generic analysis result is factored out into the Maven artifact \texttt{solver-simqpn} at the group \texttt{tools.descartes.dni} on the Descartes Maven Repository\footnote{https://se4.informatik.uni-wuerzburg.de/nexus/}. The used version 3.1.1-SNAPSHOT is linked as dependency into the adaptation framework. The SimQPN simulation Maven artifact \texttt{simqpn-standalone} of the group \texttt{tools.descartes.simqpn} is linked to this DNI specific SimQPN adapter.
Packaging the Maven artifact for the transformation to a SimQPN input file, the abstract simulation call, and the backwards transformation to generic analysis result is contribution of this work, too.

6.1.4. Other Dependencies

In addition to DNI related dependencies the adaptation framework uses other third party libraries. This includes among other things the Apache Log4J library [38] for logging purposes, JUnit libraries [39] for testing, and Java Architecture for XML Binding (JAXB) [40] for converting XML strings, e.g., used in the framework configuration file, to Java objects.

These dependencies are also linked as Maven dependencies to the adaptation framework and are resolved and downloaded automatically on compiling.

6.2. Inputs

The adaptation framework requires some inputs, such as the DNI network infrastructure, the adaptation points and a configuration file. These inputs, especially the parameters of the configuration file, are described in this section.

6.2.1. DNI Network Infrastructure Model

The adaptation framework suggests network adaptations based on an existing network structure. This network structure is specified as a DNI network infrastructure model, described in Section 2.7. By default this model is stored in a XML file, which has to be accessible by the framework. The file path itself does not have to be specified explicitly when running an adaptation process, as the DNI network infrastructure is referenced by a DNI adaptation points model.

6.2.2. DNI Adaptation Points Model

The DNI adaptation points specify the alternative configurations for the used parameters and entities. The adaptation points model is introduced in Section 4.3. As the DNI network infrastructure, this model is typically stored in a XML file. The file has to be accessible by the adaptation framework and the path to this file has to be specified by the setAdaptationPointsModel(File file) method on an adaptation framework instance. The DNI adaptation points references a DNI network infrastructure model, which is automatically loaded when reading the adaptation points from this file.

6.2.3. Configuration File

The adaptation framework has some required and optional settings, which could either be provided as a configuration file or as a Java object, e.g., to be generated programmatically. To set the framework configuration by file, the method setConfiguration(File file) on a framework instance is used. Calling setConfiguration(FrameworkConfiguration frameworkConfiguration) sets an object of type FrameworkConfiguration as framework configuration.

An example configuration file is depicted in Listing A.1. The required and optional configuration parameters are explained in Table 6.1.
Table 6.1.: Configuration parameters of the DNI adaptation framework.

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Default</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound-on-agreement-fulfillment</td>
<td>boolean</td>
<td>true</td>
<td>Setting this to true, bound branches as soon as no further violations occur. To get further solutions, which would not be cost-optimal, turn this parameter to false.</td>
</tr>
<tr>
<td>bound-worse-cost-models</td>
<td>boolean</td>
<td>true</td>
<td>By default, adaptation branches exceeding costs of already found solutions are bounded, as described in Section 5.2.9. Turning this flag to false, deactivates the cost optimization bounding.</td>
</tr>
<tr>
<td>maximum-adaptations</td>
<td>integer</td>
<td>-1</td>
<td>This value specifies the maximum number of adaptation actions, before branches are bounded by the maximum adaptation supervisor module, described in Section 5.2.7. A negative value allows an unlimited number of adaptations.</td>
</tr>
<tr>
<td>solving-adaptationplan-output-dir</td>
<td>string</td>
<td>&lt;none&gt;</td>
<td>Output directory for solving adaptation plans. All existing files in this directory will be removed on framework startup.</td>
</tr>
<tr>
<td>statistic-file</td>
<td>string</td>
<td>&lt;none&gt;</td>
<td>Specifies a file, where the framework statistic should be written after framework execution. The file is overwritten on framework startup.</td>
</tr>
<tr>
<td>strategy-tactic-map</td>
<td>nodes</td>
<td>&lt;empty&gt;</td>
<td>Contains the strategy-tactic-mapping by specifying multiple entries by the syntax <code>&lt;entry strategy=&quot;STRATEGY_NAME&quot; tactic=&quot;TACTIC_CLASS&quot;/&gt;</code>.</td>
</tr>
</tbody>
</table>

6.2.4. Service Level Agreements (SLAs)

Scope of the framework is to solve SLA violations through adaptation of parameters and entities. In order to detect SLA violation, the user has to define these agreements. The supported service level objectives are described in Section 4.4. The SLAs, possible from multiple customers, are handled in a so called agreement repository. This repository can either be submitted as a Java object via the method `setAgreementRepository(AgreementRepository agreementRepository)` or specified as an XML file by calling `setAgreementRepository(File file)` on a framework instance. An example XML file for an agreement repository is depicted in Listing A.2.

Within the root element `agreement-repository` multiple `agreement` elements, e.g., for multiple customers, are allowed. In an `agreement` node the `objective` nodes are defined. Beside the type of the objective, the metric which should be observed has to be specified. The calibration of the DNI network infrastructure and its SimQPN solver currently only supports a metric for minimum throughput. The optional attributes `flow`, `application`, `node`, `link`, and `interface` allow a fine-grained definition on where the objectives should have an effect.

Table 6.2 depicts the available objectives, the corresponding metric, and the optional effect attributes.
Table 6.2.: Service level objectives for DNI adaptation framework.

<table>
<thead>
<tr>
<th>Objective</th>
<th>Metric</th>
<th>Optional Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectiveLinkThroughput</td>
<td>min-throughput</td>
<td>flow, application, link</td>
</tr>
<tr>
<td>ObjectiveNetworkInterface-Throughput</td>
<td>min-throughput</td>
<td>flow, application, node, interface</td>
</tr>
<tr>
<td>ObjectiveSwitchingThroughput</td>
<td>min-throughput</td>
<td>flow, application, node</td>
</tr>
</tbody>
</table>

6.2.5. Cost Constraints

Section 5.2 described the cost bonding module to cut off branches, exceeding a user-defined cost limit. These cost constraints are optional and can either be set as a Java object by calling `setCostConstraints(CostConstraints costConstraints)` method on a framework instance, or defining them as an XML file and specifying the file through invocation of `setCostConstraints(File file)`. Currently, the framework supports the cost types `INVESTMENT` and `HANDLINGTIME`. Specification of each of them is optional and an unset value means no limitation on this cost type.

An example XML file, defining cost constraints for both cost types is depicted at Listing A.3. The root element `cost-constraints` could contain multiple `constraint` nodes, at most one for each cost type. Each `constraint` is specified by the `type` attribute and the `limit` attribute with data type double defining the maximum value. A more detailed description about the cost types and their interpretation can be found at Section 4.2. Table 6.3 summarizes the different cost types.

Table 6.3.: Cost constraints for DNI adaptation framework.

<table>
<thead>
<tr>
<th>Cost Type</th>
<th>Default</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>HANDLINGTIME</td>
<td>unlimited</td>
<td>Maximum number of manual handling time for all scheduled adaptations.</td>
</tr>
<tr>
<td>INVESTMENT</td>
<td>unlimited</td>
<td>Maximum investment (financial) for all scheduled adaptations.</td>
</tr>
</tbody>
</table>

6.3. Outputs

The adaptation framework outputs the solving adaptation plans, which tackle the SLA violations. Beside these solving models, an optional statistic file for an adaptation run can be exported. The output path of both outcomes is specified in the configuration file, described in Section 6.2.3. The interpretation of these outputs is described in this section.

6.3.1. Adaptation Plan

The determined adaptations through the adaptation process are not applied directly to the DNI network infrastructure model, instead they are collected as adaptation actions in the adaptation plan, as described in Section 6.2. Each adaptation branch contains its own adaptation plan and iteratively adds more adaptation actions through the adaptation process until the adaptation plan tackles the SLA violation or the adaptation branch is bounded. The resulting adaptation plans are output as XML files. A sample XML adaptation plan can be found at Listing B.4.

Through a transformation, introduced in Section 4.7, the adaptation plan can be applied to the original DNI network infrastructure to get the adapted DNI model. Additionally,
the adaptation plan can be used for APIs to apply the scheduled adaptations on a real network.

The root element **adaptationPlan** contains an attribute **branchId**, which specifies an unique ID for each adaptation branch and its associated adaptation plan. The nodes in the adaptation plan correspond to the adaptation plan model described in Section 4.5. The sequence of adaptation actions is ordered by associated adaptation type and the origin [UID] of the adapted entity.

### 6.3.2. Statistic

Beside the resulting adaptation plans a statistics file of the adaptation process can be exported. This file is optional and is primary significant for evaluation purposes. The statistic contains some overall measurement values, and statistics for several framework modules. An example statistic output is depicted at Listing B.5. The observed values are explained in Table 6.4.

<table>
<thead>
<tr>
<th>Module</th>
<th>Parameter</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation Framework (global)</td>
<td>duration</td>
<td>duration of the overall adaptation process in seconds.</td>
</tr>
<tr>
<td>Adaptation Framework (global)</td>
<td>solving models</td>
<td>number of adaptation plans, solving the [SLA] violation.</td>
</tr>
<tr>
<td>Analyzer</td>
<td>number of analysis</td>
<td>number of executed analysis, e.g., simulations.</td>
</tr>
<tr>
<td>Analyzer</td>
<td>avg. analysis duration</td>
<td>average duration for each analysis in seconds.</td>
</tr>
<tr>
<td>Agreement Violation Detector</td>
<td>number of non violating branches</td>
<td>number of branches, containing solving models.</td>
</tr>
<tr>
<td>Agreement Violation Detector</td>
<td>number of runs</td>
<td>counter of how often this module was triggered.</td>
</tr>
<tr>
<td>Agreement Violation Detector</td>
<td>number of violating branches</td>
<td>number of branches, where [SLA] violations have been detected.</td>
</tr>
<tr>
<td>Solving Model Collector</td>
<td>bound on agreement fulfillment</td>
<td>output of the configuration flag for bounding solving branches.</td>
</tr>
<tr>
<td>Solving Model Collector</td>
<td>number of solving models</td>
<td>number of models, which have been collected as solutions.</td>
</tr>
<tr>
<td>Redundancy Eliminator</td>
<td>number of history hits / discarded branches</td>
<td>number requests to history, which have been identified as duplicates.</td>
</tr>
<tr>
<td>Redundancy Eliminator</td>
<td>history size</td>
<td>number of adaptation plans in history.</td>
</tr>
<tr>
<td>Maximum Adaptation Supervisor</td>
<td>bounded branches</td>
<td>number of bounded branches, caused by exceeding the maximum adaptation count.</td>
</tr>
<tr>
<td>Maximum Adaptation Supervisor</td>
<td>max adaptations</td>
<td>number of maximal adaptation actions.</td>
</tr>
<tr>
<td>Cost Bounder</td>
<td>cost violations / discarded branches</td>
<td>number of bounded branches, caused by exceeding the cost limits, specified as user constraints.</td>
</tr>
<tr>
<td>Cost Bounder</td>
<td>cost analysis</td>
<td>number of executed cost analysis.</td>
</tr>
</tbody>
</table>
6.4. Structure of Java Project

The classes of the adaptation framework are structured in multiple packages, separated by their responsibilities. This section introduces these Java packages. In addition to the described packages, corresponding packages for JUnit testing are also included in the Java project, which are not explained here more in detail.

6.4.1. Package tools.descartes.dni.adaptation.framework

The package tools.descartes.dni.adaptation.framework contains the class AdaptationFramework which has to be instantiated in order to run an adaptation process. This is the main class for interacting with the framework.

Additionally, the package contains the AdaptationFrameworkException which is thrown if an exception on startup or on runtime occurs.

6.4.2. Package tools.descartes.dni.adaptation.sla

The tools.descartes.dni.adaptation.sla package contains classes for defining the SLAs and its contained objectives. The AgreementRepository is the container for the SLAs, represented by Agreement class. Each agreement contains a collection of multiple objectives.

Objectives can be instantiated from classes ObjectiveLinkThroughput, ObjectiveNetworkInterfaceThroughput, and ObjectiveSwitchingThroughput. These objectives inherit from the abstract objective AbstractObjective. As the currently defined objectives base on flow and application and focus on minimum throughput observation, the abstract objective AbstractObjectiveFlowApplicationThroughput is defined between the AbstractObjective and the three existing objective classes.

6.4.3. Package tools.descartes.dni.adaptation.cost

The tools.descartes.dni.adaptation.cost package contains a container class CostConstraints for objects of class Constraint. The cost types are specified in the CostType enumeration.

6.4.4. Package tools.descartes.dni.adaptation.model

The package tools.descartes.dni.adaptation.model contains several internal and external used models for the framework. The class AdaptationBranch is used within the iterative adaptation process to transfer the adaptation plan, the violated SLAs, the analysis result and other information to the next module. The BranchCache is also internally used to persist some tactic related meta-data on the adaptation branch. The enumeration Strategy lists the available strategies. The class FrameworkConfiguration represents the configuration of the framework. An object of this class has to be instantiated if a framework configuration object should be programatically created, instead of providing a XML configuration file.
6.4.5. Package tools.descartes.dni.adaptation.module

The package tools.descartes.dni.adaptation.module contains all framework modules, which functionalities are described in Section 5.2. Each module inherits from the abstract class FrameworkModule, which enforces the implementation of the method generateWorkerThread(AdaptationBranch branch) by all modules. This method must return a runnable object, which is executed in a thread pool, defined by the abstract framework module class.

Additionally, the abstract class FrameworkModule implements the interface IStatistic. This interface defines the method getStatistic(), which returns a string-to-string map, containing the statistic of a framework module. This method is invoked after framework execution and the module statistics are collected from all modules by this method.

Assessment of Goals

In Section 1.4 the following goal and two research questions have been defined.

Goal 7: Develop a framework for the adaptation of DNI models.

RQ 7.1: How could the framework separated into multiple modules?

RQ 7.2: Which interfaces should the modules provide and require?

Research question RQ 7.1 has already been answered in Section 5.2.9. Each module has to implement the abstract class FrameworkModule. This inheritance forces the implementation of the method generateWorkerThread(AdaptationBranch branch), which has to return a runnable object, executing the workflow of the module. Research question RQ 7.2 is therefore answered.

By answering research questions RQ 7.1 and RQ 7.2 goal 7 is fulfilled. Details of the used adaption approach for the adaptation framework are described in Chapter 5, technical details are explained in Chapter 6.

6.4.6. Package tools.descartes.dni.adaptation.tactic

The tools.descartes.dni.adaptation.tactic package contains all tactics. A tactic has to implement the ITactic interface, which requires defining the execute(AdaptationBranch adaptationBranch) method. The argument AdaptationBranch submits a single adaptation branch with the corresponding adaptation plan. The tactic searches for alternatives on the adaptation branch and returns an AdaptationBranch collection.

The algorithmic approach of the tactic, i.e., how alternative configurations are determined, is not defined. This is part of each tactic, as explained in Section 5.3.

6.4.7. Package tools.descartes.dni.adaptation.plan

The package tools.descartes.dni.adaptation.plan contains the class AdaptationPlan, representing the data structure of the adaptation plan. The other classes included in this package are child elements of this class and define the adaptation actions, and a compatible data structure for the Dependency of DNI and its corresponding units.

The adaptation plan class also includes methods to merge adaptation actions into an existing adaptation plan, i.e., replacing adaptation actions if such an action to an entity already exists. The applyTo(NetworkInfrastructure networkInfrastructure) method of the adaptation plan consumes a DNI network infrastructure model instance and applies the adaptation actions through the in Section 4.7 described transformation on the model.
6.4.8. Package tools.descartes.dni.adaptation.util

The tools.descartes.dni.adaptation.util package contains several utilities used within the adaptation framework as helper classes. The DependencyComparator implements a Java comparator and enables comparisons between different instances of dependencies, annotated by units. This enables to compare bandwidths of 100 Mbps and 1 Gbps, for example. Internally this is solved via normalizing the values of the dependencies. The DependencyCostComparator also implements a Java comparator to compare different dependencies by an in the comparator constructor defined cost type. This comparator is used in conjunction to the DependencyUtils within tactics to extract the next cost-optimal dependency from a dependency set.

The class CostCalculator is responsible to calculate the costs for an adaptation plan on a given adaptation points model. This class iterates through the scheduled adaptation actions, extract the corresponding costs from the adaptation points and summarizes them for each cost type.

The ValidationUtils are used to extract violating entities from analysis result and a list of violated objectives. This class supports extracting throughput violating nodes, throughput violating network interfaces and throughput violating links.

6.4.9. Package tools.descartes.dni.adaptation.experiment

The package tools.descartes.dni.adaptation.experiment contains a single class ExperimentRunner which is used to run multiple experiments for evaluation purposes in batch mode. This class is neglectable for the adaptation framework and is therefore not explained here in detail.

6.5. Integration of Additional Tactics

One of the best characteristics of the adaptation framework is the extendability by additional tactics. The power of the framework is not limited by the predefined reference tactics, instead additional tactics can be included. The additional tactics could implement another, maybe more intelligent, algorithm for adaptations or could introduce new algorithms. This section describes how additional tactics can be included in the adaptation framework. For simplicity the implementation of an additional tactic in Java programming language is described, but other languages could be used in a similar way through a Java adapter class.

6.5.1. Developing a Tactic

A tactic is a Java class and should be stored in package tools.descartes.dni.adaptation.tactic. A tactic has only a few requirements and some optional features. The tactic class must implement the interface ITactic. This enforces the implementation of the execute(AdaptationBranch adaptationBranch) method, with the return type Collection<AdaptationBranch>. The tactic consumes an adaptation branch, which contains the current adaptation plan, the DNI adaptation points with the referenced DNI network infrastructure, the SLAs with a flag for violated objectives, and the analysis result.

How the tactic determines alternative configurations is scope of the tactic. To enable a target-oriented adaptation algorithm, the violated SLAs as well as the analysis result can be used. The return object of the execute(AdaptationBranch adaptationBranch) method is a collection of adaptation branches. This allows the tactic to branch to different dimensions, e.g., different attributes or to try different alternatives in parallel.
To ensure cost-optimality, the tactic has to output the cost-optimal solution, each for one cost type.

New branches within a tactic should be generated by invoking the method `generateChild(BranchType branchType)` on the input adaptation branch. This method returns a new adaptation branch with a copy of all branch related data. The `BranchType` parameter should be set to `TACTIC` for branching in tactics and only influences the numbering scheme for branch generation.

Beside this requirements it is recommended to create an instance of the used Log4J framework [35]. This is typically done by declaring a `Logger` object, like `private static final Logger logger = LogManager.getLogger()` as a class variable. Any logging within the tactic can be done by `logger.info(String message)` or the other logging methods for traces, debug, warning, and errors, as specified in the Log4J manual [7].

### 6.5.2. Mapping a Tactic to a Strategy

In Section 5.1.2 the triggering process of a tactic was described. Each strategy has one or multiple tactics mapped with it, which are invoked when the strategy is triggered. This mapping is specified in the framework configuration file, as described in Section 6.2.3. A sample configuration file with several mappings is depicted at Listing A.1.

Line 9 of the example configuration file shows a mapping between the strategy `OPTIMIZE_SWITCHING THROUGHPUT` and the tactic class `tools.descartes.dni.adaptation.tactic.IntermediatePerformanceUpgradeTactic`. Every time the `OPTIMIZE SWITCHING THROUGHPUT` strategy is triggered the `IntermediatePerformanceUpgradeTactic` tactic beside other tactics is invoked. Mappings between other strategies and tactics can be defined by the same scheme.

### 6.6. Example Execution of Adaptation Framework

This section describes the execution of an adaptation process by the adaptation framework. The adaptation framework must be available to a executable Java class. The framework sources could either be included into the existing adaptation framework Java project, as the `ExperimentRunner` class in the package `tools.descartes.dni.adaptation.experiment`. This is preferable for testing purposes. For a long-term and enhanced invocation of the adaptation framework, the Maven project of the adaptation project should be included as dependency into a separate Java project.

In addition to the adaptation framework, also the inputs, as described in Section 6.2 have to be provided. This includes the DNI adaptation points with the referenced DNI network infrastructure, the framework configuration and the SLA agreement repository. Specifying the cost constraints is optional. The DNI adaptation points and the DNI network infrastructure have to be provided as files, the other inputs could also be programmatically defined.

The invocation of an adaptation run is depicted in Listing 6.1. Line 1 creates an instance of the adaptation framework. Line 2 to Line 5 set set inputs for the framework, which are provided as files in this example. On Line 6 the adaptation process is started. The `optimize()` method on a framework instance invokes the iterative adaptation process. As soon as all adaptation branches are bounded, i.e., by finding solving models or by filter modules, the adaptation framework stops. The found solutions can be retrieved as a collection of adaptation plans by calling the `getSolvingModels()` method on the adaptation framework instance, as shown in Line 7.

Listing 6.1: Example invocation of DNI adaptation framework in Java.

```java
AdaptationFramework framework = new AdaptationFramework();
framework.setAdaptationPointsModel(new File("network01.dniap"));
framework.setConfiguration(new File("configuration.xml"));
framework.setAgreementRepository(new File("agreements.xml"));
framework.setCostConstraints(new File("costconstraints.xml"));
framework.optimize();
Collection<AdaptationBranch> solvingModels = framework.
  getSolvingModels();
```

6.7. Licensing

The in this work developed DNI adaptation points meta-model, DNI to SimQPN adapter, and the DNI adaptation framework are published under the Apache License 2.0.

The used DNI core meta-model for network infrastructure is also licensed under the Apache License 2.0. The Queueing Petri net Modeling Environment (QPME), which contains the SimQPN simulation framework, is distributed under the Eclipse Public License 1.0. The Apache Log4J library is published under the Apache License 2.0. The JUnit testing framework subject to the Eclipse Public License. The Java Architecture for XML Binding (JAXB) is licensed under a dual license - CDDL 1.1 and GPL 2.0 with class-path exception.

The used libraries of Eclipse Modeling Framework (EMF) are licensed under the Eclipse Public License 2.0. Google Guava is distributed under the Apache License 2.0.

---

8 https://www.apache.org/licenses/LICENSE-2.0
10 https://oss.oracle.com/licenses/CDDL+GPL-1.1
11 https://spdx.org/licenses/GPL-2.0-with-classpath-exception.html
12 https://www.eclipse.org/legal/epl-2.0/
7. Evaluation

In this thesis, an adaptation process for DNI network infrastructure models is developed and implemented as an adaptation framework. The parts modeling, adaptation process, and technical implementation are evaluated in this chapter. Section 7.1 investigates the requirements and assessments of languages and frameworks for adaptation models. Section 7.2 introduces two experimental setups, which are used for the subsequent evaluations. Section 7.3 validates qualitative requirements on the adaptation process. Quantitative characteristics of the adaptation process, are evaluated by case studies in Section 7.4.

7.1. Requirements and Assessment of Languages and Frameworks for Adaption Models

Vogel and Giese investigated requirements and assessments of languages and frameworks for adaptation models [36]. These requirements and assessments have been introduced more in detail in Section 3.6.

This section assesses the language requirements for the developed models, described in Chapter 4, and the adaptation process, introduced in Chapter 5. The adaptation process and the adaptation framework implementation, described in Chapter 6, are evaluated by the corresponding framework requirements.

Table 7.1 depicts the language requirements, specified by Vogel and Giese [36]. Each language requirement (LR) is shortly described. The numbering scheme is equivalent to the original scheme, so that the full specification of each requirement can be looked up at the description of Vogel and Giese [36]. A label of strongly supported (++), supported (+), not supported (-), or irrelevant (0) depicts, if a language requirement is fulfilled by the in this work developed adaptation model and adaptation process. A statement constitutes the decision for every requirement.

In addition to the language requirements, Table 7.2 shows the framework requirements, also identified by Vogel and Giese [36]. Each framework requirement (FR) is shortly described and the numbering scheme is retained to the original specification [36]. The label definition and the statement for each requirement is equivalent to the assessment of language requirements of Table 7.1.
Table 7.1.: Assessment of language requirements for adaptation models.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Assessment</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR-1: Goals</td>
<td>The goals define what the system should do, in form of functional specifications.</td>
<td>++</td>
<td>Specified by traffic definition in DNI network infrastructure model.</td>
</tr>
<tr>
<td>LR-2: Quality Dimensions</td>
<td>The quality dimensions specify via QoS criteria how the system should provide the goals, defined in LR-1.</td>
<td>++</td>
<td>Defined by SLAs and corresponding objectives.</td>
</tr>
<tr>
<td>LR-3: Preferences</td>
<td>As multiple quality dimension (LR-2) can be specified, the preferences set the priority of competing qualities.</td>
<td>-</td>
<td>Multiple solutions are discovered. A concluding assessment could be done in post processing.</td>
</tr>
<tr>
<td>LR-4: Access to Reflection Models</td>
<td>Adaptation models must base on the language of reflection models, as the reflection models contain information about the current system and its environment.</td>
<td>++</td>
<td>The DNI network infrastructure model provides relevant information about the running system and its environment.</td>
</tr>
<tr>
<td>LR-5: Events</td>
<td>Information from events, emitted by the monitor step, should be referenced by adaptation models, to serve as a trigger for the decision-making process and support locating the phenomena in the reflection model.</td>
<td>++</td>
<td>The violated objectives are determined by a set of SLAs which can be used in the adaptation process. Additionally, the analysis result provides fine-grained information about metrics.</td>
</tr>
<tr>
<td>LR-6: Evaluation Conditions</td>
<td>Adaptation models must support the specification of conditions to evaluate the running system and its environment, related to the goals, quality dimensions, and preferences.</td>
<td>++</td>
<td>The in Section 4.4 introduced SLAs can be evaluated and are supported by the adaptation process.</td>
</tr>
<tr>
<td>LR-7: Evaluation Results</td>
<td>The results of the evaluation of the conditions must be included in adaptation model.</td>
<td>++</td>
<td>Each objective contains information about its violation state. These information is accessible by the adaptation process.</td>
</tr>
<tr>
<td>LR-8: Adaptation Options</td>
<td>The configuration space for the system must be captured by adaptation models. This specifies the variability of the system and which options could be adapted.</td>
<td>++</td>
<td>The adaptation options are captured by the adaptation points model, introduced in Section 4.3.</td>
</tr>
<tr>
<td>LR-9: Adaptation Conditions</td>
<td>As not all adaptation options are feasible in every situation, the adaptation condition specifies the appropriate ones for each situation.</td>
<td>++</td>
<td>Based on the violated objectives, the strategy selector, introduced in Section 5.2.4 selects only strategies which are capable to solve the violation.</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
<td>Assessment</td>
<td>Assessment Statement</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LR-10:</td>
<td>Adaptation Costs and Benefits Specifying cost for adaptation options in the</td>
<td>++/+</td>
<td>Costs are strongly supported (++) through a multi-dimensional cost model, introduced in</td>
</tr>
<tr>
<td></td>
<td>adaptation model is useful to choose a suitable solution, if multiple are</td>
<td></td>
<td>Section 4.2 The benefits of the adaptation options are provided as multiple Pareto-optimal</td>
</tr>
<tr>
<td></td>
<td>present. Benefits describe the expected effects of adaptation options and</td>
<td></td>
<td>solutions, if multiple exist (+).</td>
</tr>
<tr>
<td></td>
<td>should also be included in adaptation model.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR-11:</td>
<td>History of Decisions Previous decisions like evaluation results or applied</td>
<td>++</td>
<td>By the nature of the used branch and bound algorithm, each adaptation branch references</td>
</tr>
<tr>
<td></td>
<td>adaptation options should be included as history in adaptation models.</td>
<td></td>
<td>its predecessor. Each branch contains information about planned adaptations (adaptation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>plan) and analysis result.</td>
</tr>
<tr>
<td>LR-12:</td>
<td>Modularity, Abstractions and Scalability An adaptation model should consist</td>
<td>++/0</td>
<td>The adaptation process is modularized in multiple modules (++) as described in Section</td>
</tr>
<tr>
<td></td>
<td>of multiple sub-models instead of being one monolithic model.</td>
<td></td>
<td>5.1.4.1. The extendability is given by adding additional modules and tactics. As the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>adaptation model works on top of DNI the abstraction is prescribed by DNI(0).</td>
</tr>
<tr>
<td>LR-13:</td>
<td>Side Effects Concepts that do not cause and cause side effects on the running</td>
<td>0</td>
<td>As the adaptation process works on different branches and the suggestions are currently</td>
</tr>
<tr>
<td></td>
<td>system should be clearly distinguished by the language.</td>
<td></td>
<td>not automatically applied to the real network, side effects cannot occur.</td>
</tr>
<tr>
<td>LR-14:</td>
<td>Parameters Adaptation options should be parameterized by the adaptation</td>
<td>+</td>
<td>The user requirements can be specified as goals (LR-1) and quality dimensions (LR-2).</td>
</tr>
<tr>
<td></td>
<td>models to adjust models at runtime.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR-15:</td>
<td>Formality A degree of formality enables on-line and off-line validation or</td>
<td>++</td>
<td>Each suggested adaptation is validated through an analysis before it is returned as a</td>
</tr>
<tr>
<td></td>
<td>verification of adaptation models.</td>
<td></td>
<td>solving model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
<td>Assessment</td>
<td>Statement</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LR-16: Reusability</td>
<td>The reusability of the language for adaptation models is encouraged, if the core concept is independent from the language for reflection models.</td>
<td>+</td>
<td>The used concepts of the adaptation process, i.e., branching and bounding, using strategies, tactics and actions, and the definition of costs and SLAS can be applied to other adaptation models. The implementation of the adaptation points model and the implementation of the adaptation tactic depends on the underlying DNI network infrastructure model.</td>
</tr>
<tr>
<td>LR-17: Ease of Use</td>
<td>The design of the language for adaptation model should facilitate an intuitive usage by software engineers.</td>
<td>++</td>
<td>The modularity of the adaptation process and the separation of concerns eases the usage.</td>
</tr>
</tbody>
</table>
Table 7.2.: Assessment of framework requirements for adaptation models.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FR-1: Consistency</td>
<td>Adaptations on reflection models and the running system should be consistent.</td>
<td>0</td>
<td>As the suggested adaptation is currently not applied to the real system, this requirement cannot be evaluated.</td>
</tr>
<tr>
<td></td>
<td>Changes should be performed atomically and correctly to a reflection model and the corresponding set of model changes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR-2: Incrementality</td>
<td>Incrementally techniques should be used to apply or execute adaptation models to promote efficiency.</td>
<td>++</td>
<td>The used algorithm uses incremental changes to adapt the model.</td>
</tr>
<tr>
<td>FR-3: Reversibility</td>
<td>Incremental executed adaptation options should be incrementally reversible.</td>
<td>+</td>
<td>As each branch references its predecessor, the parent branch represents the reversed state.</td>
</tr>
<tr>
<td>FR-4: Priorities</td>
<td>Priorities on modular adaptation models would help the framework to efficiently and easily identify the first entry points for adaptation operations.</td>
<td>+</td>
<td>The strategy selector, as described in Section 5.2.4 only triggers tactics, which have effect on the detected bottleneck.</td>
</tr>
<tr>
<td>FR-5: Time Scales</td>
<td>Different, simultaneously executable time scales for analysis and adaptation planning enables using the framework in mission-critical situations quickly as well as in other situations, where comprehensive and sophisticated reasoning and planning are feasible.</td>
<td>0</td>
<td>The analysis depends on solvers, available for the DNI network infrastructure. If there are no technical limitations of the solvers, they could be run simultaneously.</td>
</tr>
<tr>
<td>FR-6: Flexibility</td>
<td>The framework should support adding, removing or modifying adaptation models at runtime.</td>
<td>+</td>
<td>As the adaptation tactics are responsible for the adaptation algorithms, the flexibility, including learning effects, are provided by them.</td>
</tr>
</tbody>
</table>
It can be concluded that several of the by Vogel and Giese specified language and framework requirements for adaptation models are strongly fulfilled by the developed adaptation model and adaptation approach. As currently no API for the adaptation of a real network is connected to the adaptation framework, the requirements which require interaction with the actual system, cannot be evaluated appropriately. Some limitations are given by the underlying DNI network infrastructure model, especially the abstraction level. The DNI specific adaptation tactics enable a target-oriented adaptation, which result in an optimized runtime for the adaptation process. However, including such domain knowledge, impends the re-usability of the adaptation algorithms for other models. The pursued MAPE-K adaptation control loop facilitates the fulfillment of several of the specified requirements.

7.2. Experimental Setup

The validation of the adaptation process requirements in Section 7.3 and the quantitative assessments through case studies in Section 7.4 will be evaluated through adaptation framework executions. In order to run an adaptation process, the in Section 6.2 required inputs consisting of DNI network infrastructure model, DNI adaptation points model, configuration file, Service Level Agreements (SLAs), and optional cost constraints have to be provided. Section 7.2.1 describes a small DNI network infrastructure model and Section 7.2.2 a more complex one, which are used as input for the evaluation in Section 7.3 and Section 7.4.

The in this section introduced DNI network infrastructure models include the network structure, the traffic specification, and the network configuration, as intended by DNI and described in Section 2.7. Additionally, a default framework configuration for the evaluations will be provided. If changes on these inputs are necessary for an evaluation, the required modifications will be explicitly mentioned. The other inputs like the adaptation point model, the SLAs and the cost constraints vary greatly due the assessments and will be described for every evaluation.

For the evaluations the in Table 7.3 depicted strategy-to-tactic mapping is used. This mapping is identical for all evaluations and is part of each framework configuration file.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Tactic</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIMIZE_SWITCHING_THROUGHPUT</td>
<td>IntermediatePerformanceUpgradeTactic</td>
</tr>
<tr>
<td>OPTIMIZE_SWITCHING_THROUGHPUT</td>
<td>IntermediatePerformanceReplacementTactic</td>
</tr>
<tr>
<td>OPTIMIZE_SWITCHING_THROUGHPUT</td>
<td>NodeReplacementTactic</td>
</tr>
<tr>
<td>OPTIMIZE_INTERFACE_THROUGHPUT</td>
<td>NetworkInterfacePerformanceUpgradeTactic</td>
</tr>
<tr>
<td>OPTIMIZE_INTERFACE_THROUGHPUT</td>
<td>NetworkInterfacePerformanceReplacementTactic</td>
</tr>
<tr>
<td>OPTIMIZE_INTERFACE_THROUGHPUT</td>
<td>NetworkInterfaceReplacementTactic</td>
</tr>
<tr>
<td>OPTIMIZE_INTERFACE_THROUGHPUT</td>
<td>NodeReplacementTactic</td>
</tr>
</tbody>
</table>
7.2. Experimental Setup

<table>
<thead>
<tr>
<th>OPTIMIZE_LINK_THROUGHPUT</th>
<th>LinkPerformanceThroughputUpgradeTactic</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIMIZE_LINK_THROUGHPUT</td>
<td>LinkPerformanceReplacementTactic</td>
</tr>
<tr>
<td>OPTIMIZE_LINK_THROUGHPUT</td>
<td>LinkReplacementTactic</td>
</tr>
</tbody>
</table>

7.2.1. Experimental Network Model 1

Experimental network model 1 will be primarily used for the validation of the adaptation process requirements in Section 7.3. A graphical representation of the network structure with its nodes and network interfaces, and the connections between them is shown in Figure 7.1. There are two end nodes, client $C_1$ on the left side and server $S_1$ on the right side. Each of them contain a network interface, which connects them to switch $SW_1$, representing an intermediate node.

![Figure 7.1.: Structure of experimental network model 1.](image)

A 2 GB file transfer is modeled as traffic flow from server $S_1$ to client $C_1$. The traffic is routed from server $S_1$ via switch $SW_1$ to client $C_1$. In the initial configuration, all Ethernet interfaces contain a performance description, which limit their throughput to a speed of 1 Gbps. Additional performance descriptors on the end nodes, on the intermediate node, or on the links between them are not configured.

7.2.2. Experimental Network Model 2

Experimental network model 2 will be used for the investigation of the cost-optimal solutions, also known as Pareto-front, in Section 7.3.7. Furthermore, this model is used for the quantitative assessments in Section 7.4. The network structure, including the nodes, their corresponding network interfaces, and the links between them is depicted in Figure 7.2. The network contains three clients $C_{11}$, $C_{12}$, and $C_{21}$, placed on the left side in the figure. On the right side two servers $S_1$ and $S_2$ are connected to the network. The three clients as well as the two servers are modeled as end nodes.

The network contains four intermediate nodes, represented by switches $SW_{10}$, $SW_{20}$, $SW_{30}$, and $SW_{40}$. Clients $C_{11}$ and $C_{12}$ are connected to switch $SW_{10}$. Client $C_{21}$ is connected to Switch $SW_{30}$. The switches $SW_{10}$ and $SW_{20}$ are connected to switch $SW_{30}$. Switch $SW_{30}$ has a single link to Switch $SW_{40}$ and servers $S_1$ and $S_2$ are also connected to this device.

An 2 GB file transfer from server $S_1$ to client $C_{11}$ is modeled as traffic flow. The flow is routed from server $S_1$ over switch $SW_{40}$, switch $SW_{30}$ and switch $SW_{10}$ to the destination host client $C_{11}$. All involved network interfaces are initially configured with a performance description, limiting the throughput to 1 Gbps. The end nodes, intermediate nodes and links contain no performance descriptions which would limit the throughput.
7.3. Validation of Adaptation Process Requirements

Chapter 5 describes several requirements of the adaptation process, like the detection of bottlenecks, a target-oriented adaptation, several filter modules and cost-optimal solving models. This section verifies the requirements through different experiments on the adaptation framework.

For the evaluation, the in Section 7.2 introduced network models are used with a variation of SLAs, adaptation points and cost constraints. For each validation, the properties of these input models are defined. Additionally, the adaptation points are graphically illustrated, as they are relevant for the specific evaluations. The expected adaptation process is depicted in an abstract flow chart, before the actual adaptation process, executed by the adaptation framework, is investigated. For greater clarity, only the relevant steps are shown in the flow charts and some actions are summarized. The validation is based on the extracted framework statistic, the resulting adaptation plans, and the log files.

The following requirements are validated. Section 7.3.1 illustrates the detection of solving models in a non-violated scenario. In Section 7.3.2, the target-oriented adaptation on a violated entity is validated. The selection of a cost-optimal solution is shown in Section 7.3.3. In Section 7.3.4, two tactics are executed in parallel, whereby one succeeds and another is not able to find a solution. The bounding of adaptation branches, exceeding the cost limit, is validated in Section 7.3.5. Bounding of branches, exceeding the number of maximal adaptation actions, is verified in Section 7.3.6. The essential requirement of finding all cost-optimal solutions is validated in Section 7.3.7. Section 7.3.8 summarizes the results of the requirement validation.

7.3.1. Solving Model Detection

Requirement: The adaptation process has to detect models, which do not violate the agreed SLAs and identify them as solving models to output them as solution.
7.3. Validation of Adaptation Process Requirements

Setup: For the validation of the solving model detection, experimental network model 1 is used. The corresponding DNI network infrastructure with an already fulfilled SLA is provided as input for the adaptation framework. As no adaptation process will be triggered, the adaptation points model is out of scope for this evaluation. User defined cost constraints as well as a maximum number of adaptation actions are not set.

Expected Behavior: In order to fulfill the requirement of solving model detection, the adaptation process has to proceed the following procedure, which is also depicted in Figure 7.3. The first iteration of the adaptation process has to start with an empty adaptation plan. For the first analysis, the empty adaptation plan is applied to the original DNI network infrastructure, which will result in an unchanged model. The analysis result is afterwards checked for SLA violations in the agreement violation detection module, where no violations have to be detected. The solving model collector recognizes that no SLA violations occurred and identifies the current, empty adaptation plan as solution and provides it as output. The adaptation branch is bounded afterwards and as no further adaptation branches exist, the adaptation process ends here.

![Figure 7.3.: Expected adaptation process for the solving model detection.](image)

Observation and Assessment: An experiment on the adaptation framework with the described inputs showed the expected behavior. The requirement of solving model detection is therefore fulfilled. Additionally, the bounding of an adaptation branch after detecting a solving model and the output of an empty adaptation plan was also shown.

7.3.2. Violating Entity Detection and Target-Oriented Adaptation

Requirement: The adaptation process has to detect the violating entity in a network and has to apply target-oriented adaptations to this entity. The target-oriented adaptation is achieved by invocation of strategies and tactics, based on the violated objective.

Setup: For the detection of the violating entity and the execution of a target-oriented adaptation, experimental network model 1 is used for validation. An objective is specified, which ensures that all network interfaces on the path, traversed by the file transfer, reach a minimum throughput of 1 Gbps. To demonstrate a bottleneck, the throughput of interface e01 on switch SW1 is limited to 100 Mbps. The adaptation points provide an alternative bandwidth of 1 Gbps for all switch interfaces. The bottleneck as well at the adaptation points for Ethernet interface e01 are depicted in Figure 7.4. Neither any cost constraints nor a limitation of the number of adaptation actions are specified.

Expected Behavior: In order to fulfill the requirement of violating entity detection and target-oriented adaptation, the adaptation process has to proceed the following procedure, which is also depicted in Figure 7.5. In the first analysis, the original DNI
network infrastructure is simulated through SimQPN. Based on the analysis result and the provided agreements, a throughput violation on network interface e01 has to be detected. As a violation was identified, the adaptation branch has not to be identified as solution in this iteration. After triggering the interface throughput optimization strategy the four, in the framework configuration file specified, tactics have to be invoked. The tactics for network interface performance replacement, network interface replacement and node replacement will not find any alternatives, as no appropriate alternatives are defined in the adaptation points.

The network interface performance upgrade tactic has to identify the network interface e01 on switch SW1 as bottleneck and searches for alternative throughput parameters, based on the interface type. The alternative throughput of 1 Gbps has to be found in the adaptation points and a corresponding adaptation action for a throughput upgrade has to be created and appended to the currently empty adaptation plan. The filtering modules must have no effects on the resulting adaptation branch.
In the second iteration, the adaptation plan is applied to the original DNI network infrastructure model, which has to result in a model with an upgraded interface performance on network interface e01 on switch SW1. A subsequent SLA violation detection, based on the analysis result of the adapted model, has to identify no further violations. The adaptation plan with the interface performance upgrade action has to be output as solving model and the adaptation branch has to be bounded. The adaptation framework has to terminate after this, as no other adaptation branches exist.

**Observation and Assessment:** An experiment on the adaptation framework with the described inputs showed the expected behavior. The violated objective triggered its associated strategy, which invoked the related tactics. The tactics identified correctly the network interface representing the bottleneck and discovered an alternative configuration with improved performance. In the second iteration, no SLA violations have been detected on the adapted DNI network infrastructure model and the adaptation plan was identified as a solution. The requirement of violating entity detection and target-oriented adaptation is therefore fulfilled.

**Assessment of Goals**

In Section 1.4 the following optional goal and two optional research questions have been defined.

**Goal 8:** Experimental evaluation of sample DNI model instances. *(optional)*

**RQ 8.1:** Does the suggested solutions improve the original configuration? *(optional)*

**RQ 8.2:** How can the suggested solutions be improved with regard to runtime? *(optional)*

From Section 7.3.2 it can be concluded that the suggested solution improves the original configuration. Therefore research questions RQ 8.1 is answered. A throughput violation was detected on interface e01 of switch SW01. The problem has been solved by increasing the throughput of this interface from 100 Mbps to 1 Gbps.

Research question RQ 8.2 is answered later.

### 7.3.3. Cost-Optimal Parameter Adaptation

**Requirement:** The adaptation process has to suggest cost-optimal solutions, from a list of alternative parameters. If a specific parameter is optimized, the alternative with lowest costs has to be selected first.

**Setup:** For the validation of the selection of a cost-optimal alternative parameter, the scenario of Section 7.3.2 is reused. An objective observes a minimum bandwidth of 200 Mbps for the file transfer from server S1 to client C11. A bottleneck is created on Ethernet interface e01 on switch SW1 by limiting the throughput to 100 Mbps. Bandwidths of 500 Mbps and 1 Gbps are modeled as alternatives for the network interfaces in the adaptation points, as shown in Figure 7.6. Each of the alternative bandwidths is annotated by an investment cost value. Investment costs of 600.0 are assigned to the higher bandwidth of 1 Gbps. It is assumed that the bandwidth of 500 Mbps, as this is no standard speed for Ethernet, arises higher investment costs of 800.0. Cost constraints and a limitation of the number of adaptation actions are not set.
**Expected Behavior:** In order to fulfill the requirement for cost-optimal parameter adaptation, the adaptation process has to proceed the following procedure, which is also depicted in Figure 7.5. In the first iteration, the initial DNI network infrastructure model is analyzed. A SLA violation on network interface e01 on switch SW1 has to be detected. As the network interface e01 with the bandwidth of 100 Mbps on switch SW1 did not meet the agreed objective of an interface throughput of 200 Mbps the adaptation process has to discover alternative configurations. Instead of selecting the next-best, related to the performance gain, alternative bandwidth of 500 Mbps, the alternative bandwidth of 1 Gbps has to be chosen. The tactic has to select the 1 Gbps with a investment effort of 600.0 as cost-optimal solution, as the 500 Mbps would require a higher investment of 800.0. After the cost-optimal value of 1 Gbps is chosen, the fulfillment of the violated objective has to be verified by a second analysis. The resulting adaptation plan has to be output as solution.

**Observation and Assessment:** An experiment on the adaptation framework with the described inputs showed the expected behavior. In this experiment, the higher bandwidth, but more cost-optimal solution of 1 Gbps has been chosen, as this is the cost-optimal solution. The requirement of a cost-optimal parameter adaptation is therefore fulfilled.

### 7.3.4. Multiple Tactics Execution

**Requirement:** The adaptation process has to trigger multiple tactics, as they are suitable to solve a detected SLA violation, in parallel branches.

**Setup:** For the validation of executing multiple tactics in parallel, experimental network model 1 is used. One objective observes the file transfer flow and maintains a minimal network interface throughput of 1 Gbps. The bandwidth on Ethernet interface e01 on switch SW1 is limited to 100 Mbps to create a bottleneck. The adaptation points define two different types of adaptations for the interface, illustrated in Figure 7.7. At first, the current used interface gets annotated by an alternative bandwidth of 500 Mbps. Secondly, an additional compatible network interface type is added to the switch type of SW1. This new network interface type supports bandwidths of 100 Mbps and 1 Gbps. User constraints for cost and number of adaptation actions are not specified.

**Expected Behavior:** In order to fulfill the requirement of parallel execution of multiple adaptation tactics, the adaptation process has to proceed the following procedure, which is also depicted in Figure 7.8. The agreement violation detector has to detect the violated SLA on network interface e01 on switch SW1 in the first iteration. Defined through the violated objective, the strategy to optimize the interface throughput has to be triggered. This strategy invokes four associated tactics, as specified in the framework configuration file. The adaptation branch has to be forked as copy to the tactics.
The network interface performance upgrade tactic forces on upgrading the throughput. The bandwidth for the current used interface has to been set to the only available alternative parameter value of 500 Mbps by this tactic. The network interface replacement tactic has to replace the currently used network interface through the alternative net-
work interface in parallel. The performance of the alternative network interface has to be initially set to 100 Mbps, because this is the cost-optimal solution candidate.

In the next iteration, both adaptation branches, one with the upgraded interface bandwidth and the other with the replaced network interface, are analyzed. The SLA violation on both branches still remains, as none of them provides the minimum required throughput. Again, the strategy selector has to invoke different tactics independently from other branches. As only the interface bandwidth upgrade tactic has effect on the subsequent adaptation, only this tactic will be considered from this point on. On the first branch with the original Ethernet interface, which was previously upgraded from 100 Mbps to 500 Mbps, no other alternative bandwidths are available. As the tactic has no valid alternatives, the corresponding adaptation branch has to be bounded here and no adaptation plans have to be returned.

The second branch, with the previously replaced new network interface type, has an initial bandwidth of 100 Mbps for interface e01. The interface throughput upgrade tactic has to discover an alternative bandwidth of 1 Gbps from the adaptation points and set the higher bandwidth for interface e01 on switch SW1. A subsequent analysis in the third iteration has to identify this branch as solution, as no further SLA violations occur.

Observation and Assessment: An experiment on the adaptation framework with the described inputs showed the expected behavior. Different adaptation tactics have been invoked in parallel, and independent from each other. The requirement of executing multiple adaptation tactics in parallel is therefore fulfilled.

7.3.5. Cost-Limit Bounding

Requirement: The adaptation process has to bound branches, if they exceed the maximum costs, specified by user constraints.

Setup: For the validation of the cost-limit bounding, experimental network model 1 is used. An objective observes a minimum interface throughput of 1 Gbps for the file transfer between server S1 and client C1. A bottleneck is created on network interface e01 on switch SW1 by limiting the bandwidth to 100 Mbps. The adaptation points define two compatible network interfaces for switch SW1, as depicted in Figure 7.9. For the current used network interface the alternative bandwidths of 100 Mbps and 1 Gbps are specified. The bandwidth of 100 Mbps is annotated by investment costs of 100.0, the higher bandwidth of 1 Gbps arises an investment effort of 800.0. The second interface type provides the same alternative bandwidths. The 100 Mbps bandwidth on the second interface type is annotated by investment costs of 200.0, while the higher bandwidth of 1 Gbps generates investment costs of 400.0. For the adaptation process, a limit of 500.0 is set for investment costs. The number of maximum adaptation actions is not restricted.

Expected Behavior: In order to fulfill the requirement of bounding branches, exceeding the cost-limit, the adaptation process has to proceed the following procedure, which is also depicted in Figure 7.10.

The analysis and the agreement violation detection has to identify the violated interface throughput on Ethernet interface e01 on switch SW1 in the first iteration. The triggered strategy invokes different adaptation tactics, including upgrading the interface bandwidth and replacing the network interface. The interface upgrade tactic has to discover and set the alternative bandwidth of 1 Gbps for network interface e01. The in parallel executed interface replacement tactic has to replace network interface e01 by the second interface type and set the bandwidth to the initial value of 100 Mbps. Both tactics should not consider any costs and especially the limitation of costs here. The two adaptation branches are forwarded to the filter modules.
Figure 7.9.: Adaptation points for the validation of cost-limit bounding.

Figure 7.10.: Expected adaptation process for the cost-limit bounding.
The first branch, based on the initial network interface with an upgraded bandwidth to 1 Gbps, arises investment costs of 800.0, which exceeds the limitation of 500.0 for investment. Therefore, the branch has to be bounded by the sfilter module. Possible additional performance upgrades for this network interface would not be considered, as they still result in more investment costs as the bounded branch. The second adaptation branch only establishes costs for the throughput of 100 Mbps and has not to be bounded here.

The analysis in the second iteration has to detect furthermore a SLA violation, and the interface upgrade tactic has to upgrade the bandwidth of the second interface type to 1 Gbps. This adaptation action arises investment costs of 400.0 for this interface type on the adaptation branch. The subsequent cost bounding module has to not bound this branch, as it does not exceed the limit of 500.0 for investments. In the third iteration, the agreement violation detector has not to detect any furthermore SLA violations and the adaptation plan, corresponding to this adaptation branch, has to be returned as solution.

**Observation and Assessment:** An experiment on the adaptation framework with the described inputs showed the expected behavior. As only the adaptation plan from the second branch was output, it can be concluded that the first branch has been bounded. The adaptation framework log file confirms this observation. The requirement of bounding branches, exceeding the cost-limit specified by user constraints, is therefore fulfilled. The cost constraint bounding also contributes to the requirement of a runtime optimized adaptation.

### 7.3.6. Adaptation Count Bounding

**Requirement:** The adaptation process has bound adaptation branches, which exceed the maximum number of adaptation actions, specified by user constraints.

**Setup:** For the validation of the adaptation count bounding, experimental network model 1 is used. The file transfer between server S1 and client C1 is observed by an objective, enforcing a minimum network interface throughput of 1 Gbps. For this scenario the initial bandwidth of Ethernet interface e01 on switch SW1 is set to 100 Mbps to create a bottleneck. The adaptation points provide two compatible network interfaces for switch SW1 as depicted in Figure 7.11. Both interface types support bandwidths of 100 Mbps and 1 Gbps. As costs are not relevant for this evaluation, the alternative configurations are not annotated by any cost types and no cost limit is set. The number of adaptation actions is restricted by the value of “1”, which means that only one adaptation action is allowed to solve a SLA violation.

![Figure 7.11.: Adaptation points for the validation of adaptation count bounding.](image-url)
**Expected Behavior:** In order to fulfill the requirement of solving model detection, the adaptation process has to proceed the following procedure, which is also depicted in Figure 7.3. A bottleneck on Ethernet interface e01 on switch SW1 has to be detected by the analysis and the agreement violation detection on first iteration. The adaptation process invokes the four tactics for optimizing the interface throughput. The interface upgrade tactic has to set the alternative throughput to 1 Gbps on the initial interface type. This adaptation requires one adaptation action, which is appended to the empty adaptation plan on this branch.

The parallel executed interface upgrade tactic has to replace the interface type by the alternative network interface. As the alternative interface also requires a performance description one of the available bandwidths is selected. Which of them is chosen is irrelevant for this evaluation. The interface replacement as well as setting the initial interface performance are two adaptation actions, which have to be added to the adaptation plan of this adaptation branch.

After the execution of the two tactics, the two resulting adaptation branches have to traverse the filter modules. The maximum adaptation supervisor compares the number of adaptation actions of the branches to the maximum allowed adaptation actions, set by user constraint. The adaptation branch with one adaptation action has to pass the filter module, while the second adaptation branch with two adaptation actions has to be bounded by this module. The first and only resulting adaptation branch is analyzed in the second iteration of the adaptation process and as no further SLA violations has to be detected, the corresponding adaptation plan represents a solving model.
Observation and Assessment: An experiment on the adaptation framework with the described inputs showed the expected behavior. As only the adaptation plan with the single adaptation action was output, it can be concluded that the branch with two adaptation actions has been bounded. The adaptation framework log file confirms this observation. The requirement of bounding branches, exceeding the number of adaptation actions specified by user constraints, is therefore fulfilled. The adaptation count bounding also contributes to the requirement of a runtime optimized adaptation.

7.3.7. Cost-Optimal Solution Discovery (Pareto Front)

Preface: The adaptation process includes several runtime-optimizations, as target-oriented adaptation and several bounding steps. Although a short runtime is essential for the adaptation process, the cost-optimality has still to be ensured.

Requirement: The adaptation process has to discover all cost-optimal solutions.

Setup: For the validation of the cost-optimal solution discovery, the returned solutions from the optimized adaptation process and a brute force approach are compared. As the difference between the two approaches is more noticeable on larger network models with more adaptation points, the experimental network model 2 is used for this evaluation. Similar to the previous introduced evaluations an objective observes a minimum network interface throughput of 1 Gbps for the file transfer between server $S1$ and client $C11$. The performance of Ethernet interface $e03$ on switch $SW30$ is initially set to 100 Mbps, to create a bottleneck for the file transfer.

The adaptation points are modeled on different levels for this evaluation. The switches $SW30$ and $SW40$ get assigned switch type $A$, which is replaceable over an adaptation group by switch type $B$ and $C$. Switch type $B$ and $C$ allow the three different network interface types $B$, $C$, and $D$, which are compatible with the initial switch model. Alternative interface type $B$ supports bandwidths of 100 Mbps and 1 Gbps. For alternative interface type $C$ the bandwidths of 1 Gbps and 10 Gbps are available. Interface type $D$ allows bandwidths of 10 Mbps, 100 Mbps, 1 Gbps, or 10 Gbps. Each of the alternative configuration is annotated by costs of type investment and handling time. The adaptation points including the costs are illustrated in Figure 7.13. Cost constraints and a limit for the number of maximal adaptation actions are not set.

Expected Behavior: For this evaluation, a summarized description of the required procedure is given, as through the multiple adaptation alternatives, the adaptation process creates a large number of branches. For this reason, a flow chart, illustrating the decisions and branches, is omitted as well. To fulfill the requirement to find all cost-optimal solutions only the discovered solutions are relevant, and the decision process is negligible.

The brute force algorithm will generate all solutions, regardless of the introduced costs. A brute force approach includes no target-oriented adaptation, which means that all network interfaces on switch $SW30$ and $SW40$ will be adapted, even if they are no bottleneck. Additionally, instead of bounding branches after they have been identified as solutions, other, maybe more expensive alternative configurations, would also be tried.

The target-oriented adaptation process has to select the executed tactics, based on the violated objectives. These tactics have only to adapt the entity, which is the origin of the violation. As soon as a solution is found on a branch, the branch has to be bounded. Filter modules will also facilitate to keep the number of adaptation branches low. But in spite of all runtime optimizations, the target-oriented adaptation process has to discover all cost-optimal solutions, which have been found by the brute force approach.
Observation and Assessment: The brute force algorithm outputs 231 solving models, while the optimized adaptation process discovered 3 solutions after 10 iterations. The 3 solutions from the optimized adaptation process, including their introduced costs are shown in Table[7.4]. The returned solutions contain interface replacements and bandwidth specifications, which can be identified through the corresponding adaptation plans. It can be seen that no switch replacements are included in the solutions, although branches for switch replacement have been generated. The adaptation process identified the three branches, containing the interface replacement on the original switch type as solving models, in an early state. Other branches, introducing higher costs in all dimensions like the switch replacements, have been bounded therefore.

Table 7.4.: Resulting adaptations of the optimized adaptation process to discover cost-optimal solutions (Pareto front).

<table>
<thead>
<tr>
<th>Solution</th>
<th>New Interf. Type</th>
<th>New Bandw.</th>
<th>Investment</th>
<th>Handl. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.D1</td>
<td>Type B</td>
<td>1 Gbps</td>
<td>20.0</td>
<td>128.0</td>
</tr>
<tr>
<td>0.D2</td>
<td>Type C</td>
<td>1 Gbps</td>
<td>27.0</td>
<td>121.0</td>
</tr>
<tr>
<td>0.D3</td>
<td>Type D</td>
<td>1 Gbps</td>
<td>40.0</td>
<td>114.0</td>
</tr>
</tbody>
</table>
The solving models from the optimized adaptation process as well as the solving models from the brute force algorithm are depicted in Figure 7.14. The x-axis depicts the investment cost and the y-axis the handling time. It can be seen that the solutions of the optimized adaptation process, depicted as blue filled circles, are all cost optimal, building a Pareto front. No other solution, discovered by the brute force algorithm, results in less costs in any dimension.

![Figure 7.14: Pareto front generated by the optimized adaptation process compared to solutions, generated by a brute force approach.](image)

Investigations on the performance of the optimized adaptation process will be discussed in the quantitative evaluation in Section 7.4.3.

7.3.8. Summary of Requirement Validation

Section 7.3 validated the requirements of the adaptation process, as described in Chapter 5. From the investigation it can be concluded that all requirements are fulfilled. It should be mentioned that the validation has been executed on small experimental network models for a better overview and transparency, but the results are transferable also to larger network models. Table 7.5 depicts a summary of the validated requirements for the adaptation process.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Evaluation</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>solving model detection</td>
<td>Section 7.3.1</td>
<td>fulfilled</td>
</tr>
<tr>
<td>violating entity detection and target-oriented adaptation</td>
<td>Section 7.3.2</td>
<td>fulfilled</td>
</tr>
<tr>
<td>cost-optimal parameter adaptation</td>
<td>Section 7.3.3</td>
<td>fulfilled</td>
</tr>
<tr>
<td>multiple tactics execution</td>
<td>Section 7.3.4</td>
<td>fulfilled</td>
</tr>
<tr>
<td>cost-limit bounding</td>
<td>Section 7.3.5</td>
<td>fulfilled</td>
</tr>
<tr>
<td>adaptation count bounding</td>
<td>Section 7.3.6</td>
<td>fulfilled</td>
</tr>
<tr>
<td>cost-optimal solution discovery (Pareto front)</td>
<td>Section 7.3.7</td>
<td>fulfilled</td>
</tr>
</tbody>
</table>
7.4. Quantitative Assessments (Case Studies)

Section 7.3 validated the requirements for the adaptation process, which can be classified as qualitative assessments. Beside the functionality, the adaptation process has also to fulfill some quantitative requirements. This section investigates such qualitative requirements like the efficiency of the algorithmic approach, the scalability to complex network models, and the processing time on extensive adaptation points. For the evaluation, the in Section 7.2 introduced experimental network models will be used. If relevant, a statement for transferability to other network models is made for each assessment.

This section is outlined as follows. In Section 7.4.1 the scalability of the adaptation process on large network models is analyzed. Section 7.4.2 investigates impact factors on the runtime of the adaptation process. The runtime of the optimized adaptation process compared to the runtime of a brute force algorithm is depicted in Section 7.4.3. Section 7.4.4 summarizes the investigations from this section.

7.4.1. Investigation of Scalability

Motivation: It is assumed that networks in real world are larger than the experimental network models, introduced in Section 7.2. One motivation for the automated adaptation process are data center networks, which are too complex for network experts, to make educated guesses for network adaptations. This case study investigates the impact of the network size to the runtime of the adaptation process. Experimental network model 1 and 2, which differ in number of nodes and links, are used for this evaluation.

Setup: For the experiment with network model 1 an objective, observing a minimal network interface throughput of 1 Gbps for the file transfer between server $S1$ and client $C1$, is defined. A similar objective to observe a minimal network interface throughput of 1 Gbps between server $S1$ and client $C11$ for the file transfer is set for experimental network model 2.

The switches in both network models are annotated by four alternative interfaces, supporting different sets of alternative bandwidths. The adaptation points for experimental network model 1 are depicted in Figure 7.15, the adaptation points for model 2 are shown in Figure 7.16. These adaptation points for both models are equivalent, so each model has the same degrees of freedom. Neither cost constraints nor limits for the number of adaptation actions are set.

Observation: Both adaptation runs, result in an identical number of three solving models. This behavior was expected through the the equivalent adaptation points. The observed parameters of this investigation for the adaptation process on experimental network model 1 and 2 are depicted in Table 7.6. The runtime of the adaptation processes differ, which could be explained through a longer simulation time for larger network models and little bit more time for searching the violated entity in a larger network model. The adaptation run on the small network model as well as the adaptation run on the larger network model executed the identical number of analyses, which is a significant recognition. As the target-oriented adaptation identifies the violating entity and only applies adaptations on this, the number of other nodes in the network model is irrelevant for the efficiency of the adaptation process.

Assessment: It can be concluded that the size of the network model has no impact on the number of required analysis to solve a violation on a single entity. This statement is important, as the adaptation process could also be used for more complex networks, as deployed in real world, without an increasing number of required analysis. However, the simulation time will increase by the size of the network model, but the efficiency of the simulation engine is not considered in this work.
Figure 7.15.: Adaptation points for the investigation of scalability on experimental network model 1.

Figure 7.16.: Adaptation points for the investigation of scalability on experimental network model 2.
7.4.2. Impacts on Runtime of Adaptation Process

Section 7.4.1 shows that the size of the network model has no impact on the number of analyses within an adaptation process. Beside the size of models, there are several other factors, which could potentially influence the number of branches, the required number of analyses and therefore the overall runtime of the adaptation process.

In this investigation four influence factors on the overall runtime are analyzed. The investigation is divided into four separate parts. Section 7.4.2.1 investigates how cost annotations influence the overall runtime. Section 7.4.2.2 deals with the trend between the two cost dimensions. In Section 7.4.2.3 the impact of the number of alternative parameters is investigated. Section 7.4.2.4 analyzes the impact of the number of alternative entities on the runtime.

Instead of comparing the runtime in a temporal metric, the runtime is measured through the number of required analysis. As the analysis requires the most time of an adaptation iteration, this is the most significant factor for the investigation of runtime of an adaptation process.

The investigations for the different influence factors are executed on experimental network model 1. An objective observes a minimal interface throughput of 1 Gbps for the file transfer between server S1 and client C1. A bottleneck is created on network interface e01 on switch SW1, by limiting the bandwidth to 100 Mbps. The adaptation points and the cost annotations are explained for every part of the analysis. Cost constraints and a limit for the number of adaptation actions are not set.

7.4.2.1. Impact of Cost Annotations

In this part of the investigation, the impact of cost annotations to the overall runtime is analyzed. For this, the adaptation process runs two times on the experimental network model 1. The observations of the investigation are depicted in Table 7.7.

Table 7.7.: Observed parameters from the investigation of impact of cost annotations to the runtime of the adaptation process.

<table>
<thead>
<tr>
<th>Observation</th>
<th>With Cost Annotation</th>
<th>Without Cost Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of analyses</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>number of solving models</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

On the first run, the adaptation points as illustrated in Figure 7.17 are used. The currently used interface has an alternative bandwidth of 1 Gbps, which would potentially solve the expected SLA violation. In addition to the bandwidth upgrade, also the network interface can be replaced by two other interface types, supporting bandwidths of 100 Mbps and 1 Gbps. The adaptation process discovered three solving models and executed six simulations. One model represents the bandwidth upgrade on the initial
network interface type, and the other twice models the network interface replacement in addition to a bandwidth upgrade. The executed analyses were executed on the initial network model, on the branch with the bandwidth upgrade, on the two branches with the interface replacement with an initial bandwidth of 100 Mbps, and additional two times on the interface replacement branches with a bandwidth upgrade.

Figure 7.17.: Adaptation points for the investigation of impact of no cost annotation to the runtime of the adaptation process.

In a subsequent adaptation process, the adaptation points are annotated by costs, as depicted in Figure 7.18. It is assumed that an interface replacement would arise more investment costs and require more handling time, instead of only increasing the bandwidth. Running the adaptation process on the cost annotated model only returns one solving model via four analyses. The solving model represents the bandwidth upgrade on the original network interface. The analyses have been executed on the initial network model, on the branch with the bandwidth upgrade, and on the two branches with the interface replacement branches with an initial bandwidth of 100 Mbps. The solving model has been identified as solution in the second iteration. The interface replacement branches were bounded after that, as they result in higher costs as the already found solution.

From this observation it can be concluded that the runtime of the adaptation process benefits from cost annotations.

Figure 7.18.: Adaptation points for the investigation of impact of cost annotations to the runtime of the adaptation process.

7.4.2.2. Impact of Opposed Cost Annotations

When selecting the next cost-optimal alternative parameter from a list of alternatives, different cost-best alternatives could be found for each cost type. This is meant by
opposed cost annotation. In such case an own adaptation branch for each cost type is generated. This part investigates the impact of generating additional adaptation branches, caused on opposed cost annotations, to the runtime for the adaptation process. For this investigation, two adaptation processes are executed on experimental network model 1. The observations of the investigation are depicted in Table 7.8.

Table 7.8.: Observed parameters from the investigation of impact of opposed cost annotations to the runtime of the adaptation process.

<table>
<thead>
<tr>
<th>Observation</th>
<th>Commutated Cost Annotation</th>
<th>Opposed Cost Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of analyses</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>number of solving models</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The adaptation points for the first run are depicted in Figure 7.19. The current used bandwidths of 100 Mbps can be replaced by bandwidths of 1 Gbps and 10 Gbps, which would obviously both solve the SLA violation. A higher performance value arises more costs in both dimensions for the first run, as depicted in the figure. The adaptation process returns one solving model by two simulations for this investigation. The solving model contains the bandwidth upgrade to 1 Gbps, as this is cost optimal for both cost types.

Figure 7.19.: Adaptation points for the investigation of impact of commutated cost annotations to the runtime of the adaptation process.

A second adaptation process bases on the adaptation points depicted in Figure 7.20. The costs are modeled opposed now, the highest bandwidth of 10 Gbps arises still more investment costs but less handling time, compared to the 1 Gbps alternative. The adaptation process output two solving models by three simulations. One models contains the bandwidth upgrade to 1 Gbps and the other model the upgrade to 10 Gbps. Output of these two solutions requires the additional executed simulation, compared the the first run of the adaptation process.

It can be concluded that opposed cost annotations have negative influence on the runtime of the adaptation process.

7.4.2.3. Impact of Number of Alternative Parameters

In this part of the investigation, the impact of the number of alternative parameters to the runtime of the adaptation process is analyzed. For this, the adaptation process runs two times on the experimental network model 1. The observations of the investigation are depicted in Table 7.9.
Figure 7.20.: Adaptation points for the investigation of impact of opposed cost annotations to the runtime of the adaptation process.

Table 7.9.: Observed parameters from the investigation of impact of number of alternative parameters to the runtime of the adaptation process.

<table>
<thead>
<tr>
<th>Observation</th>
<th>Single Alternative</th>
<th>Multiple Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of analyses</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>number of solving models</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For a first run, the adaptation points as illustrated in Figure 7.21 are used. There is only one alternative bandwidth of 1 Gbps available, which solves the SLA violation. The adaptation process terminates after two simulations, one simulation on the initial bandwidth of 100 Mbps to detect the violation, and a second simulation on the 1 Gbps upgrade to identify this as solution.

Figure 7.21.: Adaptation points for the investigation of impact of number of less alternative parameters to the runtime of the adaptation process.

In a second run of the adaptation process, the adaptation points as depicted in Figure 7.22 are used. In contrast to the first run, there are additional alternative parameters of 10 Gbps and 40 Gbps available. The cost annotation is required to avoid the effect of opposed costs as introduced in Section 7.4.2.2. Running the adaptation process on this adaptation points result also in two simulations with one solving model, containing the upgrade to 1 Gbps. As parameter adaptations are iteratively executed, the adaptation process terminates, as soon as a solution was found.

It can be concluded that additional parameters, which would also solve the SLA violation, do not negatively influence the runtime of the adaptation process. It has to be mentioned that alternative parameters with higher performance, which do not solve the violation, have negative impact on the runtime, as they are iteratively selected and also have to be analyzed.
7.4.2.4. Impact of Number of Alternative Entities

This part of the investigation deals with a different number of alternative entities in an adaptation process. For this, the adaptation process runs two times on the experimental network model 1. The observations of the investigation are depicted in Table 7.10.

Table 7.10.: Observed parameters from the investigation of impact of number of alternative entities to the runtime of the adaptation process.

<table>
<thead>
<tr>
<th>Observation</th>
<th>Two Alternatives</th>
<th>Five Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of analyses</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>number of solving models</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

For the first run, the adaptation points, depicted in Figure 7.23, are used. There are two alternative network interface types, which solve the SLA violation. Solving the violation through a bandwidth upgrade on the current used network interface type is consciously not available, as this investigation should focus on entity replacement. The adaptation process with this adaptation points result in two solving models, one for each alternative interface type, discovered by three simulations.

Figure 7.23.: Adaptation points for the investigation of impact of number of less alternative entities to the runtime of the adaptation process.
A second adaptation run is executed on the adaptation points, depicted in Figure 7.24. This adaptation points are extended by three additional alternative interfaces. Running the adaptation process on this adaptation points outputs five solving models after six simulations.

Figure 7.24.: Adaptation points for the investigation of impact of number of many alternative entities to the runtime of the adaptation process.

It can be concluded that a higher number of alternative entities negatively influences the runtime for the adaptation process. If the alternative entities contain a solution or not is subordinate for this consideration.

Assessment of Goals

In Section 1.4 the following optional goal and two optional research questions have been defined.

**Goal 8:** Experimental evaluation of sample DNI model instances. *(optional)*

**RQ 8.1:** Does the suggested solutions improve the origin configuration? *(optional)*

**RQ 8.2:** How can the suggested solutions be improved with regard to runtime? *(optional)*

Research question RQ 8.1 has already been answered in Section 7.3.2.

Research question RQ 8.2 could be answered in multiple ways. The S/T/A approach, introduced in Section 5.1.2 enables a target-oriented adaptation. In comparison to a brute force adaptation this approach improves the runtime of the adaptation process. The bounding modules, described in Section 5.2, filter useless branches and avoid unnecessary analysis. Other impact factors, depending on input models, number of SLA violations, user constraints, cost annotations, and configuration settings are described in Section 7.4.2 and depicted in Table 7.12.

Research question RQ 8.1 and RQ 8.2 are answered, therefore the optional goal 8 is fulfilled. The related experimental evaluation of models is executed in Section 7.3 and Section 7.4. The experimental models are introduced in Section 7.2.
7.4.3. Comparison to Brute Force

In Section 7.3.7 the requirement of the discovery of all cost-optimal solutions is validated. The solutions, discovered by the runtime, target-oriented adaptation process are compared to the solutions, found by a simple brute force approach. It can be concluded that the optimized adaptation approach is able to find all cost-optimal solutions, generating the Pareto front.

In addition to this functional validation, also the performance aspect of the optimized adaptation approach is important. The optimized runtime of the adaptation process is among others achieved through a target-oriented selection of strategies and tactics, based on the type of violation. Additionally, the executed tactics discover the violated entities and only execute adaptation actions on these. Several bounding modules also have a positive impact on runtime of the adaptation process, as useless adaptation branches are bounded in an early state to avoid unnecessary time-consuming simulations.

The evaluation in this section compares the performance relevant observations from the optimized adaptation process and the brute force approach of the experiment, introduced in Section 7.3.7. Table 7.11 shows the relevant observations of both approaches, extracted from the statistic.

<table>
<thead>
<tr>
<th>Observed Metric</th>
<th>Brute Force</th>
<th>Optimized Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall runtime (sec.)</td>
<td>6379</td>
<td>24</td>
</tr>
<tr>
<td>solving models</td>
<td>231</td>
<td>3</td>
</tr>
<tr>
<td>number of analyses</td>
<td>3190</td>
<td>10</td>
</tr>
<tr>
<td>number of adaptation plans in history of redundancy eliminator</td>
<td>3189</td>
<td>17</td>
</tr>
<tr>
<td>bounded branches by cost optimization</td>
<td>-</td>
<td>8</td>
</tr>
</tbody>
</table>

The overall runtime differs extremely between both approaches. Especially in online scenarios, when suggestions have to be available after a short time, the brute force approach is less helpful. The comparison bases on a small network model. With an increasing network size and with increasing network traffic, the simulation time also increases which would result in even more time required for the analysis. The optimized adaptation process discovered 3 solving models, compared the 231 solutions from the brute force approach. As already validated, the small set of 3 solving models by the optimized approach, contains all cost-optimal solutions, which would be preferable by the user. As the brute force approach considers all possible solutions, much more branches have been generated, which result in a higher number of analyses. The remaining 228 solutions from the brute force approach result in higher cost and are therefore not cost-optimal. The number of adaptation plans in history of the redundancy eliminator module are informal. The optimized approach bounded 8 branches, caused by already discovered cost-optimal solutions.

From the investigation it can be concluded that the adaptation process benefits from the optimized approach, related to the required time to discover solving models. There are no quality disadvantages, related to cost-optimal solutions, which was already validated in Section 7.3.7. This observation is significant, as it can be assumed that for more complex real world networks a brute force approach could not be handled and such optimized approach becomes necessary.
7.4.4. Summary of Quantitative Assessments

The investigations of Section 7.4 analyzed different impact factors on the runtime of an adaptation process. The results are depicted in Table 7.12 and are summarized as follows.

Section 7.4.1 shows that the size of the network on a constant number of violating entity, does not negatively influence the runtime of an adaptation process. From the investigations of Section 7.4.2.1 it can be concluded that in general the runtime benefits from cost annotations, as they allow bounding of branches, which are more expensive in each cost-dimension as already found solutions. Modeling the costs opposed has little negatively impact on the adaptation process, as shown in Section 7.4.2.2. Through opposed costs more branches have to be generated, which have to be analyzed. The investigation of Section 7.4.2.3 showed that multiple alternative parameter which solve the SLA violation have no impact on the runtime of the adaptation process. From the investigation of Section 7.4.2.4 in can be concluded that an increasing number of alternative entities influences the runtime negatively, as additional branches have to be analyzed.

Table 7.12.: Summary of impact factors to the runtime of an adaptation process.

<table>
<thead>
<tr>
<th>Impact Factor</th>
<th>Number of Analyses</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost annotations</td>
<td>decreasing</td>
</tr>
<tr>
<td>opposed cost annotations</td>
<td>slightly increasing</td>
</tr>
<tr>
<td>increasing number of alternative parameters</td>
<td>constant</td>
</tr>
<tr>
<td>increasing number of alternative entities</td>
<td>increasing</td>
</tr>
<tr>
<td>increasing size of network</td>
<td>constant</td>
</tr>
<tr>
<td>cost limitations (user constraint)</td>
<td>decreasing</td>
</tr>
<tr>
<td>limiting number of adaptation actions</td>
<td>decreasing</td>
</tr>
</tbody>
</table>

User constraints as limiting the costs or the number of adaptation actions have a decreasing effect on the runtime of the adaptation process, as adaptation branches can be bounded in an early state. The statements for the impacts cannot be generalized, as the factors interfere with each other, and strongly depend on the network infrastructure, the adaptation points and the user constraints. But the investigations show some principles to assess the influence factors.

In addition to the investigation of impact factors to the runtime, an comparison between a brute force approach and the target-oriented adaptation process in Section 7.4.3 showed, that the optimized approach extremely decreases the number of required analysis, while still providing all cost-optimal solutions.
8. Conclusion

This chapter concludes this master thesis and considers the following topics. Section 8.1 summarizes this thesis and highlights the contributions of this work. The limitations of this approach are described in Section 8.2. Section 8.3 introduces possibilities for future extensions on this work.

8.1. Summary

Emerging technologies like cloud computing require dynamic and flexible IT infrastructures. Resources are allocated on demand by customers, to provide scalable services. Data center providers are faced with a variety of requirements for such infrastructures. For customers it is not only relevant that the services are available, also the Quality of Service (QoS) is important. Examples for QoS metrics are response time, transmission delay, throughput and reliability. The expectations on these metrics are commonly agreed between the data center provider and the customer in a contract, the so called Service Level Agreement (SLA).

The dynamic resource allocation in data centers could affect the QoS metrics of the adapted service, as well as other services, which share the same hardware resource. Therefore, it is not sufficient to ensure the fulfillment of the agreed QoS characteristics only on instantiation of new services, instead the SLAs have to be continuously monitored and assessed. If a SLA violation is detected, the data center provider has to react, to solve the violation and fulfill the customers performance expectations.

Two approaches can be identified to solve a SLA violation. Firstly, a network expert analyze the network and make an educated guess of how to solve the violation. But on complex networks with many parameters with mutual influences, it is even for network experts difficult to find a, especially, cost-optimal solution. A second approach is the automatic detection of SLA violations and suggestion of appropriate solutions. As the whole process, from SLA violation up to the execution of an adaptation, could be automatically executed without any human interaction, this approach suits well for self-aware environments. The analysis and the adaptation process can be executed on a system model, to evaluate the solution, before applying it to a real system.

Model-based solutions for computing resources already exist, like the adaptation framework of the Descartes Modeling Language (DML). But for data center networks such
approach is unknown so far. Some tools focus on the automatic suggestion and adaptation of particular metrics, but an approach for a variety of metrics and especially for a wide range of objectives is not known. The model-based approach for analyzing and optimization of networks for automatic suggestion of network adaptations based on violation is scope of this thesis.

The model-based adaptation approach requires the specification of multiple models. For a definition of the network infrastructure, including network configuration and workload, the Descartes Network Infrastructures Modeling (DNI) language is used. Some required extensions to this meta-model are introduced in Section 4.1. The available adaptations for entities and parameters are described in the adaptation points model, shown in Section 4.3. The costs for each adaptation are defined in a sophisticated multi-dimensional cost model, which is included in the adaptation points model and depicted in Section 4.2. The SLAs are also modeled as described in Section 4.4 and are provided as input model for the adaptation process to validate the analysis result against the SLAs. The discovered adaptation actions are not applied to the network model directly, instead they are modeled in a so called adaptation plan, introduced in Section 4.5. A generic analysis result model, described in Section 4.6, is developed, to generalize the output of analyzers to process the results independently from the used solver. These models are necessary for the model-based adaptation process.

The adaptation process is inspired by several concepts. Challenges for the adaptation process are a target-oriented adaptation, an acceptable runtime even on large network models and the cost-optimality of the returned solutions. A MAPE-K adaptation control loop, described in Section 5.1.1, is used to iteratively adapt the network model. This includes among others an analysis phase for detecting violations and a planing phase for discovering possible adaptations. A knowledge component stores central accessible models like the adaptation points. The Strategies/Tactics/Actions (S/T/A) approach, introduced in Section 5.1.2, enables a target-oriented adaptation. Through a strategy and tactic selection only such adaptation algorithms are executed, which could possibly solve the violation. As it is difficult to predict the performance of a specific adaptation, several different adaptation actions can be tracked in multiple adaptation branches through a Branch and Bound algorithm, described in Section 5.1.3. Besides the branching, useless branches, e.g., the ones exceeding the maximal costs, are bounded through several filter modules. These filters reduce the number of adaptation branches and reduce the runtime of the adaptation process. The adaptation process is developed as a pipeline, consisting of multiple modules, introduced in Section 5.1.4.

The models, introduced in Chapter 4, and the adaptation process, described in Chapter 5, are implemented as a Java application. Implementation details of the DNI adaptation framework are given in Chapter 6. The adaptation framework is public available.

The evaluation for adaptation process is described in Chapter 7. In Section 7.1, the requirements and assessment of languages and framework for adaptation models are evaluated. Vogel and Giese specified these requirements, which any adaptation language should provide. It has been shown that the most requirement are fulfilled by the developed adaptation process. As the adaptation only operates on a model without applying the adaptations to a real systems, some of these requirements could not be evaluated. Section 7.2 describes two network infrastructure models in different sizes for an experimental setup, required for the subsequent evaluations. The adaptation process has to fulfill several requirements, as the tracking of multiple adaptation branches or finding the cost-optimal solutions. These qualitative requirements are evaluated in Section 7.3. The scalability of the approach, other performance impacts and the effect of the target-oriented approach in relation to the runtime are investigated in a quantitative
evaluation in Section 7.4. It has been shown that the number of required analyses remains constant on increasing networks, assuming the same variety of available adaptations and SLA violations. The number of analyses is a significant factor for the runtime, as simulations commonly take the most time in the adaptation process. In one experiment, the target-oriented adaptation process has been compared to a brute force approach. This comparison has shown that the optimized approach required only 0.3 percent of analyses to discover the cost-optimal solutions, compared to the brute force algorithms, which considered all possible solutions. The runtime of the adaptation process took only 24 seconds compared to 6379 seconds for the brute force approach. These values strongly depend on the variety of adaptation points, but it is assumed that especially in large networks the variety of adaptation points would increase.

The contributions of this thesis are summarized as follows. At first, models required for the definition of the available adaptations, and models for the adaptation process have been created. Secondly, a target-oriented adaptation process has been developed, focusing on runtime and cost-optimality. The third contribution is the implementation of the models and the process in an adaptation framework.

Section 1.4 defined several goals and research questions, which have been achieved and answered during this work. The statements for each goal and research question are given after the section, which relates to the subject of the goal and question. Additionally, the optional goal and research questions have been fulfilled and answered in Chapter 7. Therefore, all goals have been accomplished.

8.2. Limitations

The introduced adaptation process also contains some limitations, which can be divided into restrictions through the used DNI network infrastructure model and limitations by the adaptation framework.

The latest version of DNI contains a bug in the transformation of SDN flow rules. Therefore, currently no SDN networks can be analyzed and the adaptation of such networks is not supported. As soon as this bug is fixed, the adaptation points model and the adaptation process can be extended by SDN features. The model transformation to the current used SimQPN solver is only calibrated for throughput metric. As the solver is extended to other metrics like latency, also such objectives can be observed. The adaptation process has to be extended to support such additional metrics. Additionally, all limitations from the DNI network infrastructure meta-model apply, as described by Rygielski in 2017. [18].

The adaptation process contains also some limitations. As currently only SimQPN supports the generic analysis result, introduced in Section 4.6 only this solver can be used for the adaptation process. As soon as transformations from other solver specific analysis result to the generic analysis result are available, they can be used as alternative to SimQPN. The implementation of SimQPN allows only one simulation at a time in a single thread. As SimQPN does not support the execution of multiple simulations in parallel, the adaptation process is currently limited to a single thread. However, the implementation uses thread pool executors, which allow increasing the thread count, if other solvers are used or SimQPN supports multi-threading in future.

Based on the described limitations, the current model implementation and adaptation process is also restricted to this. Extending the DNI core meta-model by adding new entities or parameters, or adding new metrics to the SLAs model, the adaptation process has also to be adapted.
An additional limitation are the implemented reference tactics. Currently, only operations with direct impact on the violating entity are supported. For example, rerouting the traffic to avoid an overloaded link would also be a valid adaptation, indirectly solving a [SLA] violation. Such operations are not supported by the tactics, but it is one of the strengths of the adaptation process that such, more sophisticated tactics, can be easily included as described in Section 5.3.

8.3. Future Work

The developed adaptation models and adaptation process introduces several possibilities for future extensions.

As already mentioned in Section 8.2, the current implemented reference tactics only support adaptations with direct impact. The power of the adaptation process increases with an increasing number of adaptation tactics. Extending the adaptation framework by additional tactics in future, would add more alternatives to solve an [SLA] violation. The development of additional tactics means especially using existing optimization approaches for some particular metrics and include them into the framework.

The addition of more solvers would also contribute to a more sophisticated framework. Existing solvers like OMNeT++ could be added through an adapter, providing the generic analysis result, or new solvers could be developed. Development of fast mathematical methods to solve [DNI] network infrastructure models would massively decrease the required runtime.

More metrics, like latency, could be added to the analysis and objectives to be observed and derive an automatically adaptation for this. This requires an extension of the [DNI] transformation to the specific solvers and a calibration step. For such upgrade, a corresponding objective has to be defined for the [SLA] model and the adaptation process has to be extended to support the additional metric.

The adaptation framework requires a [DNI] network infrastructure model as input. Extracting such a model, especially the current workload, from a real-world network would facilitate the use of the adaptation framework. This has to be implemented as a separate preprocessing step before executing the adaptation process.

The adaptation framework outputs adaptation plans with adaptation actions, which solve the detected [SLA] violation. By network [APIs] such adaptation actions, if they are executable by software, could be applied to a real-world network. In conjunction with the automatic extraction of the [DNI] network infrastructure model, this enables a full automatically adaptation process.

For the adaptation process it is irrelevant, from where the modeled traffic in the [DNI] network infrastructure model comes from. Instead of using the current workload, also a forecaster can be used to predict a future workload. This enables a proactive approach, by adapting the network before a violation occurs.

The scope of [DNI] is limited to network infrastructure. Alternative model-based approaches, like the sister project [DML] focus on computing resources. Linking the adaptation processes of computing resources together with the developed network adaptation process would enable an integrated adaptation process for networks and computing resources.
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>MDA modeling and meta-modeling layers</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Methods for performance assessment</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>A queuing place and its shorthand notation</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Control plane in traditional network devices and SDN</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>Dependency structure in DNI meta-model</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>Entity structure in DNI meta-model</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>Network infrastructure as root element in DNI meta-model</td>
<td>16</td>
</tr>
<tr>
<td>2.8</td>
<td>Network structure of DNI meta-model</td>
<td>16</td>
</tr>
<tr>
<td>2.9</td>
<td>Network interface structure in DNI meta-model</td>
<td>17</td>
</tr>
<tr>
<td>2.10</td>
<td>Network traffic structure in DNI meta-model</td>
<td>18</td>
</tr>
<tr>
<td>2.11</td>
<td>Network configuration structure in DNI meta-model</td>
<td>19</td>
</tr>
<tr>
<td>2.12</td>
<td>Reference architecture for MAPE-K adaptation control loop</td>
<td>22</td>
</tr>
<tr>
<td>3.1</td>
<td>Workflow of STA adaptation process</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Mapping between MAPE-K control loop and STA process</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Extension of DNI performance specification entity by identifier</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Inheritance structure of ITypedEntity</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Extend DNI entities by entity types</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>Meta-model of Cost</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>Example cost model instance</td>
<td>37</td>
</tr>
<tr>
<td>4.6</td>
<td>Meta-model of Adaptable Dependencies</td>
<td>39</td>
</tr>
<tr>
<td>4.7</td>
<td>Meta-model of adaptation points</td>
<td>40</td>
</tr>
<tr>
<td>4.8</td>
<td>Example model instance of adaptable dependencies</td>
<td>42</td>
</tr>
<tr>
<td>4.9</td>
<td>Example model instance of adaptable entities</td>
<td>46</td>
</tr>
<tr>
<td>4.10</td>
<td>Meta-model for SLA definition</td>
<td>47</td>
</tr>
<tr>
<td>4.11</td>
<td>Example object of a minimum switching throughput objective</td>
<td>48</td>
</tr>
<tr>
<td>4.12</td>
<td>Example observation of a minimum switching throughput objective</td>
<td>49</td>
</tr>
<tr>
<td>4.13</td>
<td>Dependency structure used in adaptation plan</td>
<td>51</td>
</tr>
<tr>
<td>4.14</td>
<td>Structure of adaptation plan</td>
<td>52</td>
</tr>
<tr>
<td>4.15</td>
<td>Class-diagramm of generic analysis result</td>
<td>55</td>
</tr>
<tr>
<td>4.16</td>
<td>Class-diagramm of SimQPN result</td>
<td>57</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of adaptation process</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>MAPE-K adaptation control loop</td>
<td>61</td>
</tr>
<tr>
<td>5.3</td>
<td>Cascaded MAPE-K adaptation control loop</td>
<td>62</td>
</tr>
<tr>
<td>5.4</td>
<td>Objective to strategy mapping in DNI adaptation process</td>
<td>64</td>
</tr>
<tr>
<td>5.5</td>
<td>Example strategy to tactic mapping in DNI adaptation process</td>
<td>65</td>
</tr>
<tr>
<td>5.6</td>
<td>Example branching by strategy and action in DNI adaptation process</td>
<td>67</td>
</tr>
<tr>
<td>5.7</td>
<td>Pipeline of modules used in adaptation process</td>
<td>69</td>
</tr>
<tr>
<td>5.8</td>
<td>Pipeline with branch and bound example</td>
<td>70</td>
</tr>
<tr>
<td>5.9</td>
<td>Decision logic of solving model collector</td>
<td>73</td>
</tr>
</tbody>
</table>
5.10. Sequence diagram of tactic executor. ........................................ 73
5.11. Example of redundancy elimination caused on identical adaptation branches. 74
5.12. Decision logic and interaction with history in redundancy eliminator module. 75
5.13. Example of cost optimization bounding. .................................... 77
5.14. Schema of parameter adaptation within a tactic. .......................... 79

6.1. Dependencies of the DNI adaptation framework. .......................... 84

7.1. Structure of experimental network model 1. ................................. 101
7.2. Structure of experimental network model 2. ................................. 102
7.3. Expected adaptation process for the solving model detection. .......... 103
7.4. Adaptation points for the validation of violating entity detection and target-oriented adaptation. ................................................. 104
7.5. Expected adaptation process for the violating entity detection and target-oriented adaptation. ................................................. 104
7.6. Adaptation points for the validation of cost-optimal parameter adaptation. .......................................................... 106
7.7. Adaptation points for the validation of multiple tactics execution. .......... 107
7.8. Expected adaptation process for the multiple tactics execution. .......... 107
7.9. Adaptation points for the validation of cost-limit bounding. .............. 109
7.10. Expected adaptation process for the cost-limit bounding. .................. 109
7.11. Adaptation points for the validation of adaptation count bounding. .... 110
7.12. Expected adaptation process for the adaptation count bounding. ......... 111
7.13. Adaptation points for the validation of cost-optimal solution discovery (Pareto front). ...................................................... 113
7.14. Pareto front generated by the optimized adaptation process. .......... 114
7.15. Adaptation points for the investigation of scalability on experimental network model 1. ................................................. 116
7.16. Adaptation points for the investigation of scalability on experimental network model 2. ................................................. 116
7.17. Adaptation points for the investigation of impact of no cost annotation to the runtime of the adaptation process. ..................... 118
7.18. Adaptation points for the investigation of impact of cost annotations to the runtime of the adaptation process. ..................... 118
7.19. Adaptation points for the investigation of impact of commutated cost annotations to the runtime of the adaptation process. .......... 119
7.20. Adaptation points for the investigation of impact of opposed cost annotations to the runtime of the adaptation process. .......... 120
7.21. Adaptation points for the investigation of impact of number of less alternative parameters to the runtime of the adaptation process. .... 120
7.22. Adaptation points for the investigation of impact of number of many alternative parameters to the runtime of the adaptation process. ...... 121
7.23. Adaptation points for the investigation of impact of number of less alternative entities to the runtime of the adaptation process. .... 121
7.24. Adaptation points for the investigation of impact of number of many alternative entities to the runtime of the adaptation process. ...... 122
# List of Tables

4.1. Entity classes of DNI with their corresponding entity type classes. . . . . 33  
4.2. Example of adaptations with different cost types. . . . . . . . . . . . . . . 35  
4.3. Example of target-depending costs. . . . . . . . . . . . . . . . . . . . . . . 35  
4.4. Example of source- and target-depending costs. . . . . . . . . . . . . . . 36  
6.1. Configuration parameters of DNI adaptation framework. . . . . . . . . . . 86  
6.2. Service level objectives for DNI adaptation framework. . . . . . . . . . . 87  
6.3. Cost constraints for DNI adaptation framework. . . . . . . . . . . . . . . 87  
6.4. Interpretation of the DNI adaptation framework statistic. . . . . . . . . . 88  
7.1. Assessment of language requirements for adaptation models. . . . . . . . 96  
7.2. Assessment of framework requirements for adaptation models. . . . . . . 99  
7.3. Strategy-to-tactic mapping used for evaluations. . . . . . . . . . . . . . . 100  
7.4. Resulting adaptations of the optimized adaptation process to discover cost-optimal solutions (Pareto front). . . . . . . . . . . . . . . . . . . . . . 113  
7.5. Validated requirements of the adaptation process. . . . . . . . . . . . . . 114  
7.6. Observed parameters from the investigation of scalability. . . . . . . . . 117  
7.7. Observed parameters from the investigation of impact of cost annotations to the runtime of the adaptation process. . . . . . . . . . . . . . . . . . . . . 117  
7.8. Observed parameters from the investigation of impact of opposed cost annotations to the runtime of the adaptation process. . . . . . . . . . . . . . . . . . . . 119  
7.9. Observed parameters from the investigation of impact of number of alternative parameters to the runtime of the adaptation process. . . . . . . . . . . . . . . . . . . . 120  
7.10. Observed parameters from the investigation of impact of number of alternative entities to the runtime of the adaptation process. . . . . . . . . . . . . . . . . . . . 121  
7.11. Comparison between brute force and optimized adaptation approach. . . 123  
7.12. Summary of impact factors to the runtime of an adaptation process. . . . 124
## Listings

6.1. Example invocation of DNI adaptation framework in Java. . . . . . . . . . 93

A.1. Example configuration file for DNI adaptation framework. . . . . . . . . . 143

A.2. Example agreement repository file for DNI adaptation framework. . . . . 143

A.3. Example cost constraints file for DNI adaptation framework. . . . . . . . 144

B.4. Example resulting adaptation plan from DNI adaptation framework. . . . 144

B.5. Example resulting statistic from DNI adaptation framework. . . . . . . . 146
Acronyms

API Application Programming Interface. [35] [61] [63] [65] [88] [100] [128]

BFS Breadth First Search. [21]

BnB Branch and Bound. [7] [20] [21] [71] [76] [126]

BNF Backus Normal Form. [9]

CDF Cumulative Distribution Function. [14]


CPU Central Processing Unit. [54]

DES Discrete Event Simulation. [11] [27]

DFS Depth First Search. [21]

DML Descartes Modeling Language. [12] [20] [23] [26] [60] [65] [66] [125] [128]

DNI Descartes Network Infrastructures Modeling. [v] [vii] [viii] [2] [5] [7] [12] [14] [20] [23]
[31] [34] [36] [38] [39] [41] [43] [45] [46] [49] [56] [61] [67] [70] [72] [76] [77] [80] [81] [83] [88] [90] [93]
[95] [100] [103] [105] [106] [122] [126] [128]

EMF Eclipse Modeling Framework. [9] [93]

FDDI Fiber Distributed Data Interface. [19]


HPC High-performance Computing. [13]

IP Internet Protocol. [19] [27]

IT Information Technology. [7] [20] [23] [27] [125]

LQN Layered Queueing Networks. [26]

LQNS Layered Queueing Network Solver. [20]

LQSIM Layered Queueing Network Simulator. [20]

LTE Long Term Evolution. [27]
MAPE  Monitor, Analyze, Plan und Execute. 28 60

MAPE-K  Monitor-Analyze-Plan-Execute over a shared Knowledge. 3 7 21 24 60 62 68 70 71 81 100 126

MDA  Model Driven Architecture. 9

MDE  Model Driven Engineering. 7

MiniDNI  Mini Descartes Network Infrastructures Modeling. 19

MOF  Meta-Object Facility. 9

MTU  Maximum Transmission Unit. 18

NFV  Network Function Virtualization. 2

NS-3  NS-3 Network Simulator. 23 27

NSGA-II  Non-dominated Sorting Genetic Algorithm-II. 26

OMG  Object Management Group. 9

OMNeT++  Objective Modular Network Testbed in C++. 20 27 53 128

PCM  Palladio Component Model. 26 27

PN  Petri net. 11 12

QoS  Quality of Service. 5 vii 1 3 7 8 14 24 25 27 46 96 125

QPME  Queueing Petri net Modeling Environment. 93

QPN  Queueing Petri Net. 7 11 12 56

S/T/A  Strategies/Tactics/Actions. 23 24 26 30 34 35 36 44 46 70 71 76 81 122 126

SDN  Software Defined Networking. 2 7 12 14 16 17 19 20 32 127

SFP  Small Form-factor Pluggable. 32 35 36

SimQPN  Simulator for Queueing Petri Nets. 7 11 12 20 31 53 56 57 62 71 83 86 93 104 127

SLA  Service Level Agreement. 5 vii 1 4 7 8 14 24 27 31 44 49 59 64 66 68 71 72 74 77 79 81 89 91 92 96 98 100 102 103 105 106 108 110 111 117 119 122 124 128

SLO  Service Level Objective. 7

SMTP  Simple Mail Transfer Protocol. 8

SPN  Stochastic Petri Net. 11 12

TCO  Total Cost of Ownership. 34 35 36 37

TCP  Transmission Control Protocol. 19

UID  Unique Identifier. 1 5 32 33 43 48 53 54 65 88

UML  Unified Modeling Language. 9
vCPU  virtual Central Processing Unit. 25
VM  Virtual Machine. Ⅴ vii 2 17 25 27
Wi-Fi  Wi-Fi. 27
WiMAX  Worldwide Interoperability for Microwave Access. 27
XML  Extensible Markup Language. 9 85 87 89 93
Bibliography


Appendix

A. Adaptation Framework Input Examples

Listing A.1: Example configuration file for DNI adaptation framework.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<config>
  <bound-on-agreement-fulfillment>true</bound-on-agreement-fulfillment>
  <bound-worse-cost-models>true</bound-worse-cost-models>
  <maximum-adaptations>-1</maximum-adaptations>
  <solving-adaptationplan-output-dir>logs/experiment20/solutions</solving-adaptationplan-output-dir>
  <statistic-file>logs/experiment20/statistic.txt</statistic-file>
  <strategy-tactic-map>
    <entry strategy="INCREASE_SWITCHING_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.IntermediatePerformanceUpgradeTactic"/>
    <entry strategy="INCREASE_SWITCHING_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.IntermediatePerformanceReplacementTactic"/>
    <entry strategy="INCREASE_SWITCHING_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.NodeReplacementTactic"/>
    <entry strategy="INCREASE_INTERFACE_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.NetworkInterfacePerformanceUpgradeTactic"/>
    <entry strategy="INCREASE_INTERFACE_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.NetworkInterfacePerformanceReplacementTactic"/>
    <entry strategy="INCREASE_INTERFACE_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.NodeReplacementTactic"/>
    <entry strategy="INCREASE_LINK_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.LinkPerformanceThroughputUpgradeTactic"/>
    <entry strategy="INCREASE_LINK_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.LinkPerformanceReplacementTactic"/>
    <entry strategy="INCREASE_LINK_THROUGHPUT"
           tactic="tools.descartes.dni.adaptation.tactic.LinkReplacementTactic"/>
  </strategy-tactic-map>
</config>
```
Listing A.2: Example agreement repository file for DNI adaptation framework.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<agreement-repository>
  <agreement name="customer1">
    <objective xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="objectiveNetworkInterfaceThroughput" flow="File2GB" min-throughput="992195814"/>
  </agreement>
</agreement-repository>
```

Listing A.3: Example cost constraints file for DNI adaptation framework.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cost-constraints>
  <constraint type="HANDLINGTIME" limit="3600.0"/>
  <constraint type="INVESTMENT" limit="500.0"/>
</cost-constraints>
```

B. Adaptation Framework Output Examples

Listing B.4: Example resulting adaptation plan from DNI adaptation framework.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <link action="REPLACE" originUid="Link-SW10-C11" typeUid="LinkTypeB"/>
  <link action="REPLACE" originUid="Link-SW30-SW10" typeUid="LinkTypeB"/>
  <link action="REPLACE" originUid="Link-SW40-S1" typeUid="LinkTypeB"/>
  <link action="REPLACE" originUid="Link-SW40-SW30" typeUid="LinkTypeB"/>
  <linkPerformance action="REPLACE" originUid="Link-SW10-C11-Perf" typeUid="LinkPerformanceTypeB">
    <maximalSupportedBandwidth xsi:type="constantLongVariable" uid="_OSwsQJ-REee3vPB2Qcb7uA" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
      <value>10</value>
    </maximalSupportedBandwidth>
    <propagationDelay xsi:type="variable" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  </linkPerformance>
  <linkPerformance action="REPLACE" originUid="Link-SW30-SW10-Perf" typeUid="LinkPerformanceTypeB">
    <maximalSupportedBandwidth xsi:type="constantLongVariable" uid="_OSwsQJ-REee3vPB2Qcb7uA" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
      <value>10</value>
    </maximalSupportedBandwidth>
    <propagationDelay xsi:type="variable" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  </linkPerformance>
  <linkPerformance action="REPLACE" originUid="Link-SW10-C11-Perf" typeUid="LinkPerformanceTypeB">
    <maximalSupportedBandwidth xsi:type="constantLongVariable" uid="_OSwsQJ-REee3vPB2Qcb7uA" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
      <value>10</value>
    </maximalSupportedBandwidth>
    <propagationDelay xsi:type="variable" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  </linkPerformance>
  <linkPerformance action="REPLACE" originUid="Link-SW30-SW10-Perf" typeUid="LinkPerformanceTypeB">
    <maximalSupportedBandwidth xsi:type="constantLongVariable" uid="_OSwsQJ-REee3vPB2Qcb7uA" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
      <value>10</value>
    </maximalSupportedBandwidth>
    <propagationDelay xsi:type="variable" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  </linkPerformance>
  <linkPerformance action="REPLACE" originUid="Link-SW10-C11-Perf" typeUid="LinkPerformanceTypeB">
    <maximalSupportedBandwidth xsi:type="constantLongVariable" uid="_OSwsQJ-REee3vPB2Qcb7uA" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
      <value>10</value>
    </maximalSupportedBandwidth>
    <propagationDelay xsi:type="variable" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
  </linkPerformance>
</adaptationPlan>
```
<maximalSupportedBandwidth>

<propagationDelay xsi:type="variable" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</linkPerformance>

<networkInterface action="REPLACE" originUid="SwitchSW30-e01" typeUid="NetworkInterfaceTypeC"/>
<networkInterface action="REPLACE" originUid="SwitchSW30-e02" typeUid="NetworkInterfaceTypeC"/>
<networkInterface action="REPLACE" originUid="SwitchSW30-e03" typeUid="NetworkInterfaceTypeC"/>
<networkInterface action="REPLACE" originUid="SwitchSW40-e01" typeUid="NetworkInterfaceTypeB"/>
<networkInterface action="REPLACE" originUid="SwitchSW40-e02" typeUid="NetworkInterfaceTypeB"/>
<networkInterface action="UPDATE" originUid="ClientC11-eth0-Perf">
<interfaceThroughput xsi:type="constantLongVariable" uid="_L-3DUJ8Kee11R1$95UJfQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
<value>10</value>
</interfaceThroughput>
</networkInterfacePerformance>
<networkInterfacePerformance action="UPDATE" originUid="ServerS1-eth0-Perf">
<interfaceThroughput xsi:type="constantLongVariable" uid="_L-3DUJ8Kee11R1$95UJfQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
<value>10</value>
</interfaceThroughput>
</networkInterfacePerformance>
<networkInterfacePerformance action="UPDATE" originUid="SwitchSW10-e01-Perf">
<interfaceThroughput xsi:type="constantLongVariable" uid="_L-3DUJ8Kee11R1$95UJfQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
<value>10</value>
</interfaceThroughput>
</networkInterfacePerformance>
<networkInterfacePerformance action="UPDATE" originUid="SwitchSW10-e03-Perf">
<interfaceThroughput xsi:type="constantLongVariable" uid="_L-3DUJ8Kee11R1$95UJfQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
<value>10</value>
</interfaceThroughput>
</networkInterfacePerformance>
<networkInterfacePerformance action="REPLACE" originUid="SwitchSW30-e01-Perf" typeUid="NetworkInterfacePerformanceTypeC">
<interfaceThroughput xsi:type="constantLongVariable" uid="_FvPfYJ8D5ee11R1$95UJfQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
<value>10</value>
</interfaceThroughput>
</networkInterfacePerformance>
<networkInterfacePerformance action="REPLACE" originUid="SwitchSW30-e02-Perf" typeUid="NetworkInterfacePerformanceTypeC">
<interfaceThroughput xsi:type="constantLongVariable" uid="_FvPfYJ8D5ee11R1$95UJfQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
<value>10</value>
</interfaceThroughput>
</networkInterfacePerformance>
<networkInterfacePerformance action="REPLACE" originUid="SwitchSW30-e03-Perf" typeUid="NetworkInterfacePerformanceTypeC">
<interfaceThroughput xsi:type="constantLongVariable" uid="_FvPfYJ8D5ee11R1$95UJfQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<unit xsi:type="speedUnit" speed="BITS_PER_SECOND" unitPrefix="G"/>
<value>10</value>
</interfaceThroughput>
</networkInterfacePerformance>
Listing B.5: Example resulting statistic from DNI adaptation framework.

```
Adaptation Framework
1 duration 6379.264 sec
2 solving models 231

Module: Analyzer
3 number of analysis 3190 runs
4 avg. analysis duration 1.998 sec

Module: AgreementViolationDetector
5 number of non violating branches 231
6 number of runs 3190
7 number of violating branches 2959

Module: SolvingModelCollector
8 bound on agreement fulfillment false
9 number of solving models 231

Module: RedundancyEliminator
10 number of history hits / discarded branches 116 hits
11 history size 3189 ad. plans

Module: MaximumAdaptationSupervisor
12 bounded branches 0
13 max adaptations infinite

Module: CostConstraintBounder
14 cost violations / discarded branches 0
15 cost analysis 3189

Module: CostOptimizationBounder
16 cost analysis 3189
17 bounded branches 0
```