University of Twente

Final Project

Monitoring Service-level Agreements for Logistics Service Providers

Author:
Tim Kerkhoven

Supervisors:
Dr. Luís Ferreira Pires
Leon R. de Vries, MSc
Dr.Ir. Vadim Zaytsev

September 21, 2023

Contents

Chapter 1

Introduction

This thesis investigates the possibility of automating the monitoring of Service-Level Agreements (SLAs) for business service providers, using logistics service providers our main example. This chapter provides the motivation and problems that drive the project, its objectives, the approach to reach those objectives, as well as the structure of the thesis.

1.1 Motivation

Logistics service providers provide services to support the transport of goods. As a late transport of goods could, for example, hold up an entire factory, reliability of the service is important. Therefore many service providers use Service-Level Agreements (SLAs), which define a commitment between a service provider and a client about particular aspects of the provided service. The commitments are often about timing requirements of the service provision, e.g., the time it should take before a shipping container is put on a sea ship after it reaches the service provider's inland terminal. An SLA can also be used to determine which party is responsible when additional costs are incurred, such as the fine when a container is left too long at a port.

Currently, these SLAs are checked manually using data exports, which requires a large time investment. As this is generally done at the end of the month, there is no real-time monitoring to check whether the service provision is within the boundaries of the SLA. For example, a planner cannot easily see if a container should be on a train tomorrow or if the next day is still within the bounds of the agreement. Therefore, it is much preferred if the monitoring of SLAs could be done automatically and in real-time by an automated system, with a way for users of the system to be aware if the actions they take violate an SLA or not. This would enable upfront identification of some problems, instead of the current analysis afterwards, which could improve the quality of service delivered.

For example, Cofano Software Solutions is a company that offers end-to-end logistics solutions [1]. They develop Stack, which is a cloud application that helps sea and inland terminals plan shipping container transports and provides functionality for managing their terminals [2]. The system is built around the transport of shipping containers. A complete usage generally consists of several

separate parts, which together model a single use of the container, e.g., picking up a container from a sea ship, moving it to the container terminal, putting it on a truck, moving it to a customer, unloading, moving it back to the container terminal, ventilating it, and putting it in depot. Cofano's Stack will be used as a case study for this project.

Logistics service providers are not the only ones providing a mix of physical and digital services. In principle, many providers of business services could benefit from a generalised way of automating SLA monitoring.

1.2 Problem Statement

Two problems were identified in Section 1.1: the cost of manual SLA monitoring and the lack of real-time SLA monitoring with decision support for business services, with the former having the larger current impact and being the main focus of this thesis. An automated SLA monitoring system is suggested as a solution to this problem. To perform automated monitoring of SLAs, however, a specification of the SLA is required, as the system needs to be able to interpret SLAs. A proper SLA specification, in turn, requires a decent understanding of services and SLAs.

1.3 Objectives

To solve the main problems stated in Section 1.2, an automated SLA monitoring system is required. The aim of this thesis is to demonstrate how automated SLA monitoring can be applied for the specific use case of logistics service providers, and, through generalisation, for the general use case of business services. It is assumed for this solution that the required data is available, i.e. the business service is supported by an IT service. This is shown through a proof of concept application implemented for the example use case of Cofano, upon which a general purpose solution is built.

To reach this aim, to following main request question was defined:

Research Question 1. How to develop a system to automatically monitor SLAs and provide real-time decision support?

This question can essentially be split in two parts: how to describe SLAs, and how to monitor them. Two further questions were constructed:

Research Question 2. What should be the constraints of an SLA specification language?

Research Question 2.1. *What SLA specification languages already exist?*

Research Question 2.2. *How can an SLA specification language be used by a monitoring system?*

Research Question 3. What should an SLA monitoring system look like?

Research Question 3.1. *What SLA monitoring solutions already exist?*

Research Question 3.2. *What should SLA monitoring look like to provide real-time decision support?*

Research Question 3.3. *What should an architecture for an SLA monitoring system look like to provide both normal reporting and real-time decision support?*

1.4 Approach

To answer our research questions we first needed background information about services and SLAs. For services, this was done by finding influential sources in order to learn about their definition. This was followed by a similar process to learn about SLAs, looking again at the definition, but also at their life cycle and contents.

Once sufficient knowledge on services and SLAs was gathered, a way to specify SLAs was devised. This was done by first identifying requirements for an SLA specification language. A literature study was then performed to find existing SLA specification languages, which were then compared to each other and analysed with respect to the identified requirements. Using the results of the comparison, an SLA specification was designed. This entire process is based on Wieringa's design cycle [3].

Finally, the automated SLA monitoring system was designed, again using Wieringa's design cycle [3]. First off, monitoring to provide real-time decision support was properly defined: what should this accomplish? Then, requirements for the system were identified based on the goals of the system. Next, a literature study was performed to identify existing SLA monitoring solutions. These were analysed with respect to the requirements, to find if any of them are suitable. Then, a new system was designed, based on suitable other systems. A prototype of this system was implemented, and both it and the SLA specification were verified.

1.5 Structure of the Report

This thesis first provides a background for SLAs in Chapter 2. It then shows the state of practice in Chapter 3. Afterwards it discusses the requirements and context of the SLA specification language in Chapter 4, followed by the design and implementation of the language in Chapter 5. Chapter 6 then discusses the requirements and context of the SLA monitoring system, the design and prototype of which is described in Chapter 7. The verification of the system is discussed in Chapter 8. Finally, the conclusions of this thesis are presented in Chapter 9.

Chapter 2

Background

This chapter provides some background information for SLAs. It will first give a definition for services and SLAs. Then an overview of the concepts and components of SLAs is provided. Finally, it will describe the SLA life cycle.

2.1 Services

Nardi *et al.* identifies that a "service" can be seen from various perspectives, each emphasising different aspects [4]. They claim that although the notion of a service seems intuitive, it is far from trivial. It is, in fact, a case of systematic polysemy, where a word has multiple related meanings. For example, depending on context, "service" could mean both service offering and service delivery.

This research will use the following very loose definition of a service: a commitment to do something between parties. The reasons for using such a poor definition is that for this research the nature of the service is not important, and constricting it might provide issues later on. Nardi *et al.* also shows that defining a service is a rather complex matter.

The life cycle of a service can be described as three main phases: the service offer, the service negotiation, and the service delivery [4]. In the service offer, the services are presented to target customers, and important aspects are described and published. The service negotiation is about establishing an agreement between a customer and a provider about their responsibilities. Finally, in the service delivery phase, the agreed upon commitments are fulfilled.

2.2 SLA Definition

Verma provides the following description of an SLA: "A service level agreement (SLA) is a formal definition of the relationship that exists between a service provider and its customer" [5]. Sprenkels and Pras, Maarouf *et al.* provide similar definitions, with Sprenkels and Pras also describing bilateral SLAs among pairs of organisations, in which case each organisation provides a service to the other [6], [7]. An SLA defines not only what is expected of the service provider, but also what is expected of the customer. The service provider and customer can also come from the same organisation, with one department providing a service to another, as described by Beaumont [8]. Finally, Karten emphasises the

potential of SLAs to clarify responsibilities, strengthen communication, reduce conflict, and build trust [9].

In this paper the following definition of an SLA will be used: "A binding agreement between a service provider and one or more other parties over a provided service".

2.3 Concepts & Components

According to Paschke and Schnappinger-Gerull, an SLA contains technical, organisational, and legal information, with the most common components being [10], [11]:

Involved parties Role references within the SLA, can be either signatory parties or supporting parties.

Contract validity period Specifies how long the SLA is valid and enforceable.

Service definitions Specifies the service functionality, components, and observable parameters.

Service Level Objectives Quality of service guarantees that must be met by a specific obliged party. They can have validity periods, qualifying conditions on external factors such as time of day, as well as the conditions that a party must meet.

Action guarantees A commitment that a particular activity is performed by an obliged party if a given precondition is met. This includes compensations, rewards, recovery, and management actions.

These components are similar, but not exactly the same as the most common SLA components identified by Verma [5], [6], [12]:

Description of the service to be provided. Including the type of service and any qualifications of the type of service to be provided.

Expected performance of the service. Specifically describing its reliability and responsiveness. Reliability includes availability requirements and responsiveness includes how soon actions related to the service provision should be performed in the normal course of operations, which should also be defined in the SLA. Metrics used should be objective and measurable.

Procedure for handling problems with the service. This includes all information necessary to resolve possible problems. It typically also describes a time frame for the response and problem resolution.

Process for monitoring and reporting the service level. Describes who performs monitoring, what (types of) statistics are collected, how often they are collected, and how they may be accessed.

Consequences for a party not meeting its obligations. Depends on the nature of the relationship between the parties. Typical options include reimbursement, fines, or the ability to terminate the relationship.

Escape clauses and constraints. Describes under which conditions the SLA does not apply, or when it would be considered unreasonable to meet the requirements set in the SLA. Often contains constraints on the customer behaviour, i.e. conditions under which the service provider may void the SLA.

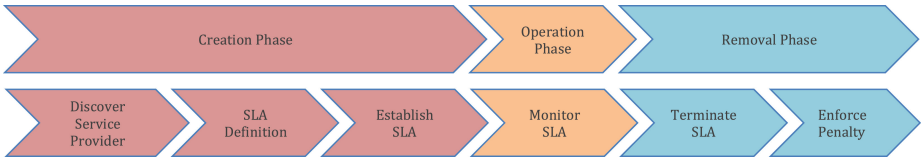

Figure 2.1: SLA life cycle shown by Lu *et al.* [13].

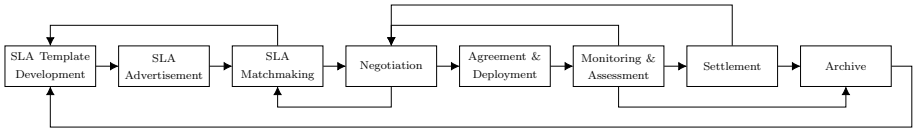

Figure 2.2: SLA life cycle proposed by Kritikos *et al.* [11].

Not all components are present in every SLA, but a suitable SLA should provide an overview of the different items that can go wrong with the provided service and the measures to cover those situations [5]. Karten even advocates starting with a limited-scope SLA that grows over time [9].

2.4 SLA Life Cycle

To better understand SLAs, their life cycle, which is related to the life cycle of a service, should be considered. Sprenkels and Pras, Maarouf *et al.* all identify three high level phases: the creation phase, the operational phase, and the removal phase (see Figure 2.1) [6], [7], [13]. These three phases are expanded into six more detailed phases by Maarouf *et al.* [7]. This division into phases helps structure the modelling of SLAs into separate parts.

Kritikos *et al.* proposes a different, more detailed life cycle, which is shown in Figure 2.2. It clearly shows multiple transitions between states that go backwards or skip parts of it. Although no higher level phases are shown, the states of this life cycle can be categorised in the same three high level phases shown in Figure 2.1. Both life cycles are further described below.

2.4.1 Creation Phase

Figure 2.2 starts the creation phase of an SLA with the development of an SLA template, describing the service, which is then used to advertise the service and match with potential clients. These *SLA Advertisement* and *SLA Matchmaking* steps correspond with the purpose of the *Discover Service Provider* step from Figure 2.1. Once a client and service provider decide to collaborate, the service terms of the agreement have to be identified and negotiated, which corresponds to the *SLA Definition* step from from Figure 2.1 and the *Negotiation* step from Figure 2.2. The final step of the create phase of Figure 2.1 is *Establish SLA*. In this step, the SLA template is filled in to form a specific agreement, and both parties commit to it [14]. In the corresponding step, *Agreement & Deployment* from Figure 2.2 the SLA template from its first step is used instead. Both steps include the deployment and configuration of the services and monitoring as agreed upon in the SLA.

As this project does not consider automated negotiation and service deployment, several steps from the proposed life cycle by Kritikos *et al.* are not relevant here. In particular, *SLA Advertisement*, and *SLA Matchmaking* describe mostly automated processes where SLA templates from service providers are used to show their services' capabilities, and are matched with the desired functionality of the client. The same goes for the first step of Figure 2.1.

The parts that are most important to this project are the creation of the SLA template and specific SLA, and the configuration of monitoring. The SLA representation is important, as the monitoring specification has to be derived from it. For this project it is assumed that a specific SLA has been created, agreed upon, and deployed.

2.4.2 Operational phase

Once an SLA has been established and the service provisioning has started, monitoring for SLA violations begins. This is done in step *Monitor SLA* in Figure 2.1 and step *Monitoring & Assessment* in Figure 2.2. There are some concerns with the monitoring, such as which party should be in charge of the monitoring process, and how can fairness be assured [7]. Given the nature of this project, the party in charge in the case of this monitoring system is the service provider. Other options include a trusted third party, or a trusted module on the client side [7], with the former being the best overall option. Fairness can in part be guaranteed because both parties can independently obtain parts of the monitoring information. Other information comes from the service provider, but since the services we are considering are mostly physical, this information can generally be confirmed afterwards, either from another party involved or from the results of the provided service. As the monitoring is done on the side of the service provider, a certain level of trust is required, however. It might be worth it to create a way for the client to independently access the monitoring information of their SLAs as well.

Violations are "*un-fulfillments*" of the agreement, which are defined in European law as defective performance (lower service level than agreed), late performance (service provided with delay), and no performance (service not provided) [7].

There are three broad provisioning categories based on the violation definition: "All-or-Nothing", "Partial", and "Weighted Partial" [7]. In "All-or-Nothing" provisioning, all SLOs must be satisfied, because a single violation leads to complete failure and re-negotiation of the SLA. If only some SLOs are mandatory, "Partial" provisioning only requires those to be met for successful service delivery. In "Weighted Partial" provisioning, the SLOs are assigned a weight, and a certain threshold must be met instead. While violations in the latter two provisioning categories do not need to cause re-negotiation, they may still have other repercussions, such as fines.

During the operational phase it might also be possible for the client to change certain parameters of the SLA. This should be defined in the SLA itself, and all changes need to be mapped to the service and monitoring system [6].

This phase is the main focus of this project: the actual monitoring of SLAs.

2.4.3 Removal phase

There are generally two scenarios that might cause SLA termination: normal time-out, or violation of contract terms [7]. Violations do not necessarily cause termination of the SLA, but when they do, penalties might apply as agreed upon in the SLA. When an SLA is in its removal phase, monitoring should be stopped and any remaining configuration should be cleaned up.

Figures 2.1 and 2.2 take a slightly different approach to this phase: Lu *et al.* differentiates between the termination (*Terminate SLA*) and enforcement (*Enforce Penalty*) steps of this process, whereas Kritikos *et al.* shows those together as the *Settlement* step, followed by an *Archive* step [7], [11], [13]. The archiving step follows from a statutory period where the SLA must be kept as a legal document describing how services where provided [11].

While the removal phase is not directly related to the core of this project, it could be important to support proper archiving and clear information for use in a penalty enforcement step.

Chapter 3

State of Practice

To get a better understanding of the current state of practice, we looked at examples from logistics service providers and try to generalise to business service providers. Several sources were used to get an overview of the state of practice of logistics service providers: an SLA example of a logistics service provider, a Request For Information (RFI) example of a different logistics service provider, two interviews with domain experts, and a brief literature study. This chapter describes the results of our investigation and discusses them.

3.1 Running Example

This section introduces a running example to have a consistent scenario to relate the sources to. The example chosen for this is the import and export of goods through a port. Specifically as seen through the viewpoint of a container terminal.

First of is the import scenario, which is illustrated in Figure 3.1. In this case, the following service is provided: a full container will be transported from the port it arrives in to a location of the customer, where it can be unloaded. The empty container will then be transported to a depot, where its carrier can reuse it. In this process, the following steps are taken:

(1) The service provider receives an order for the service, including required documents
(2) The service provider receives a notice of the estimated time of arrival
(3) The service provider plans the transports for the provided service
(4) The ship carrying the container arrives at the port and is unloaded
(5) The service provider receives documentation for the release of the container
(6) A barge is used to transport the container from the port to the container terminal, where the container is weighed
(7) A truck is used to transport the container to the customer's location
(8) The container is unloaded
(9) The truck returns the empty container to the container terminal
(10) The empty container is transported to a depot via barge

The export scenario is quite similar and provides the following service, as illustrated in Figure 3.2: an empty container is transported to the customer's

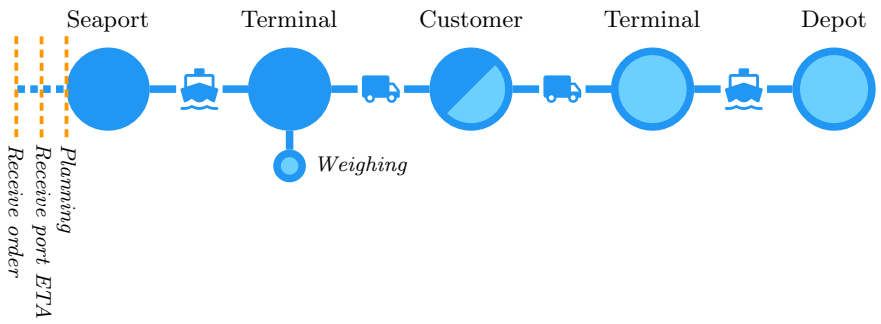

Figure 3.1: Visualisation of an import scenario: a full container is picked up from a sea terminal and returned empty to a depot.

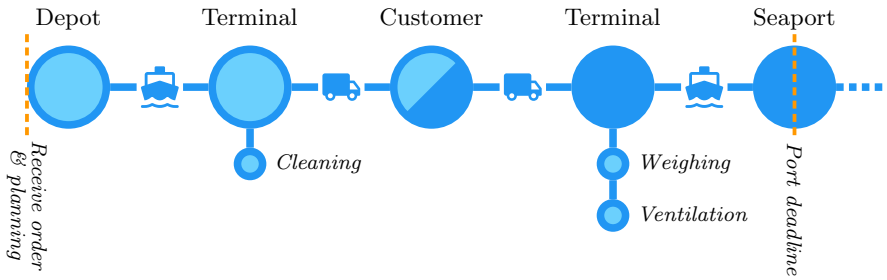

Figure 3.2: Visualisation of an export scenario: an empty container is transported to a customer for loading, then moved to a port.

location, where it can be loaded. The full container will then be transported to the port from where it will be exported. In this process, the following steps are taken:

(1) The service provider receives an order for the service, including required documents and deadline for delivering the container to the port
(2) The service provider plans the transports for the provided service
(3) A barge is used to transport an empty container from a depot to the container terminal, where the container is cleaned
(4) A truck is used to transport the empty container to the customer
(5) The container is loaded
(6) The truck transports the container to the container terminal, where the container is weighed and ventilated
(7) A barge transports the container to the port before the deadline

3.2 SLA

We obtained a recent but expired example SLA from a logistics service provider in the Netherlands that has amongst other things a container terminal and warehouses, and provides transport and storage services with a specialisation in dangerous goods. Besides the SLA, four process flows were provided by them from other SLAs, these are discussed below along the process flows in the example SLA.

The document states that it contains operation information regarding the transport of containers for the customer, within the various modalities that are available at the service provider. Agreements for three provided services are described in the SLA:

(1) Barge transport
(2) Truck transport
(3) Inland terminal activities, including a depot agreement

In this SLA, these services are used to describe the import and export of containers for the customer.

Four parties are involved in the SLA: a client, a debtor, a carrier, and a contractor. The document contains information about the transport of container for the client, on behalf of the debtor, carried out by the carrier and the contractor.

The document contains several sections, solving various problems and clarifying responsibilities and procedures, that are described below:

Process flows The SLA describes two main scenarios, the import and export of goods, which are similar to the examples given in Section 3.1. The provided services (barge transport, truck transport, and terminal activities) are combined to achieve this. Process flows are flowcharts used to clarify and agree upon several things at once:

- Which steps the process consists of
- Which party is responsible for each step in the process
- The timings of each step in the process on a timeline
- Deadlines for certain actions
- Alternatives in the process, such as multiple options for the mode of transport
- The demurrage period, i.e., the time a container is positioned at a seaport before it is picked up (see Figure 3.3)
- The detention period, i.e., the time a container is in the hinterland till its return to the agreed terminal or port (see Figure 3.3)
- Which optional activities can be performed on the terminal, and how much time they will take

The process flows also specify for each party whether days are counted in workdays or calendar days.

Demurrage and detention While the process flows clarify which periods are counted for demurrage and detention, the free period can differ depending on the carrier and type of container. The free period is a number of days for the combined demurrage and detention. When it is exceeded, fees will be charged. To ensure no confusion about these free periods occurs, the SLA lists the free period per carrier per container type.

Transport details and activities There are several additional details and activities described in the SLA. These provide more information about the process flows and procedures used, as well as assurance that certain procedures will be followed.

First off, the SLA prescribes how the contractor should secure and handle cargo. It also prescribes the procedure for the client for expediting container transports with trucks, followed by a list of exceptions that use a truck by default.

The process flows show optional activities performed by the carrier on the terminal. Each of these is described in the SLA, and where needed, procedures and agreements are provided. In the import example (Section 3.1) these activities would be performed between step 6 and 7. An overview of prices for transports and activities is provided as well.
The SLA also provides an indication of the unloading capacity, and the factors upon which it depends.
Finally, the procedure for monitoring demurrage, detention, and other periods by the client and debtor is prescribed.

Communication An SLA should clarify how parties can be contacted, as discussed in Section 2.3. This SLA does that by giving general contact and business information of most of the parties, specific contact information of these parties per department and/or function, as well as contact information for specific purposes. Finally, it also lists delivery addresses.
The SLA also specifies a set of KPIs for both import and export processes. Per KPI it specifies when and with whom it will be shared. This has several purposes, for example the debtor shares import forecasts every month which can be used by the carrier to plan, and the carrier shares lists of transported containers with the debtor for invoicing purposes.

Claims and conditions The SLA describes how claims with liability to the carrier and contractor will be handled with their insurer. It also specifies which insurer and type of insurance are used.
Finally the SLA links to externally defined conditions for the services provided.

3.3 RFI

A request for information (RFI) is a common business process with the purpose of collection information about the capabilities of suppliers. [15] This RFI was used by a company that provides global bulk chemical forwarding services, with a fleet of over 2000 tank containers, to obtain information from Cofano Software Solutions about their software. Part of the document states their requirements for information they need to monitoring their services. This gives a good insight into the metrics that need to be tracked for their SLAs, both internal and external.

Several metrics are mentioned specifically in the RFI. These are listed below, along with examples of their possible uses:

(1) By tracking on time and delayed orders, agreements can be made about a minimum percentage of on time order fulfilment. By also tracking the cause of the delays in order, the responsible parties can be held accountable when a threshold is exceeded.
(2) By tracking the utilisation of equipment, agreements can be made about a minimum percentage of equipment use. By agreeing upon a certain amount of orders, the equipment owner can plan for more optimal equipment utilisation.
(3) By tracking equipment that is damaged, in repair, or total loss, you can for example make agreements about the maximum time it should take to repair equipment after is damaged.

3.4 Interviews

Two domain experts were interviewed to get an indication of the current state of practice, and to gauge interest in and get input for an SLA monitoring system. The first domain expert is both a senior business engineer at a warehousing and logistics company, as well as terminal manager of a container terminal. The second domain expert is a product and implementation manager at Cofano Software Solutions, with previous experience as a planner for a large international transport and warehousing company, and as a key user for a transport management tool.

Several observations about the state of practice of SLAs by logistics service providers can be made from the interviews. The agreements in an SLA are used to provide clarity for all people and/or parties involved about their particular responsibilities. These clarified responsibilities can than be used to resolve conflicts. While some operational components are very common, others vary depending on the specific logistics services provided. The format of the SLAs also differs, some utilising mostly text, others leveraging flow diagrams such as in the example SLA in Section 3.2. Agreements describe both chronological events, as well as (statistics of) data over a period, such as the percentage of tank containers in use, or the number of delayed shipments. They can be about a single container, or a larger set of containers, such as all containers within a period. In general, logistics service providers use templates to create new SLAs, and they often describe a chronological chain of events on a timeline starting from the first event.

Some types of agreements, based on time windows commonly used in the import and export processes, were specifically mentioned during the interviews. They are described below and visualised in Figure 3.3, which shows the import scenario from Section 3.1 annotated with the different time windows.

(1) Demurrage, i.e., the time a container is positioned at a seaport
(2) Detention, i.e., the time a container is in the hinterland till its return to the agreed terminal or port
(3) Container dwell time, i.e., the time a container spends at a terminal
(4) Truck turnaround time, i.e., the total time a truck spends on the terminal area
(5) Delivery/pickup window, i.e., the time window in which products should be delivered or picked up
(6) Lead time, i.e., the total time between placement of an order and its completion
(7) The amount of time required between placing an order and providing the required information, and the start of the fulfilment of the order

Besides agreements about the physical processes, agreements are also made about the provision of information to prevent scenarios such as a container being physically delivered to a terminal while no record of that container can be found in the supporting systems. Interviewees also warned that certain agreements, such as delivery windows or turnaround times, must take the mode of transport into account, and that calendar days and working days are both used in SLAs. Day boundaries are also an important issue, as the period 23:50 till 00:10 spans two days, while 07:00 till 23:00 is in one day.

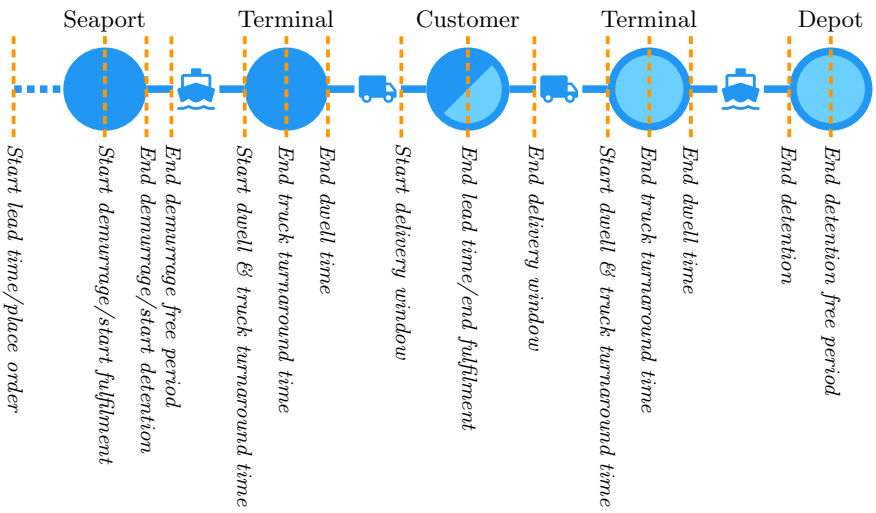

Figure 3.3: Visualisation of different time periods in the transportation process of imported goods.

With regards to the time investment for the service provider, or the person(s) responsible for creating/monitoring SLAs, it depends on the level of automation and the number of metrics that are monitored. One of the interviewees regarded the process of creating and monitoring SLAs as very time intensive. Both interviewees showed interest in using an automated monitoring system for SLAs.

Finally, the interviews yielded some insights in ways monitoring data are inspected. According to the interviews, it is important to show more than a pass/fail as there are often multiple sides to the data. Using a drill down interface would provide easily accessible additional details to get a better picture of what the pass/fail statistics mean. Another perceived benefit is the option to warn or inform people about possible SLA violations. For example, a planner could be warned when a container is planned to arrive overdue according to the SLA.

3.5 Literature

To get more information on SLAs we performed a brief literature review. The review focuses on the contents of SLAs in the logistics sector, as well as indications of useful agreements.

First of, Soomro and Song states that the use of SLAs is crucial for a business to provide services to customers successfully. [16] They can solve the problems of quality of service evaluation during development and deployment. Soomro and Song also claims that existing specifications and structures for SLAs do not fully meet the requirements for businesses.

Second, Metzger *et al.* identified the following key issues in air transport orders: [17]

(1) Late shows
(2) No shows
(3) Data delays

Late shows occur when a delay arises between the scheduled and actual times of arrival of cargo. No shows might occur because of late cancellations, or freight that is not actually available. Data delays occur when physical and digital processes are misaligned, such as when necessary data are provided only after they are required.

While this is not directly related to the examples shown in Section 3.1, they might still be relevant. The described issues are also present with other modes of transport, though their impact might be lesser. For instance, both the late shows and no shows are present in the metrics found in the RFI (see Section 3.3), and the data delays are present in deadlines the the example SLA's process flows (see Section 3.2).

Third, both Gutiérrez *et al.* and Marquezan *et al.* describe the use of frame SLAs and specific SLAs in logistics. [18], [19] Frame SLAs are described as long term SLAs that additionally offer aggregation SLOs over a longer duration. The aggregation SLOs can offer longer term agreements, such as a maximum number of transport orders. They argue that the combination should be better able to provide agreements for both the long and short term.

Fourth, Moini *et al.* indicates that container dwell time can have a considerable impact in a terminal's capacity and revenue earned from demurrage fees. [20] Container dwell time is the time a container spends at a terminal (visualised in Figure 3.3). According to Moini *et al.*, reducing it might be one of the least costly options increase a terminal's capacity. Because of this, including agreements to reduce container dwell time in SLAs is a plausible option.

Finally, Fazi and Roodbergen shows that demurrage and detention fees in combination with inland terminals can a practical option for reducing congestion in seaports, and that these fees are widely used by shipping lines. [21] Because of this, including agreements about demurrage and detention in SLAs, in order to clarify the accountable parties, is a logical step.

Both the container dwell time, and demurrage and detention are also mentioned explicitly in the interviews as being present in SLAs. The latter can also be prominently found in the example SLA provided.

3.6 Discussion

The sources used in this research all indicate that SLAs are widely used by logistics service providers, both externally and internally. Every logistics service provider has SLAs tailored to their specific situation, but many aspects are common in these SLAs. The literature showed that the use of SLAs by others businesses is also widespread. Though many different services are described in SLAs, we assume that many agreements are similarly structured. A language describing SLAs should take these abstract aspects into account and ensure that they can describe all required scenarios.

The interviews indicate that the form of the SLAs should also be taken into account in the implementation of the system, as different service providers have different preferences, e.g., text and flow charts. The example SLA, for instance, contains a very detailed flow chart, with a less complete textual description. This is also something that should be kept in mind for the representation of SLAs.

From the interviews it seems there is time saving potential in an integrated environment to design and monitor SLAs. Both interviewees showed significant

interest in such a system, indicating that there is a practical need for it. The literature supports our problem statement, claiming that the existing solutions are insufficient for businesses. This indicates that other business service providers might be in a similar position, where an SLA design and monitoring system would be desirable.

In the case of the logistics sector, many of the agreements in SLAs seem to be focused around the specific timing of actions. We assume this is a more generally applicable, and a generic way of describing these scenarios, amongst others, would be needed to provide a way to describe SLAs for them.

These findings of the state of practice in the logistics sector, and to a lesser degree other businesses, should be used to define the requirements of an SLA monitoring solution that addresses the problems found.

Chapter 4

SLA Language Context

This chapter describes the requirements and environment for the SLA domain specific language used in the monitoring system. It first derives requirements for such a language from the earlier chapters, then looks at related work and how they relate to the requirements. Next, an approach to designing an SLA domain specific language is given, followed by a discussion about the chosen tools for the creation of the DSL.

4.1 Language Requirements

Chapters 2 and 3 discuss the components and life cycle of SLAs, and the state-of-practice in logistics of SLAs, respectively. Using this information, we first look at which general parts of an SLA have to be modelled. We then define the requirements for an SLA DSL for automated monitoring for physical business services, by generalising from Chapter 3. These requirements are ordered using the MoSCoW prioritisation method.

4.1.1 SLA Parts

SLAs contain technical, organisational, and legal components, as shown in Section 2.3. Because this system is aimed at business services, some manual work is acceptable. Therefore, not all components of an SLA may need to be included in a DSL aimed at automated monitoring. Legal components are hard to enforce in an automated system, as the system does not have access to many aspects required to monitor this [10]. Therefore these components will not be modelled for monitoring. It could be possible instead to let legal components be added without having them be monitored.

Many organisational components describe aspects that are relevant in an SLA, but not directly relevant for monitoring, such as *liability*, and *level of escalation*, but also how monitoring results are reported to the involved parties, and how the SLA can be adjusted [10]. Other aspects are required and should therefore be modelled, most notably the involved parties, and the contract validity.

The most important components for the automated monitoring of SLAs are the technical components. These include service descriptions, their expected performance, and the metrics used to measure that performance. These compo-

nents combined describe what the system should be monitoring, and hence must be modelled.

Chapter 3 indicates that, from a monitoring perspective, the ability to accurately monitor the service provision is most valuable, as this is a time-consuming task. While not required for a monitoring system, there could be a benefit of being able to model legal and the discussed organisational components, which might further reduce the time required from a user. For example, being able to model other components related to the monitoring of the service provision, such as the consequences of breaking an agreement, would facilitate the definition of SLAs, as the information regarding an agreement is kept together.

4.1.2 Requirements Specification

Technical components are required for a monitoring system, as they describe what should be monitored. To be able to monitor at all, the system must be able to communicate with services to obtain the relevant metrics. Therefore, the DSL must define some way to perform this communication, for instance by describing interfaces, inputs, and outputs. As a technical description of services is not easy to work with, the DSL should take into account the type of users and make sure it is usable by them. Besides describing how to communicate with a service, it should clarify what exactly is being agreed upon. For the scope of this thesis, a textual description should be enough for this.

The monitoring process should also be able to define the actual agreements that will be monitored. In Chapter 3, we showed that in the logistics sector, for example, the timing of certain actions needs to be represented. Assuming that this is generally necessary in other domains, the DSL must be able to reason about time, e.g., by using some form of temporal logic. In addition, there must also be a way to describe obligations and permissions, e.g., by using some form of deontic logic. This would allow the DSL to reason about actions that must, might or may not be taken. Together, this should allow components of the monitoring system to reason about most types of agreements. Another observation made, is about frame SLAs, i.e. long term SLAs that offer the ability to concurrently have long-term and short-term agreements. While this could be a valuable extension to an SLA monitoring system, this is considered to be out of scope for the purposes of this thesis, to simplify the system.

Some organisational components of an SLA must also be modelled, more specifically, the involved parties and contract validity that are required for the monitoring system. Other organisational components are not required for monitoring, but could still be present in the model. The reporting of monitoring results, for example, is not relevant for the monitoring process, but modelling it in the DSL could facilitate SLA construction. Components such as liability and level of escalation, however, could also easily be described in another language or outside of the system. Legal components are similar, as they are also not required for SLA monitoring. In order to keep the scope of the project manageable, these have not been included in the DSL.

Finally, an SLA should contain contact information for the involved parties and their actors. Additionally, the role of each party should be defined, both for clarification, as well as reusability.

The requirements for the DSL are systematically described below:

Must have

DSL Requirement 1. The SLA specification language must be able to allow the description of how to communicate with services, e.g., their interfaces, inputs, and outputs.

DSL Requirement 2. The SLA specification language must be able to allow the description of agreements about a service.

DSL Requirement 3. The SLA specification language must be able to allow the description of temporal conditions in its agreements.

DSL Requirement 4. The SLA specification language must be able to allow the description of obligations and permissions in its agreements.

DSL Requirement 5. The SLA specification language must be able to allow the description of required organisational components, such as the involved parties, and the contract validity.

Should have

DSL Requirement 6. The SLA specification language should not require technical knowledge of the service to use, with the exception of specifying the interface.

DSL Requirement 7. The SLA specification language should be able to allow the description of how to communicate the monitoring results.

DSL Requirement 8. The SLA specification language should be able to allow the definition of actors.

DSL Requirement 9. The SLA specification language should be able to allow the reuse of the contact information of actors.

DSL Requirement 10. The SLA specification language should be able to allow the description of the role of actors in an SLA.

Could have

DSL Requirement 11. The SLA specification language could allow the import of a standardised technical service description to facilitate DSL Requirement 1.

DSL Requirement 12. The SLA specification language could allow the description of organisational components not needed for monitoring, such as *liability*, or *level of escalation*.

Won't have

DSL Requirement 13. The SLA specification language will not support the description of complex SLA structures, such as multi-tiered SLAs.

DSL Requirement 14. The SLA specification language will not support the modelling of legal components.

4.2 Related Work

Several techniques for the modelling of SLAs can be found in literature, the most prominent are: CC-Pi [22], SLAng [23], SLA* [24][25], WSLA [26], WS-Agreement [27], and using a model-driven Domain Specific Language (DSL) such as in Oberortner *et al.* [28].

WS-Agreement is a widespread Web Services protocol for establishing agreements between two parties, such as between a service provider and consumer. It uses an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties [27], [29]. It provides a high-level account of SLA and SLA template content, but it lacks formal semantics and does not allow the specification of fine-grained content [24]. Missing features can be added by extending the WS-Agreement, which is required for it to become a fully-fledged language [29].

WSLA is a framework for specifying and monitoring SLAs for Web Services. It consists of a flexible and extensible language based on XML Schema and a runtime architecture comprising several SLA monitoring services, which may be outsourced to third parties to ensure a maximum of objectivity [26]. However, WSLA is considered as overly constrained and it lacks flexibility [24].

Both WS-Agreement and WSLA are built for web-service scenarios, while the SLAs in this thesis we aim to support business services for logistics operations. The web services SLAs focus heavily on server performance, resource allocation, network traffic, etc, while our business services aim to support sea container logistics and the physical and digital processes around it. Both WS-Agreement and WSLA are flexible enough, however, that they could be extended for these situations. In addition, WS-Agreement has the problem that particular WS-Agreement notations, created by extensions, can be noninteroperable with other WS-Agreement notations [29].

SLAng is a language for defining SLAs supporting the needs of end-to-end quality of service (QoS) [23]. SLAng is specified in OMG's MetaObject Facility (MOF) and has a greater degree of language-independence with mappings to both XML and HUTN. It also places greater emphasis on semantics, providing formal notions of SLA compatibility, monitorability, and constrained service behaviour. A downside, however, is that it targets web services and provides only a limited set of domain-specific QoS constraints [24].

CC-Pi is a simple model of contracts for QoS and SLAs that also offers mechanisms for resource allocation and for joining different SLA requirements, combining two basic programming paradigms: name-passing calculi and concurrent constraint programming [22]. It is more generic than the options above, offering a theoretical framework for mapping SLAs to service constraints. CC-Pi is tightly coupled to the mechanics of negotiation, but does not address certain common constructs such as actor details or service interfaces [24].

SLA* is a domain-independent syntax for machine-readable SLAs and SLA templates. It is designed to be independent of underlying technologies, is decoupled from particular notions of service and modes of expression, and can be extended without sacrificing formality or semantics [24]. It does have the downside of not being easily human-understandable [29].

Finally, there is the option of defining a DSL for this purpose using metamodelling such as the one described by Oberortner *et al.* [28]. This approach has the benefit of yielding a DSL that is purpose-built for the project. A downside of

this approach is that it requires a metamodel of an SLA to be used as abstract syntax. To solve this, the model could use the previous options as inspiration for the metamodel, adjusted for the requirements and scope of this thesis.

4.3 Tools

Two popular options exist for Model-Driven DSL development: Eclipse Modelling Framework (EMF), and JetBrains Meta Programming System (MPS) [30], [31]. This section provides a brief comparison between the two, and a discussion about which solution we decided to use in our project.

4.3.1 Eclipse Modelling Framework

EMF is an open source framework using the Eclipse IDE and its extensive plugin system. To develop a DSL with it, a combination of those plugins should be used. A common option is to use Xtext for DSL construction, with OCL to specify constraints and ATL or QVT for transformations [32]–[35]. EMF with Xtext has been around for a while and is widely used in academia and industry.

Using the above combination of plugins results in a traditional tool chain: using a parser to transform text into an abstract syntax tree (AST) using a defined grammar, checking constraints, and transforming the AST into the desired output format.

4.3.2 JetBrains Meta Programming System

MPS is an open source IDE for language development, with most features built in, supported by a plugin system. The main benefit of MPS is the projectional editor: your language directly manipulates the AST instead of having to be parsed first. This works by showing the user a projection of the AST, and limiting input to valid values only. It also supports composable languages: languages can be easily extended and combined within MPS.

4.3.3 Comparison

There are some important similarities between EMF with Xtext and MPS: both are open source and both provide an IDE for the created DSL. The main benefits of the EMF approach are:

Maturity This system has been around much longer, and with that comes better support and available resources.

Traditional approach Uses a traditional approach to Model-Driven DSL design, again leading to better support and available resources.

Visual model design Model can be designed visually using plugins.

Plugins The whole process can be customised by choosing plugins that fit the user.

The main benefits of MPS are:

Projectional editing Both the design of the DSL and the resulting DSL use projectional editing, i.e., directly manipulating an AST through its projection. This removes the ambiguity of languages, as nodes are chosen explicitly. This creates the option for more complex languages.

No grammar and parser Projectional editing also removes the need for a grammar and parser, simplifying that part of the language design.

Extendability and composability A DSL made with MPS is intrinsically extensible and composable, without having to design specifically for that.

Language evolution A language version is stored with with every file, allowing the system to automatically migrate old files to a new version of the language. Migrations are also automatically generated.

Everything in one system Instead of customising the entire process, a complete package for DSL design is provided, allowing for a streamlined and stable experience, optionally somewhat customisable through limited plug-ins.

Especially the benefits of projectional editing, extendability, and language evolution have driven the choice for the MPS system. The projectional editing both simplifies the design process and allows for easier use of non-textual language elements. Intrinsic extendability makes it easier to build a generic language that can be extended for specific use cases. Finally, the built-in language evolution benefits provide a way to easily update older files to support new functionality, which can be beneficial in business use cases.

4.4 Design Approach

To design a model-based DSL to specify SLAs, a metamodel (or structure, as MPS calls it) was required first. This was done by starting with the abstract syntax of SLA* and modifying it for our purpose. It was then extended to support deontic and temporal logic, in accordance with DSL Requirements 3 and 4. Care was taken to separate technical and business levels of the language, to keep it usable for non-technical users. The structure was updated as required in iterations of the language.

After defining the structure, several other elements of the language had to be implemented: an editor, constraint, and a type system. First, a minimal viable editor was made, making sure a simple syntax exists for every concept. Second, a minimal constraint set and type system were implemented, which determined the constraints and type restrictions on the language. These components were updated when needed in iterations of the language.

Finally, the language was tested by using it to define different SLAs, based on the running example of Section 3.1, verifying that it covers the requirements.

Chapter 5

SLA Specification Language

This chapter describes the SLA specification language and the design choices made while defining it. The chapter first provides a top-level overview of the language structure, and describes the overall design choices. We also describe three important aspects of the tooling used to create the DSL: the editor, type system, and constraints. Then, all parts of the DSL are described in more detail, with an overview of their metamodel and an example. Finally, the generation of the artefacts that are used by the monitoring system is discussed.

5.1 Structure/metamodel

This section briefly explains the structure of our language and then describes the choice of starting point for the DSL, followed by a discussion of the problems encountered with the chosen approach. Finally, an overview of the DSL structure and design choices is provided.

With MPS, the language design uses aspects, such as the structure, editor, and type system. The structure aspect of a DSL is equivalent to the metamodel of a DSL in Model Driven Engineering, since it describes the concepts and relations supported by the language. The others define the concrete syntax and type system of a language, and are discussed in Section 5.2. We used MPS to develop our language.

Two main options exist for structuring a DSL: starting from scratch or using an existing model as a starting point. The second option is preferred as it prevents redoing existing work. Building on an existing model also decreases the chance of creating an unsound or incomplete model. The model chosen to built upon was discussed previously in Section 4.2, namely SLA*'s abstract syntax. This model features a well defined and seemingly complete abstract syntax for SLAs, especially when compared to the other options.

Some problems were found with SLA*'s abstract syntax with respect to our research. This model was not designed with projectional editing in mind, and thus uses unique names for referencing instead of using direct references that are supported by and preferred in MPS. It also lacks some clarity in the explanations of parts of the abstract syntax, and does not offer clear insight into its design decisions. Finally, some parts of the syntax might be more abstract than required for the DSL structure in MPS and the scope of this project.

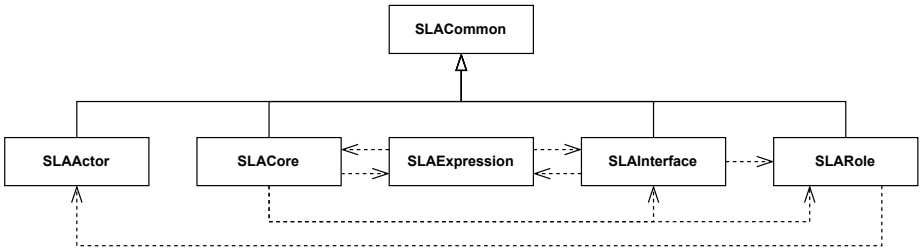

Figure 5.1: Top-level overview of the languages in the DSL, showing usage (dotted arrows) and extension (solid arrows).

Because of these problems we could not use the model in its original form, but adapted it to be used in the structure of the DSL. First, the core of the model was implemented in MPS as faithfully as possible with respect to SLA*. Second, the model was adapted to use direct references. Finally, to separate concerns, the language was split up into separate sub-languages, leveraging the intrinsic composition and extension properties of MPS:

SLACommon Common concepts and interfaces used in the other languages.
SLACore The main body of the SLA, describes the SLA itself, its sections, individual agreements, and roles in the SLA.
SLAActor Actors, either individuals or organisations, and information related to them.
SLARole Coupling of actors to the roles they have in an SLA.
SLAInterface Describes the services which the SLA is about, and the interfaces used to communicate with them.
SLAExpr A typed expression language used to define agreements. It uses the SLAInterface language to express which states and actions should or should not occur.

Figure 5.1 shows a top-level overview of the language in terms of the sub-languages, and the usage and extension relations between them.

5.2 Tooling

This section describes the tooling used to make the DSL. Specifically, it describes three aspects of MPS and how they were applied: the editor, type system, and constraints.

5.2.1 Editor

In MPS, the language editor is the mechanism to facilitate the generation of ASTs and enables the use of compact representations. They define the concrete syntax of a language, also called a projection in MPS. The purpose of this is to give the user an easier way to use the language than directly manipulating the AST. Figure 5.2 shows the difference in concrete syntax between an example of SLAActor with a simple editor and the editor. The latter of which is close to directly manipulating the AST.

```
Jane Doe
  Expensive consultant for Logistics Company

  Phone numbers :
    Mobile : (+31)678912345
  Email addresses :
    Main : j.doe@consultancy-bureau.com
  Addresses :
    Consultancy Bureau :
      Example street 12
      9999 YZ, Test City
      The Netherlands
```

(a) SLAActor example with editor

```
individual Jane Doe {

  contact information :
    contact info {

      phone number :
        phone number Mobile {
          phone number : (+31)678912345

        }
      email address :
        email address Main {
          email address : j.doe@consultancy-bureau.com

        }
      addresses :
        address Consultancy Bureau {

          address lines :
            address line {
              text : Example street 12

            }
            address line {
              text : 9999 YZ, Test City

            }
            address line {
              text : The Netherlands

            }
        }
    }
  description :
    paragraph {
      text : Expensive consultant for Logistics Company

    }
}
```

(b) SLAActor example with default editor

Figure 5.2: The difference between the default editor, similar to direct AST manipulation, and a simple editor

This paper uses a minimal editor configuration with the intent of showing a workable proof-of-concept. It is based on a simple form concept, showing the fields that need to be filled in and presenting everything in a somewhat compact manner.

Another benefit of MPS is the ability to define multiple editors for the AST. In the case of our DSL, it allows us to define a technical and textual representation of the DSL. The technical representation uses a more mathematical or programming style of displaying the DSL, while the textual representation is more oriented towards business users. The textual representation could also hide information that is not relevant for a business user. A simple example of the difference can be seen in Figure 5.3.

By using the editor aspect of MPS as described, DSL Requirement 6 is fulfilled.

5.2.2 Type system

To work with the typed expressions of SLAExpr, type restrictions should be in place. In MPS, this is done by configuring a type system. The type system we

```
test :
description
variables:
  var one : number = 42
  var two : number = 10
  var three : boolean = false
  var test = one equals two
  var duration : duration = P30D
guarantees:
  one > two → ℱ        three
                duration
```

(a) Technical representation

```
test :
description
variables:
  let number one be 42
  let number two be 10
  let boolean three be false
  let test be one equals two
  let duration duration be P30D
guarantees:
  if one greater than two then finally in duration three
```

(b) Textual representation

Figure 5.3: Same part of an AST shown with different concrete syntaxes

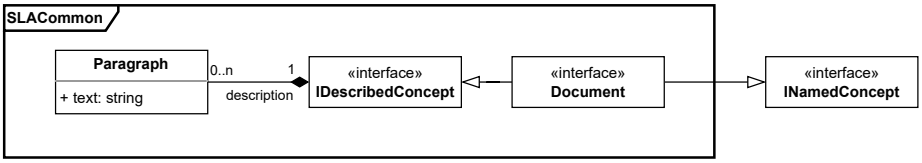

Figure 5.4: Metamodel of SLACommon, with concepts outside of the boundaries being part of MPS

created uses the types defined in the structure for two things: typing variables and type checking of expressions. The variables are both used in the expressions as well as the inputs of outputs of interfaces.

5.2.3 Constraints

By default, MPS does not restrict scoping for its languages, which means that any applicable concept instance can be referred to from everywhere. To fix this, and some minor other issues, constraints can be used. MPS constraints can either be on properties, i.e., to validate a property on a concept instance, or on references. The latter is used to constrain the scope that can be referenced. It is used on the SLA language, for instance, to prevent linking to roles defined in another SLA, and to set proper scoping for variable references.

5.3 SLACommon

SLACommon is used to define common concepts, data types, and interfaces that are used in the other languages. Its purpose is to prevent duplication of concepts and unnecessary dependencies, as well as having a single place to define new data types. Figure 5.4 shows the structure of SLACommon.

The most important concepts of the language are the interfaces *IDescribed-Concept* and *Document*. The former adds a description to a concept in a similar way that MPS's interface *INamedConcept* add a name to a concept. The latter is an interface used to indicate that a document can be generated from that concept. Part of SLACommon was based on SLA*, but it grew during the DSL development as more shared elements were required.

While this language does not specifically fulfil any requirements, it supports the other languages in doing so.

5.4 SLACore

SLACore is the core of our DSL, describing an SLA, its structure, its individual agreements, and its roles. Figure 5.5 shows the structure of SLACore. This is mostly directly based on SLA* adapted to introduce direct references where possible, and simplified to both reduce the complexity and limit the scope of the language. A new addition is the option to group agreements in concerns, allowing more flexibility when describing an SLA.

The most important concepts of SLACore are *SLA* and *Agreement*. *SLA* is a root concept, i.e., the root node of an AST that contains the basic SLA information. It is the core of the DSL, and brings everything else together. *Agreement* defines the guarantees of the SLA, including responsible parties. An example of SLACore is shown in Figure 5.6.

This language fulfils parts of DSL Requirements 2 and 5, with the other parts being fulfilled by the other sub-languages. With further extension it could also fulfil DSL Requirement 12.

5.5 SLAActor

SLAActor describes the actors of an SLA, which can be either individuals or organisations. It is based on similar concepts in SLA*, but altered to fit the language better. Some terms were also changed to better convey the intended meaning: actor was chosen over party, and operative was replaced by individual to differentiate it from organisations. It also allows basic contact information of the actors to be represented, as this is often present in SLAs.

There are four important concepts in SLAActor: *Actor*, *Individual*, *Organisation*, and *ActorReference*. *Actor* is an abstract concept that represents any actor in the SLA, and it is extended by both *Organisation* and *Individual*. *Individual* represents a single person, either as a standalone definition, or as part of an organisation. *Organisation* represents an organisation, which can include any number of individuals. Finally, *ActorReference* is used to refer to any defined *Actor*.

Figure 5.7 shows the structure of SLAActor, and Figure 5.8 shows and example of the definition of an organisation.

This language fulfils parts of DSL Requirements 5, 8 and 9, with other sub-languages fulfilling the other parts.

5.6 SLARole

SLARole describes the function of actors within an SLA. This concept was present in a simpler form in SLA*, but it is used here to be able to decouple an actor from the role it takes within an SLA. This also makes it easier to create SLA templates, as it is possible, for instance, to create a provider and client role, and add the corresponding parties to these roles later.

Figure 5.9 shows the structure of SLARole. All concepts shown here are equally important: *Role* defines the role itself, and can optionally reference actors that fulfil its role, *RoleList* is a list of defined roles, which could, for example, be used to define default roles, *RoleAssignment* is a way to assign actors to an already defined role in an SLA, and *RoleReference* is used to reference a role,

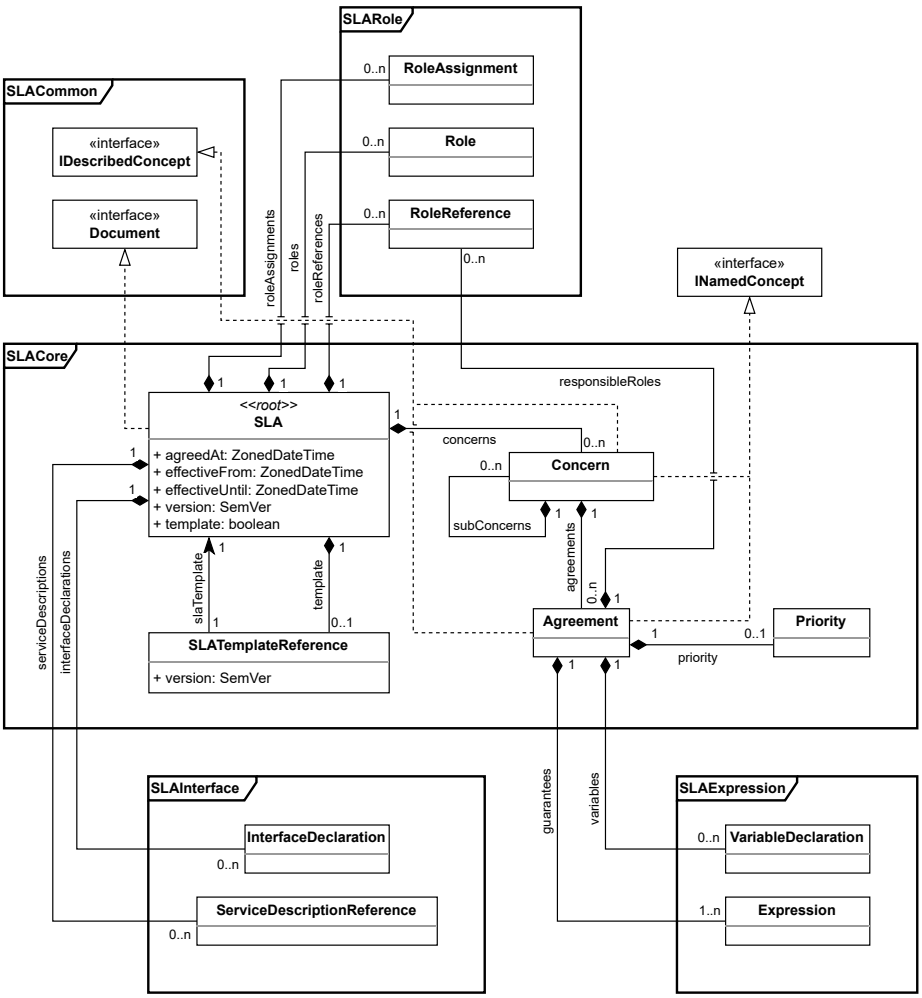

Figure 5.5: Metamodel of SLACore, including dependencies on other languages and MPS

```
SLA Running Example Scenario
  template  : false
  version   : 0.1.0
  from      : 2023-04-01
  until     : 2023-10-01
  signed on : Date on which the SLA was signed

  Description :
    Example SLA for running example

  Roles :
    Provider
    Planning contact
    Terminal contact
    BI contact
    Customer
    Add to role Customer :
      Example Company
    << ... >>

  Service Descriptions :
    LC Transport

  Interfaces :
    << ... >>

Import :
  Agreements for import orders
  agreements ...
  Transport :
    Agreements for import transport
    agreements ...
    sub concerns ...
```

Figure 5.6: Example of the concrete syntax of SLACore

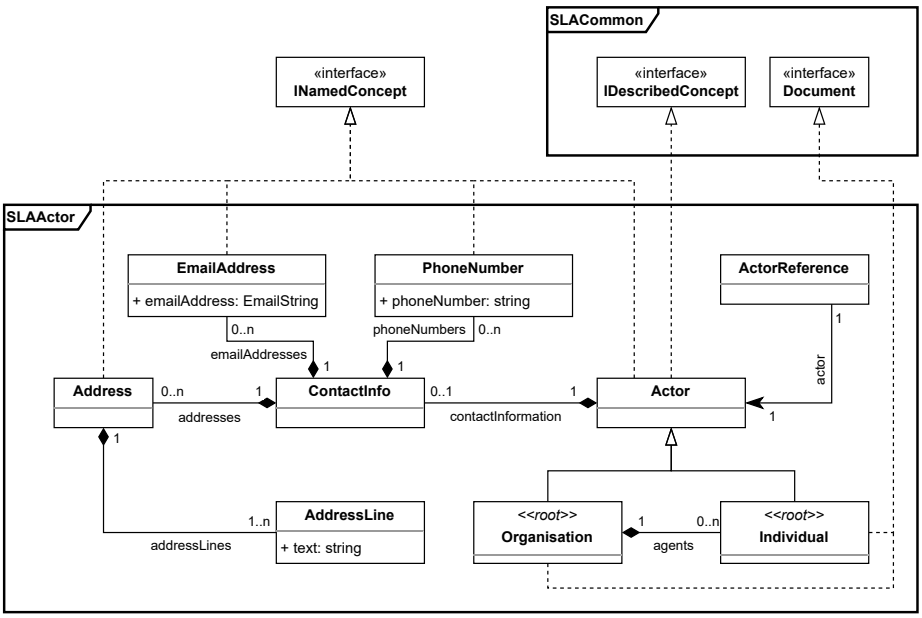

Figure 5.7: Metamodel of SLAActor, including dependencies on other languages and MPS

```
Organisation Logistics Company

Description :
  An example logistics company, providing transport
  of containers from a seaport to a customer.

Contact information :
  Phone numbers :
    Primary   : (+31)012345678
    Secondary : (+31)023456789
  Email addresses :
    Info : info@logistics-company.com
  Addresses :
    Office :
      Some street 42A
      1234 AB, Example City
      The Netherlands
    Terminal :
      Another street 4
      1243 BA, Example City
      The Netherlands

People :
  Jan de Vries
    Planning Coordinator

    Phone numbers :
      Office : (+31)034567891
      Mobile : (+31)645678912
    Email addresses :
      Main : jan.devries@logistics-company.com
    Addresses :
      << ... >>
```

Figure 5.8: Example of the concrete syntax of SLAActor

for example, when assigning the responsibility for an agreement in the SLA. Figure 5.10 shows an example of the concrete syntax of a *RoleList*.

This language fulfils DSL Requirement 10, and could support the fulfilment of DSL Requirement 7 by specifying which roles to send reports to.

5.7 SLAInterface

SLAInterface is used to describe the services which the SLAs are about, and how to interact with them. Its concepts are mostly directly based on SLA*, with adaptations similar to SLACore. The language also allows a textual description to be defined for the services, which was a requirement. Figure 5.11 shows the structure of SLAInterface.

SLAInterface has five main concepts: *InterfaceDeclaration*, *Endpoint*, *Interface*, *Operation*, and *ServiceDescription*. An *InterfaceDeclaration* is the central concept of the language, containing both a number of *Endpoint*s and an *Interface*. Every *Endpoint* declares a location and protocol used to interface with the service. The *Interface* defines a set of *Operation*s, each of which refers to an *Endpoint*, and defines which inputs can be supplied and which outputs are expected. Finally, the *ServiceDescription* is a list of *InterfaceDeclaration*s that can be defined as a standalone document, allowing easy reuse of the defined *InterfaceDeclaration*s.

An example of SLAInterface is shown in Figure 5.12.

This language fulfils DSL Requirement 1. With further extension it could also partially fulfil DSL Requirement 11.

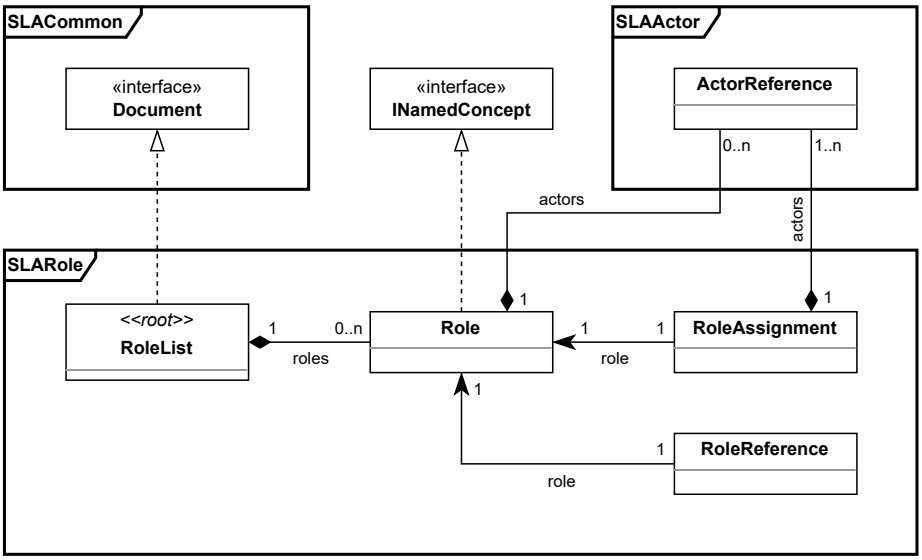

Figure 5.9: Metamodel of SLARole, including dependencies on other languages and MPS

```
Roles Default Roles :
  Provider : Logistics Company
  Planning contact : Jan de Vries
  Terminal contact : John Doe
  BI contact : Jane Doe
  Customer : << ... >>
```

Figure 5.10: Example of the concrete syntax of SLARole's *RoleList*

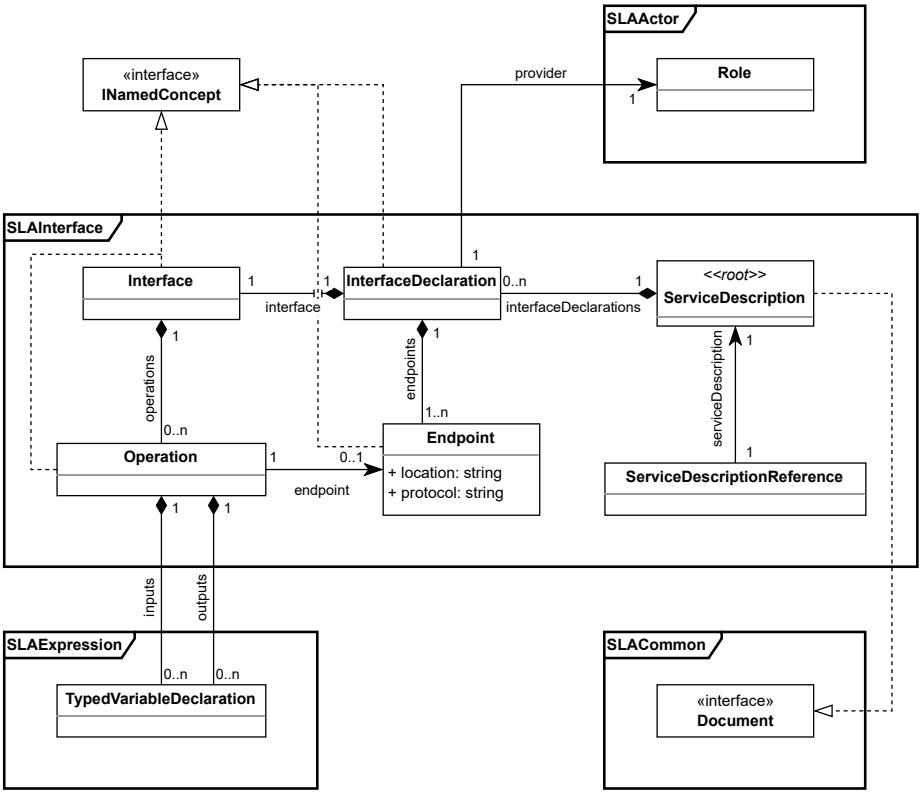

Figure 5.11: Metamodel of SLAInterface, including dependencies on other languages and MPS

```
LC Transport service description
  Order information
    provider  : Provider
    endpoints : Import order info ( HTTP ) : https://app.logistics-company.example/orderInfoImport
                Task info ( HTTP ) : https://app.logistics-company.example/taskInfo
    interface : Transport information on Import order info
                  inputs  : var orderNumber : maybe < number > = <no value>
                  outputs : var portPickup : maybe < date-time > = <no value>
                            var terminalBargeDropoff : maybe < date-time > = <no value>
                            var terminalTruckPickup : maybe < date-time > = <no value>
                            var customerArrival : maybe < date-time > = <no value>
                            var customerDeparture : maybe < date-time > = <no value>
                            var terminalTruckDropoff : maybe < date-time > = <no value>
                            var terminalBargePickup : maybe < date-time > = <no value>
                            var depotDropoff : maybe < date-time > = <no value>
                Task information on Task info
                  inputs  : var orderNumber : maybe < number > = <no value>
                  outputs : var weight : maybe < number > = <no value>
```

Figure 5.12: Example of SLAInterface

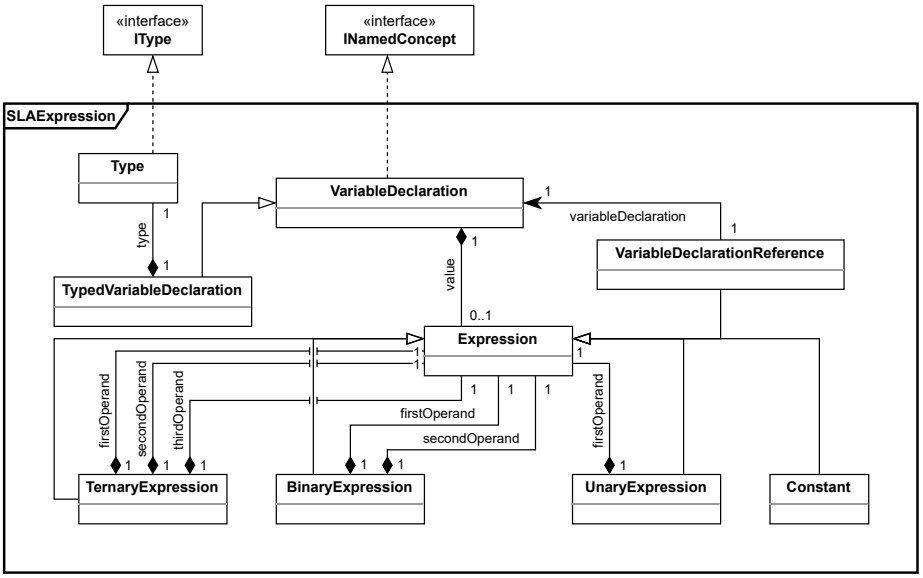

Figure 5.13: Metamodel of SLAExpr, including dependencies on other languages and MPS

5.8 SLAExpr

SLAExpr is a typed expression language defined to logically describe the agreements of the SLA. We started defining this language based on SLA*, but changed to be more aligned with logic expressions. This choice was made because of the projectional editing capabilities of MPS. Because of this, both a technical, mathematical projection, as well as a descriptive, textual projection can now be used to edit the agreements. This also allows the language to be based on mathematical expressions, which is beneficial for the monitoring system.

The language itself is based around three main groups of concepts: types, variables, and expressions, corresponding to the three concepts, *Type*, *VariableDeclaration*, and *Expression*, respectively.

The types are used for the type system discussed in Section 5.2.2. A type concept exists for every available type in the language, and they all extend *Type*.

Variables are used as a way to store an expression result as a shorthand in the agreements, as well for the inputs and outputs of SLAInterface's *Operation*s. Their types can be both implicit using type inference, or explicit using *TypedVariableDeclation*s.

Expressions are used to define constraints for the agreements. They are grouped by the number of operands in the structure, as shown in Figure 5.13. To be able to properly express the constraints as required, operators from temporal logic and deontic logic are used to reason about time, permission, and obligation.

Figure 5.13 shows its structure, and Figure 5.14 shows an example of a simple agreement using SLAExpr.

This language fulfils DSL Requirements 3 and 4, and DSL Requirement 2 together with SLACore.

```
Tasks :
  Agreements for import tasks
  Weighing :
  All containers must be weighed on the terminal
  variables:
    var was weighed : boolean = weight is present
    var has left terminal : boolean = terminalTruckPickup is present
  guarantees:
    has left terminal → was weighed
```

Figure 5.14: Example of SLAExpr

```
SLA generation_example
  template  : false
  version   : 0.1.3
  from      : Date from which the SLA is in effect
  until     : Date until which the SLA is in effect
  signed on : Date on which the SLA was signed

concern_example :
  This is a description
  agreement_example :
  << ... >>
  variables:
    let test be false
  guarantees:
    if test then 5 greater than or equals 4.995
```

Figure 5.15: Simple SLA example

5.9 Code Generation

To be able to use the specified SLA in the monitoring system, some form of export of a monitoring specification is required. As XML is well suited to tree structures and has convenient support for both generation and parsing, we chose to use XML for this.

The easiest way to generate the monitoring specification with MPS is to transform the model to a Java-based generator using native support from MPS. The Java generator then uses a external library to generate the XML files. A simple example of the input and output of the generator is shown in Figure 5.15 and Listing 5.1.

Most of the SLA is already in a tree structure or can be easily converted. Two main problems existed for the generation, related to names and references. The former problem is because there is no restriction of the uniqueness of names in the SLA specification. This can be solved, however, by generating a unique name for each node, and storing both that and the actual name in the XML document. The latter problem is how to reference from one branch of the SLA specification tree to another, such as with variables declaration. This could also be solved by generating unique identifiers for the monitoring system, and using the MPS referencing system to ensure all identifiers point to the right node of the tree.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sla name="generation_example" version="0.1.3">
    <concern name="concern_example">
        <agreement name="agreement_example">
            <guarantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            ↪ xsi:type="binaryExpression" type="ImplicationExpression">
                <firstOperand xsi:type="variableReference" reference="test1"
                ↪ type="VariableReference"/>
                <secondOperand xsi:type="binaryExpression"
                ↪ type="GreaterThanOrEqualExpression">
                    <firstOperand xsi:type="constant" type="IntegerConstant" value="5"/>
                    <secondOperand xsi:type="constant" type="FloatConstant" value="4.995"/>
                </secondOperand>
            </guarantee>
            <variable identifier="test1" name="test" type="any">
                <value xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                ↪ xsi:type="constant" type="BooleanConstant" value="false"/>
            </variable>
        </agreement>
    </concern>
</sla>
```

Listing 5.1: Generator output for Figure 5.15

Chapter 6

SLA Monitoring

This chapter describes the requirements for an SLA monitoring system, existing solutions, and an approach for this thesis' system. It first derives requirements for such a system from previous chapters and literature, then inspects related work and how they hold up to these requirements. Finally, an approach is given for designing a system based on the requirements and related work, as well as implementing a prototype for it.

6.1 Requirements

This section discusses the requirements that we have identified for an automated SLA monitoring system for logistics business services, and summarises the requirements in the MoSCoW prioritisation style.

We started with requirements from the interviews (see Chapter 3): the system should be able to give near real-time information, as well as reports over a period of time. The former aims to quickly provide users with information about the impact of their actions on the agreements of an SLA and should be able to detect violations right after they occur. This could both allow for the prevention of actual violations because of the delay between planning and physical events, and quick response to prevent further penalties in case of violations. The latter is a more traditional usage of SLA monitoring, often done through reports and dashboards.

Another point to be taken into account is the type of SLA specifications that are supported by the system. A specification language for SLAs has been defined in Chapter 5. The monitoring system must therefore be compatible with at least this monitoring specification.

Attention should also be paid to the maintainability of the system. This can partly be done through its architecture, by designing the system with modular components.

Related to this is the dependence between components. Having the monitoring configuration mostly decoupled from the SLAs would allow greater flexibility with the SLA modelling solution. It should also allow for easier extensibility, for instance, to integrate with another SLA specification later on.

Finally, the way monitoring results are accessible and explained should also be considered. Facilitating access and understanding of these results is a critical

requirement of the system, as these results should be understood by humans, not just machines.

Must have

Requirement 1. The SLA monitoring system must be able to perform periodic monitoring, e.g., by generating a monthly report.

Requirement 2. The SLA monitoring system must be able to perform event-based monitoring, e.g., by checking whether a new transport planning violates the SLA, and being able to respond to such events immediately.

Requirement 3. The SLA monitoring system must be able to use monitoring specifications derived from the SLA specification language defined in Chapter 5.

Should have

Requirement 4. The SLA monitoring system should be extendable.

Requirement 5. The SLA monitoring system should have a modular design to improve maintainability.

Requirement 6. The SLA monitoring system should be decoupled from the SLA specification.

Requirement 7. The SLA monitoring system should be able to generate reports that clearly show in a human-understandable manner which agreements where violated, and what the inputs for this agreement were.

Could have

Requirement 8. The SLA monitoring system could be able to provide detailed human-understandable reports, showing not just the agreements that were violated and its inputs, but also explaining what the violation is.

6.2 Related Work

Several techniques for monitoring SLAs can be found in literature, often focusing on a specific aspect of the process. We compared twelve different techniques using the criteria from Section 6.1, based on work by Müller *et al.* [29]. A summary of this analysis can be found in Table 6.1. We indicate the type of SLA specification supported by each technique and whether it is human understandable (H-U), whether information is coupled to the SLA specification, how the results can be accessed, how violations are explained and whether they are human understandable (H-U), whether violations are detected near real-time, and whether its components are modular. *N/A* indicates that no clear answer could be given.

The supported SLAs fall into two categories: domain-specific [36], [37], [39], [41] and general-purpose [26], [29], [38], [40], [42]–[47]. The former does not consider a general-purpose structure or notation, and might not be suited for realistic scenarios. Therefore, they are also poorly suited for use in this project. The latter does consider a general-purpose structure or notation, with many of

Table 6.1: Analysis of monitoring techniques in literature, based on Müller *et al.* *H-U* means human understandable, and *N/A* means no clear answer could be found.

	Supported SLAs	Monitor config.	Monitoring results	Explanation of violations	Real-time detection	Architecture elements
WSLA [26]	General-purpose, not H-U	Coupled	API	Partial, not H-U	Yes	Separated
Comuzzi *et al.* [36]	Domain-specific	Decoupled	API	None	Yes	Separated
Michlmayr *et al.* [37]	Domain-specific	Coupled	Query language	None	Yes	Separated
Raimondi *et al.* [38]	General-purpose, not H-U	Decoupled	Log	None	Yes	Single component
Sahai *et al.* [39]	Domain-specific	Coupled	Formal model	Partial, not H-U	Yes	Separated
Palacios *et al.* [40]	General-purpose, not H-U	Coupled	*N/A*	None	*N/A*	*N/A*
Di Penta *et al.* [41]	Domain-specific	Coupled	*N/A*	None	*N/A*	*N/A*
SLA@SOI [42], [43]	General-purpose, not H-U	Decoupled	API	Partial, not H-U	Yes	Separated
TRUSTCOM [44]	General-purpose, not H-U	Coupled	API	Partial, not H-U	Yes	Separated
Mahbub and Spanoudakis [45], [46]	General-purpose, not H-U	Coupled	*N/A*	Full, not H-U	Yes	Separated
Comuzzi and Spanoudakis [47]	General-purpose, not H-U	Decoupled	API	None	Yes	Separated
SALMonADA [29]	General-purpose, H-U	Decoupled	Formal document	Full, H-U	Yes	Separated

them supporting WS-Agreement [29], [40], [45]–[47], one using WSLA ([26]), and one using SLA* [42], [43]. None of these are easily understandable by humans, which SALMonADA [29] claims to be.

Two categories were also identified for the monitor configuration: coupled [26], [37], [39]–[41], [44]–[46] and uncoupled [29], [36], [38], [42], [43], [47]. All techniques automatically configure the monitoring component, so the difference here is in the coupling between the configuration and the SLAs. When they are coupled, any changes to the SLA specification could result in required changes to the monitor. In a decoupled system, the SLA is translated into another document that is used to configure the monitor.

For accessing the monitoring results, five different methods were identified, with some approaches not describing any [40], [41], [45], [46]. The most common approach is through an API [26], [36], [42]–[44], [47], which has the downside of not being standardised, so that it is harder to be modified over time. The same goes for using a log file [38]. A possibly more flexible solution is the use of a query language [37], but this requires the system to be able to deal with this language. Other proposed options are a formal model [39] or a formal document [29].

To explain the accessed results, most techniques offer no solution [36]–[38], [40], [41], [47]. Others either explain violations only partially [26], [39], [42]–[44], or fully [45], [46], but not in a human-understandable fashion, with one exception, namely SALMonADA [29], which claims to fully explain the violations in a human-understandable format.

Most techniques showed how to use them for near real-time detection of violations, with some [40], [41] being unclear.

Finally, most of the techniques included an architecture, except for two [40], [41]. One of the others [38] included an architecture where monitoring and

analysis were conducted in one component, with the rest separating these tasks [26], [29], [36], [37], [39], [42]–[47].

Two solutions for SLA monitoring in Table 6.1 stand out: SLA@SOI and SALMonADA. The former provides the basis of the SLA modelling solution with SLA* (see Section 5.1), and supports most of the desired functionality, with only not being understandable by humans being a problem. Similarly to SLA* though, its scope is again much larger than needed. The latter solution also supports most of the desired functionality, including human-understandable violation explanations and a more limited scope. A downside, however, is the use of WS-Agreement as a model for SLAs, which was not one of the preferred specification techniques.

6.3 Approach

Three things were done to show what the proposed SLA monitoring system looks like. First, a high level design was made based on SALMonADA's design. The design has a similar architecture with similar components, but was adjusted to fit this thesis. Second, a prototype was implemented to verify the system. As the scope of this system is a proof of concept, only parts required for verification were implemented. Finally, the system was verified using test scenarios based on Section 3.1.

Chapter 7

Monitoring System Design

This chapter describes the design of the SLA monitoring system, and a prototype to test the core functionality. It first shows the general architecture and an overview of the core components, followed by the interaction designs for three major scenarios. Then, it describes the major components of the system. Finally, it describes the proof-of-concept prototype used for verification.

7.1 Architecture

We first describe a conceptual reference model of the system, showing a high-level overview of the relevant components and agents. The model is shown in Figure 7.1. The model also includes the SLA Specification System as discussed in Chapters 4 and 5 to show how they interact. The following components are presented:

Client The client is the user of the system. They want to use the system to obtain monitoring results.

SLA Builder The SLA Builder is used by the client to specify the monitoring requirements of an SLA. It produces a Monitoring Specification that can be registered in the SLA Monitoring System to perform monitoring according to the specification.

Monitor The Monitor is responsible for collecting the monitoring data, according to the Monitoring Specification.

Analyser The Analyser is responsible for analysing the collected data for compliance with the SLA and creates an Analysis Result, a machine-readable report of the analysis.

Explainer The Explainer is responsible for transforming the Analysis Result to a human-understandable result. It produces a report that can be interpreted by the client, showing the results of the analysis.

Figure 7.2 shows an architectural model of the SLA Monitoring System, containing the main components. Compared to the conceptual model, it adds the Orchestrator, which provides the external interfaces and manages the flow of the system, keeping the other components decoupled from each other. It also shows the interfaces provided by each component.

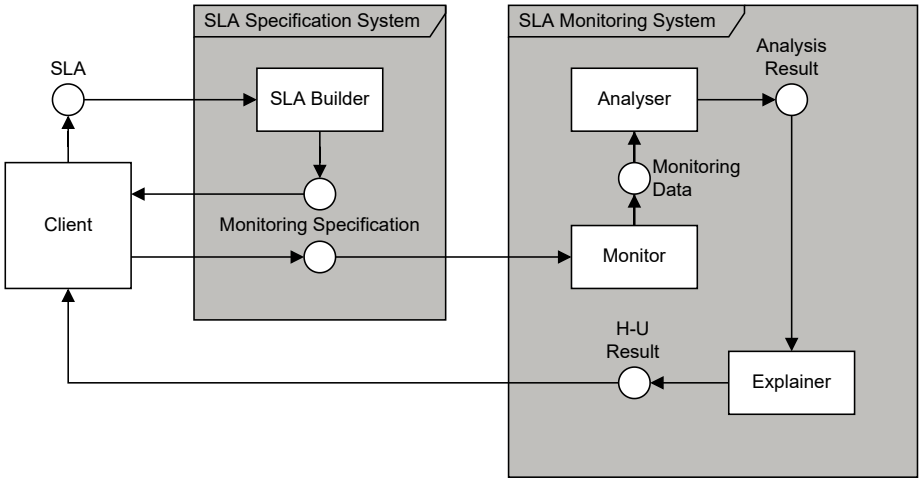

Figure 7.1: Conceptual reference model of the SLA monitoring system

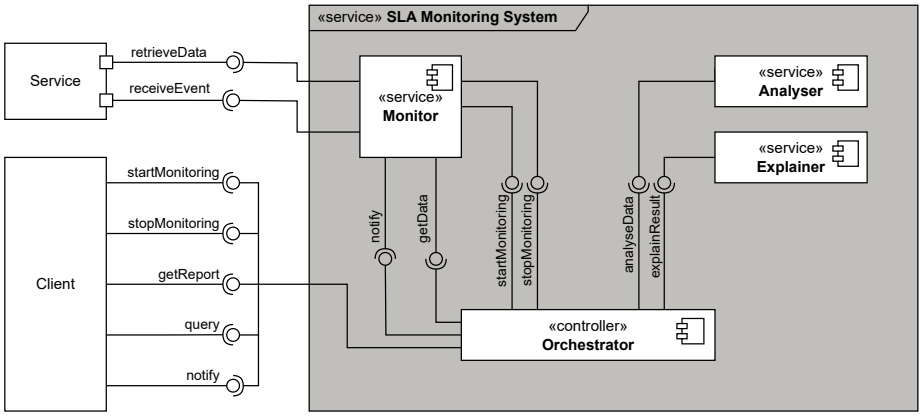

Figure 7.2: Architectural model of the SLA monitoring system

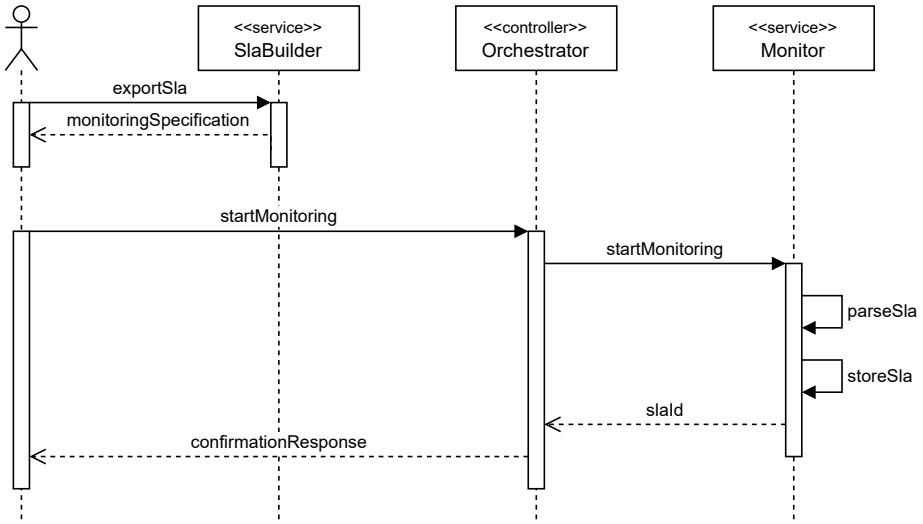

Figure 7.3: Sequence diagram for starting the monitoring of an SLA.

7.1.1 Interaction Design

Sequence diagrams were designed for three common scenarios of the monitoring system: starting the monitoring of a new SLA, requesting a monitoring report, and responding to a monitoring event.

Start Monitoring

For the first scenario, starting the monitoring of an SLA (see Figure 7.3), we assume that an SLA has been constructed using the DSL described earlier. The user first exports the monitoring specification and uploads it the the orchestrator component. The orchestrator then passes the specification to the monitor component, which start by parsing the monitoring specification and storing the result. This process also registers it as an active SLA if its validity is correct. A confirmation is then returned to the user.

Get Report

The second scenario, obtaining a report (see Figure 7.4), starts with a request from the user or client to the orchestrator. The orchestrator then tells the monitor to obtain the required data. The monitor fetches the SLA and determines which operations are required to obtain all the data. It then performs all the required operations and stores them.

The orchestrator then requests analysis by the analyser, which starts by fetching the monitoring data and SLA. It then uses its expression solver to check every agreement in the SLA. The result of this analysis is again stored and an identifier is returned to the orchestrator.

The orchestrator then tells the explainer to process the analysis result into a report. The explainer first fetches the analysis result and the SLA, and uses these to create a report. This report is stored and an identifier is returned to the orchestrator, which fetches the report and returns it to the user.

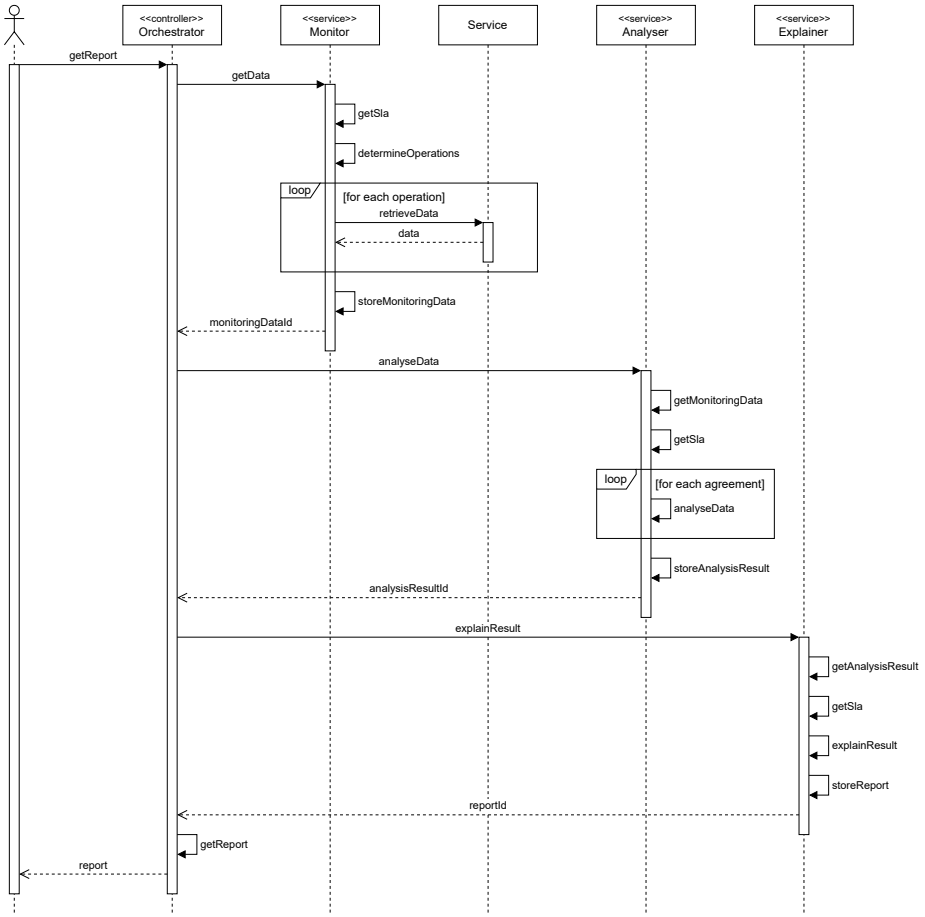

Figure 7.4: Sequence diagram for obtaining a report from the monitoring system.

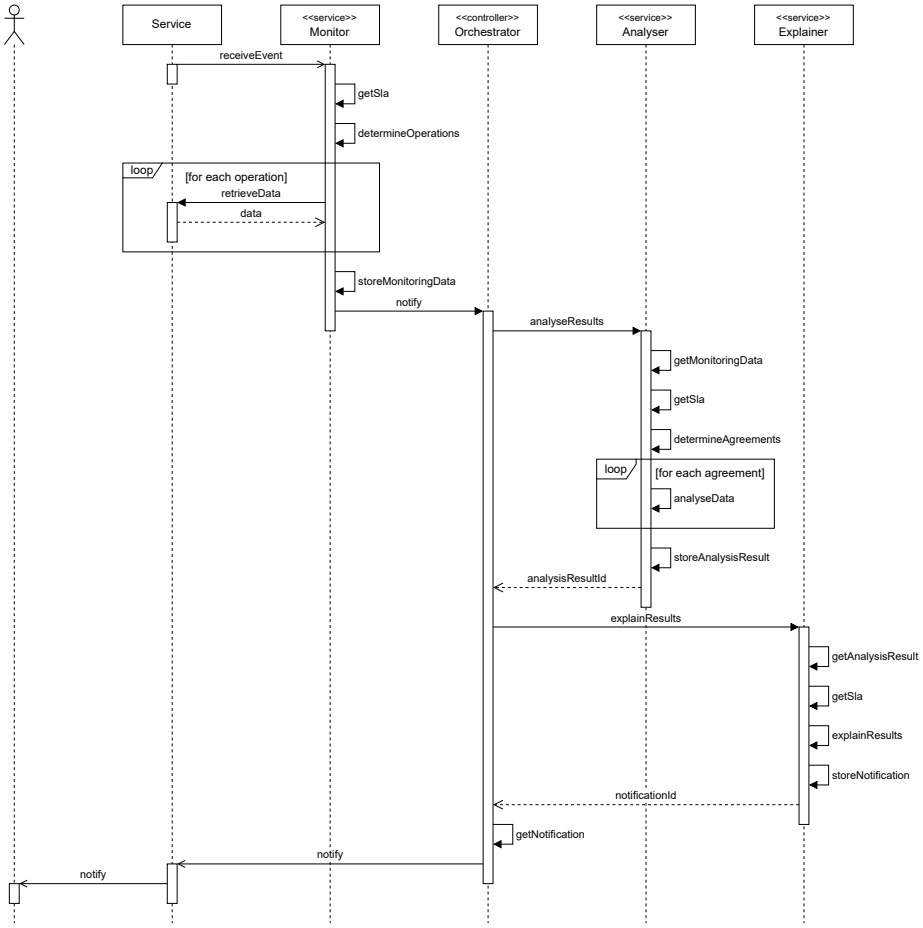

Figure 7.5: Sequence diagram of an event-based monitoring situation.

Monitoring Event

The third scenario, responding to a monitoring event (see Figure 7.5), starts with a situation where the service wants real-time decision support. To obtain this, it sends an event to the monitor, which checks it with the relevant SLA and determines which other information is required to perform the analysis. It then retrieves all the required data from the service and stores it along with the event, then notifies the orchestrator that analysis is requested.

The orchestrator then tells the analyser to check the results for violations. The only difference with the previous scenario is that it only checks the agreements that depend on the data indicated by the event. It again stores the result and returns an identifier to the orchestrator.

The orchestrator then calls the explainer to process the analysis results. The explainer also does this similarly to the previous scenario, but creates a summary of violations instead of a full report. The results are stored and passed to the orchestrator as before.

Finally, the orchestrator fetches the results and sends a notification to the

service, which in turn can inform the user if desired.

7.1.2 Domain

The system uses a shared domain to ensure all components use the same data structure. It consists of classes based on the SLA DSL's metamodel, with the addition of classes for intermediate data used by the system's components to communicate with each other.

7.2 Components

This section describes the major components of the system: the orchestrator, monitor, analyser, and explainer.

7.2.1 Orchestrator

The orchestrator component has two main responsibilities: it manages the execution flow of the system, and provides external interfaces. By having the all components communicate through the orchestrator, they can remain decoupled, and can be easily swapped with other components compatible with the orchestrator's interfaces. As the component also provides the external interfaces, all client communication is also through the orchestrator.

The orchestrator provides an interface for the client to start and stop monitoring. For the former, the client has to provide monitoring specification. The orchestrator then calls on the monitor service to actually perform the desired actions. The other interfaces provided are for obtaining a monitoring report and querying the monitoring system. These are also passed on to the monitor service. Finally, it can call an external notify interface, which can be used for event-based monitoring to notify an external system of a violation.

A design for this component is shown in Figure 7.6.

7.2.2 Monitor

The monitor component also has two main responsibilities: managing the monitoring specifications and retrieving monitoring data from external services. For the former, a sub component takes care of storing and retrieving the specifications, and the removal of expired ones. The latter is done by using interface details from the monitoring specification to call the correct external interfaces. It also looks up the correct calls to make for event-based monitoring, based on the monitoring specification and the requested details. This data is then transformed into an intermediate format more suitable for internal use. A design for this component is shown in Figure 7.7.

7.2.3 Analyser

The analyser has only one responsibility: to check the provided monitoring data for SLA violations. It does this by using an expression solver for the agreements in the SLA with the provided data. The result of the solver is provided to the orchestrator as a simple machine-readable report. The design of this component is similar to Figure 7.7 and therefore not shown separately.

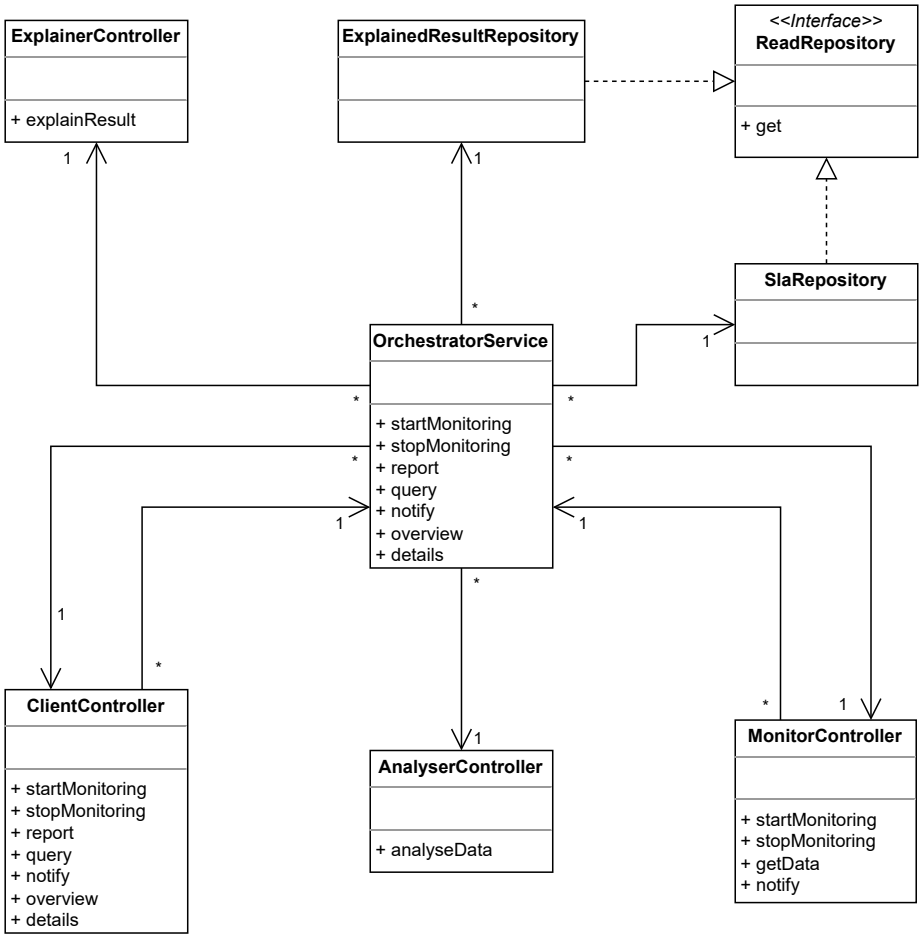

Figure 7.6: Conceptual class diagram of the Orchestrator component

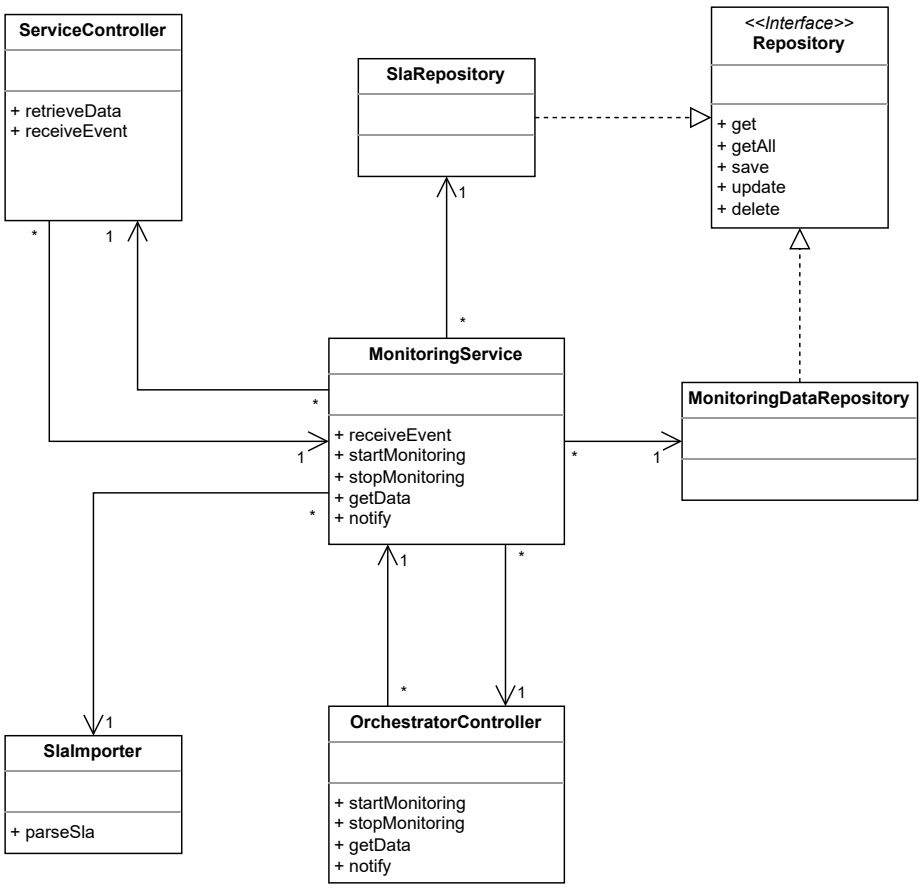

Figure 7.7: Conceptual class diagram of the Monitor component

7.2.4 Explainer

The explainer also has only one main responsibility: creating a human understandable report from the analysis result. To do this, it uses the original text used in the SLA and simple natural language structures to explain the violations. It can generate both full reports, as well as notifications for event-based monitoring. The design of this component is similar to Figure 7.7 and therefore not shown separately.

7.3 Prototype

To show the viability of the monitoring system of this thesis, a proof-of-concept prototype was made. The prototype reduces the scope of the system in the several ways.

First, the system focuses only on the functionality required to evaluate the test scenarios described in Chapter 8. This means that it only features partial support for several steps, such as the SLA builder, expression solver, and monitoring specification. It also only features a minimal user interaction, and the explainer component is quite limited.

Second, the prototype is not made to enterprise standards, and lacks the strict decoupling as described in Section 7.1. It is still loosely decoupled, and should be fine for a proof-of-concept application.

Finally, the external communication part of the monitor component is replaced with a function that reliably provides the test data required for the test scenarios.

Chapter 8

Verification

This chapter discusses how we validated the prototypes using testing. It discusses the test coverage, the testing setup including the scenarios and tests, and the fulfilment of the requirements.

8.1 Tests Coverage

To verify the DSL and the monitoring system prototype, they were tested in several ways. We identified the following functionality of the system that should be verified:

(1) Creating an SLA with the DSL
(2) Exporting a monitoring specification from the SLA builder
(3) Importing a monitoring specification into the monitoring system
(4) Generating a monitoring report, which consists of the following steps:
    (a) Retrieving data from the service
    (b) Analysing this data to find violations
    (c) Process the analysis results into a readable report
(5) Responding to a monitoring event, which consists of the following steps:
    (a) Determine what other data is required to check the requested data
    (b) Retrieve all required data from the service
    (c) Analyse the data to find violations
    (d) Process the analysis results into a readable notification

Items (1) to (3) together make up the start of monitoring an SLA, and their their sequence diagram is shown in Figure 7.3. Item (4) encompasses the steps required to generate a monitoring report, which can also be seen in Figure 7.4. Item (5) encompasses the steps required to respond to a monitoring event, which can also be seen in Figure 7.5. Because of the limitations of the prototype (see Section 7.3), the retrieval of data (Items (4)a and (5)b) is not supported and could not be tested. All parts besides Items (4)a and (5)b have been tested, and together should verify the functionality of the system.

8.2 Test Setup

Three types of tests were performed to verify the system: scenario tests, integration tests, and unit tests. This section first describes the scenarios used, and then for the points above describes which tests were performed to verify each of them.

8.2.1 Scenarios

Scenarios were used in two ways during the testing process, either as functional structures to test all aspects of a component, or as reflection of a real-world SLA monitoring situation. Scenarios of the first type were created for all of the sub-languages of the SLA DSL. They were not based on actual SLAs and focused purely on testing every aspect of the language. Two scenarios of the second type were also constructed. They do not test each individual step of the language, but they do validate the interaction between all components in the architecture. The scenarios are based on the running example of Section 3.1, one for import and one for export orders. The SLA used for the export scenario can be seen in Listing 8.1. The scenarios' SLAs define the following for a fictitious logistics service:

(1) Contract validity period
(2) Involved parties, both as roles and the corresponding actors
(3) A mock technical service description, as this is not functional in the prototype
(4) Agreements, including a description, the responsible parties, local variables, and the conditions of the agreement
    (a) The import scenario contains agreements about:
        i. Demurrage (see Figure 3.3)
        ii. Weighing requirement (as shown in Figure 3.1)
        iii. The time between placing the order and picking up the container
        iv. Delivery of port information by the customer
    (b) The export scenario contains agreements about:
        i. The time within which the container must be delivered to the customer for loading
        ii. The mode of transport used
        iii. The time between an order and it being planned
        iv. The container dwell time
        v. The time between loading the container at the customer and delivering it to the port

8.2.2 Tests

This section describes the functionality described in Section 8.1 was tested.

Item (1) The creation of SLAs with the DSL was tested by performing manual scenario tests. These tests were first performed with scenarios of the first type as described above. This verifies that all aspects of the language work and can be used to construct SLAs. In a later phase, the scenarios based

```
SLA Second Scenario
  template  : false
  version   : 1.0.0
  from      : 2023-07-01
  until     : 2024-01-01
  signed on : 2023-06-09

  Description :
    Second scenario test SLA.
    Tests order timing.

  Roles :
    Provider
    Planning contact
    Terminal contact
    BI contact
    Customer
    Add to role Customer :
      Example Company
    << ... >>

  Service Descriptions :
    LC Transport Export

  Interfaces :
    << ... >>

Export :
  Agreements for export orders
  agreements...
  Timing :
    Agreements for the timing of orders
    Planned within 1 day :
      A container is planned for the customer within one day after the order is made
      responsible roles:
        Provider
      variables:
        var isPlanned = plannedOn is present
        var planned = get plannedOn
        var diff = planned - get createdOn
        var timePassed = now - get createdOn
      guarantees:
        ( not isPlanned and timePassed less than P1D ) or ( isPlanned and diff less than P1D )
    Container at customer within 2 weeks :
      A container is delivered at the customer within 2 weeks
      responsible roles:
        Provider
      variables:
        var created = get createdOn
        var hasEta = customerEta is present
        var eta = get customerEta
      guarantees:
        hasEta → ( eta - created ) less than or equals P14D
    Express container uses truck :
      If container is required within 1 week, use a truck
      responsible roles:
        Provider
      variables:
        var created = get createdOn
        var hasEta = customerEta is present
        var eta = get customerEta
      guarantees:
        ( hasEta and eta - created less than or equals P7D ) → ( depotTerminal or else TRUCK ) equals TRUCK
    Dwell time within 2 days :
      The container should be at the terminal for at most 2 days
      responsible roles:
        Provider
      variables:
        var hasEta = terminalEta is present
        var hasEtd = terminalEtd is present
        var eta = get terminalEta
        var etd = get terminalEtd
        var hasAta = terminalAta is present
        var hasAtd = terminalAtd is present
        var ata = get terminalAta
        var atd = get terminalAtd
      guarantees:
        ( hasEta and hasEtd ) → ( etd - eta less than or equals P2D )
        ( hasAta and hasAtd ) → ( atd - ata less than or equals P2D )
    Container delivered to seaport within 1 week of stuffing :
      The container has to be delivered to a seaport within 1 week of loading it at the customer
      responsible roles:
        Provider
      variables:
        var hasEtd = customerEtd is present
        var etd = get customerEtd
        var hasEta = seaportEta is present
        var eta = get seaportEta
      guarantees:
        ( hasEtd and hasEta ) → eta - etd less than or equals P7D
```

Listing 8.1: SLA used for the export scenario, specified using our DSL.

on the running example were also used to create the SLAs used for the other tests.

Item (2) The exporting of SLAs was tested by manual integration testing. This was done by taking the SLAs created by the previous step, and manually verifying all information was correctly exported. In a later phase, the SLAs based on the running example's scenarios were also exported and verified, then used for the other tests.

Item (3) The importing of SLAs was also tested by manual integration testing. This was done by taking the exported SLAs of the previous step, and manually verifying all information was correctly exported. In a later phase, the exported SLAs based on the running example's scenarios were also imported and verified, then used for the other tests.

Item (4) The generation of monitoring reports was tested by scenario testing, supported by unit testing. The scenarios based on the running example were used, in combination with test data, to verify if correct reports were generated. The test data used tries to cover all options for a guarantee, for example see Table 8.1 for the test data used for the first guarantee of the export scenario (Listing 8.2). In some cases, a simplification was made to reduce the number of test cases, for example, in the dwell time guarantee of the export example, where the presence of the estimated and actual times where not tested separately. To further support this, the analysis step and reporting step were unit tested. The unit tests for the analysis step verify that every expression is correctly analysed, aiming for full branch coverage. An example these unit tests can be seen in Table 8.2. For the reporting step, the results of the generation of simple reports were verified with unit tests. These tests together verify the individual parts, as well as the whole process, with the exclusion of not implemented functionality as a result of the prototype's scope restriction.

Item (5) The response to monitoring events was tested by scenario testing, supported by unit testing. The scenarios based on the running example were used, in combination with test data, to verify correct notifications were generated. This is supported by the same unit tests as the previous step, verifying the underlying blocks. Further checks were performed to verify whether the correct operations could be determined to retrieve additional required data. These tests together verify the individual parts, as well as the whole process, with the exclusion of not implemented functionality as a result of the prototype's scope restriction.

8.3 Requirements Assessment

In Section 8.2 we discussed how we verified the functionality of all the "must have" requirements (Requirements 1 to 3). Therefore we conclude that our design satisfies these requirements.

The design of the monitoring system is mostly platform-agnostic, although it should be easily extendable, satisfying Requirement 4. Because the components in the design are decoupled and communicate only through their interfaces through the orchestrator, the design also satisfies Requirement 5. Although the monitoring system does not use an SLA specification written in our DSL as a direct input, the monitoring specification we generate is based directly on

Table 8.1: Example of test data used for scenario testing. This data was used to test the first agreement of the export scenario, also shown in Listing 8.2. Each row represents one of the test cases, with *createdOn* and *plannedOn* as inputs, and *diff* and *timePassed* as calculated variables.

createdOn	*plannedOn*	*diff*	*timePassed*	Result
2023-10-01 12:00:00	2023-10-01 20:00:00	*8 hours*		true
2023-10-01 12:00:00	2023-10-02 12:00:00	*24 hours*		false
Now - 12 hours			*12 hours*	true
Now - 36 hours			*36 hours*	false

```
Planned within 1 day :
  A container is planned for the customer within one day after the order is made
  responsible roles:
    Provider
  variables:
    var isPlanned = plannedOn is present
    var planned = get plannedOn
    var diff = planned - get createdOn
    var timePassed = now - get createdOn
  guarantees:
    ( not isPlanned and timePassed less than P1D ) or ( isPlanned and diff less than P1D )
```

Listing 8.2: First agreement of Listing 8.1, which checks whether an order is planned within one day. It uses two interface variables: *plannedOn* and *createdOn*. Its expression checks whether it is either not planned and less than a day has passed, or it is planned and the difference between *createdOn* and *plannedOn* is less than one day.

Table 8.2: Test data used in unit testing to verify the equivalence expression solver of the analyser component. The table shows both inputs and the result, with remarks in parentheses where needed.

Left Operand	Right Operand	Result
false	true	false
true	true	true
12	4	false
12	12	true
12	12 (different variable)	true
7.5	2.5	false
7.5	7.5	true
“Hello ”	“world!”	false
“Hello ”	“Hello ”	true
“Hello ”	12	false
12	12.0	false (different type)
1 day	2 days	false
1 day	1 day	true
2 days	1 day	false
2 days	2 days	true

our DSL. While other SLA specifications theoretically could generate a similar artefact, in practice it would be wiser to generate either a more easily accessible monitoring specification, or use a pivot model that could more easily work with other SLA specifications. With this, Requirement 6 is satisfied, but more work should be done to decrease the dependency on the SLA DSL.

Herewith we showed that the reporting functionality was verified, satisfying Requirement 7. The reporting functionality does not generate proper explanations however, and more work would be required to make it do so. Therefore Requirement 8 is not satisfied yet.

Chapter 9

Conclusions

In this thesis, we designed a DSL to specify SLAs for the purposes of monitoring, and designed a system that can automatically monitor an SLA and provide real-time decision support. Many providers of business services could benefit from such a system, as a lot of time is spent manually monitoring for violations. We have proposed a design for it, and have shown that such a system can work.

We started by obtaining information about the structure and contents of SLAs, and looking into existing SLA specification languages. Using this information, we defined requirements for an SLA specification language, and designed a DSL. We then verified that our DSL can be used for SLA monitoring, and we can therefore state that Research Question 2 is answered by this thesis.

To answer Research Question 3, we looked at existing solution for SLA monitoring systems, and designed a system based on those findings. We verified that providing real-time decision support based on SLA monitoring can be done, and described the architecture and designs required for it.

Having verified the designs for both the SLA specification language and monitoring system, we can conclude that this thesis also answered Research Question 1, and thereby all the research questions.

There are several major limitations to the current monitoring system. All components of the system are a proof-of-concept, and would need further work to become products for commercial use. The DSL, while functional, has two major problems. First, it requires some understanding of formal logic to be used. Second, when reading the DSL, it is unclear which operators have precedence, as parentheses are optional and the tree structure of an expression is not visible without manual interaction. Furthermore, the designs of the monitoring system are mostly at the conceptual level, and need to be improved upon.

Several things could be done to improve upon this thesis and make this a fully usable solution.

First, the DSL needs some improvements. The language in its current form is probably too mathematical to be used in practice. More research could be done improve its usability and clarity. It could also be extended to include features such as multi-tiered SLAs.

Second, the monitoring system could be extended. Currently, only a minimal prototype has been implemented. To be able to properly use the solution, a full implementation is required. This involves adding a monitor and orchestrator component, extending the explainer component, and updating the analyser

component to accommodate the changes.

Finally, if a full implementation is made, the monitoring solution could be verified further. The performance and scalability of the prototype needs to be evaluated, and the suitability of the system in practice needs to be tested.

Bibliography

[1] Cofano Software Solutions. "Cofano Software Solutions LinkedIn about page." (), [Online]. Available: https://www.linkedin.com/company/cofano-software-solutions-bv/about/ (visited on 04/10/2020).

[2] ——, "Terminal operating system." (2019), [Online]. Available: https://www.cofano.nl/nl/logistiek/terminals/terminal-operating-system/ (visited on 04/10/2020).

[3] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014, ISBN: 978-3-662-43838-1. DOI: 10.1007/978-3-662-43839-8.

[4] J. C. Nardi, R. de Almeida Falbo, J. P. A. Almeida, *et al.*, "A commitment-based reference ontology for services," *Inf. Syst.*, vol. 54, pp. 263–288, 2015. DOI: 10.1016/j.is.2015.01.012.

[5] D. C. Verma, "Service level agreements on IP networks," *Proc. IEEE*, vol. 92, no. 9, pp. 1382–1388, 2004. DOI: 10.1109/JPROC.2004.832969.

[6] R. Sprenkels and A. Pras, *Service Level Agreements : Internet NG Deliverable D2.7*, R. Sprenkels, Ed. Enschede: Centre for Telematics and Information Technology (CTIT), 2001.

[7] A. Maarouf, A. Marzouk, and A. Haqiq, "Practical modeling of the SLA life cycle in cloud computing," in *15th International Conference on Intelligent Systems Design and Applications, ISDA 2015, Marrakech, Morocco, December 14-16, 2015*, A. Abraham, A. M. Alimi, A. Haqiq, *et al.*, Eds., IEEE, 2015, pp. 52–58. DOI: 10.1109/ISDA.2015.7489170.

[8] N. B. Beaumont, "An overview of service level agreements," in *Outsourcing and Offshoring in the 21st Century: A Socio-Economic Perspective*, 2006, pp. 302–325. DOI: 10.4018/978-1-59140-875-8.ch014.

[9] N. Karten, "With service level agreements, less is more," *Inf. Syst. Manag.*, vol. 21, no. 4, pp. 43–44, 2004. DOI: 10.1201/1078/44705.21.4.20040901/84186.5.

[10] A. Paschke and E. Schnappinger-Gerull, "A categorization scheme for SLA metrics," in *Service Oriented Electronic Commerce: Proceedings zur Konferenz im Rahmen der Multikonferenz Wirtschaftsinformatik, 20.-22. Februar 2006 in Passau, Deutschland*, M. Schoop, C. Huemer, M. Rebstock, and M. Bichler, Eds., ser. LNI, vol. P-80, GI, 2006, pp. 25–40.

[11] K. Kritikos, B. Pernici, P. Plebani, *et al.*, "A survey on service quality description," *ACM Comput. Surv.*, vol. 46, no. 1, 1:1–1:58, 2013. DOI: 10.1145/2522968.2522969.

[12] W. Maurer, R. T. Matlus, and N. Frey, "A guide to successful SLA development and management," Gartner Group, Strategic Analysis Report, Oct. 16, 2000.

[13] K. Lu, R. Yahyapour, P. Wieder, *et al.*, "Fault-tolerant service level agreement lifecycle management in clouds using actor system," *Future Gener. Comput. Syst.*, vol. 54, pp. 247–259, 2016. DOI: 10.1016/j.future.2015.03.016.

[14] L. Wu and R. Buyya, "Service level agreement (SLA) in utility computing systems," *CoRR*, vol. abs/1010.2881, 2010. arXiv: 1010.2881.

[15] Wikipedia contributors. ""request for information — Wikipedia, the free encyclopedia." (2022), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Request_for_information&oldid=1091954681 (visited on 10/13/2022).

[16] A. Soomro and W. Song, "Developing and managing SLAs for business applications in information systems," in *Emerging Trends and Applications in Information Communication Technologies*, B. S. Chowdhry, F. K. Shaikh, D. M. A. Hussain, and M. A. Uqaili, Eds., Springer, 2012, pp. 489–500, ISBN: 978-3-642-28962-0. DOI: 10.1007/978-3-642-28962-0_46.

[17] A. Metzger, R. Franklin, and Y. Engel, "Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case," in *2012 Annual SRII Global Conference, San Jose, CA, USA, July 24-27, 2012*, IEEE Computer Society, 2012, pp. 313–322. DOI: 10.1109/SRII.2012.42.

[18] A. M. Gutiérrez, C. C. Marquezan, M. Resinas, A. Metzger, A. R. Cortés, and K. Pohl, "Extending ws-agreement to support automated conformity check on transport and logistics service agreements," in *Service-Oriented Computing - 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings*, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds., ser. Lecture Notes in Computer Science, vol. 8274, Springer, 2013, pp. 567–574. DOI: 10.1007/978-3-642-45005-1_47. [Online]. Available: https://doi.org/10.1007/978-3-642-45005-1%5C_47.

[19] C. C. Marquezan, A. Metzger, R. Franklin, and K. Pohl, "Runtime management of multi-level slas for transport and logistics services," in *Service-Oriented Computing - 12th International Conference, ICSOC 2014, Paris, France, November 3-6, 2014. Proceedings*, X. Franch, A. K. Ghose, G. A. Lewis, and S. Bhiri, Eds., ser. Lecture Notes in Computer Science, vol. 8831, Springer, 2014, pp. 560–574. DOI: 10.1007/978-3-662-45391-9_49. [Online]. Available: https://doi.org/10.1007/978-3-662-45391-9%5C_49.

[20] N. Moini, M. Boile, S. Theofanis, and W. Laventhal, "Estimating the determinant factors of container dwell times at seaports," *Maritime Economics and Logistics*, vol. 14, no. 2, pp. 162–177, 2012. DOI: 10.1057/mel.2012.3.

[21] S. Fazi and K. J. Roodbergen, "Effects of demurrage and detention regimes on dry-port-based inland container transport," *Transportation Research Part C: Emerging Technologies*, vol. 89, pp. 1–18, Apr. 1, 2018, ISSN: 0968-090X. DOI: 10.1016/j.trc.2018.01.012.

[22] M. G. Buscemi and U. Montanari, "Cc-pi: A constraint-based language for specifying service level agreements," in *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings*, R. De Nicola, Ed., ser. Lecture Notes in Computer Science, vol. 4421, Springer, 2007, pp. 18–32. DOI: 10.1007/978-3-540-71316-6_3.

[23] D. D. Lamanna, J. Skene, and W. Emmerich, "Slang: A language for defining service level agreements," in *9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003), 28-30 May 2003, San Juan, Puerto Rico, Proceedings*, IEEE Computer Society, 2003, p. 100. DOI: 10.1109/FTDCS.2003.1204317.

[24] K. T. Kearney, F. Torelli, and C. Kotsokalis, "Sla⋆: An abstract syntax for service level agreements," in *Proceedings of the 2010 11th IEEE/ACM International Conference on Grid Computing, Brussels, Belgium, October 25-29, 2010*, IEEE Computer Society, 2010, pp. 217–224. DOI: 10.1109/GRID.2010.5697973.

[25] K. T. Kearney and F. Torelli, "The SLA model," in *Service Level Agreements for Cloud Computing*, P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, Eds., New York, NY: Springer, 2011, pp. 43–67, ISBN: 978-1-4614-1614-2. DOI: 10.1007/978-1-4614-1614-2_4.

[26] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," *J. Netw. Syst. Manag.*, vol. 11, no. 1, pp. 57–81, 2003. DOI: 10.1023/A:1022445108617.

[27] A. Alain, C. Karl, D. Asit, *et al.*, "Web services agreement specification (WS-agreement)," *Global Grid Forum*, Oct. 10, 2011. [Online]. Available: http://www.ogf.org/documents/GFD.192.pdf (visited on 01/13/2021).

[28] E. Oberortner, U. Zdun, and S. Dustdar, "Tailoring a model-driven quality-of-service DSL for various stakeholders," in *ICSE Workshop on Modeling in Software Engineering, MiSE 2009, Vancouver, BC, Canada, May 17-18, 2009*, IEEE Computer Society, 2009, pp. 20–25. DOI: 10.1109/MISE.2009.5069892. [Online]. Available: https://doi.org/10.1109/MISE.2009.5069892.

[29] C. Müller, M. Oriol, X. Franch, *et al.*, "Comprehensive explanation of SLA violations at runtime," *IEEE Trans. Serv. Comput.*, vol. 7, no. 2, pp. 168–183, 2014. DOI: 10.1109/TSC.2013.45.

[30] Eclipse Foundation. "Eclipse Modeling Framework (EMF)." (), [Online]. Available: https://www.eclipse.org/modeling/emf/ (visited on 10/26/2022).

[31] JetBrains. "MPS: The Domain-Specific Language Creator." (), [Online]. Available: https://www.jetbrains.com/mps/ (visited on 10/26/2022).

[32] Eclipse Foundation. "Xtext - Language Engineering Made Easy!" (), [Online]. Available: https://www.eclipse.org/Xtext/ (visited on 10/26/2022).

[33] ——, "Eclipse OCL (Object Constraint Language)." (), [Online]. Available: https://projects.eclipse.org/projects/modeling.mdt.ocl (visited on 10/26/2022).

[34] ——, "ATL - a model transformation technology." (), [Online]. Available: https://www.eclipse.org/atl/ (visited on 10/26/2022).

[35] Object Management Group, *Meta Object Facility (MOF) 2.0 Query / View / Transformation Specification*, Version 1.3, Jun. 2016. [Online]. Available: ttp://www.omg.org/spec/QVT/1.3 (visited on 10/26/2022).

[36] M. Comuzzi, C. Kotsokalis, G. Spanoudakis, and R. Yahyapour, "Establishing and monitoring slas in complex service based systems," in *IEEE International Conference on Web Services, ICWS 2009, Los Angeles, CA, USA, 6-10 July 2009*, IEEE Computer Society, 2009, pp. 783–790. DOI: 10.1109/ICWS.2009.47.

[37] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Comprehensive qos monitoring of web services and event-based sla violation detection," in *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing*, Association for Computing Machinery, 2009, pp. 1–6. DOI: 10.1145/1657755.1657756.

[38] F. Raimondi, J. Skene, and W. Emmerich, "Efficient online monitoring of web-service slas," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*, M. J. Harrold and G. C. Murphy, Eds., ACM, 2008, pp. 170–180. DOI: 10.1145/1453101.1453125.

[39] A. Sahai, V. Machiraju, M. Sayal, A. P. A. van Moorsel, and F. Casati, "Automated SLA monitoring for web services," in *Management Technologies for E-Commerce and E-Business Applications, 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2002, Montreal, Canada, October 21-23, 2002, Proceedings*, M. Feridun, P. G. Kropf, and G. Babin, Eds., ser. Lecture Notes in Computer Science, vol. 2506, Springer, 2002, pp. 28–41. DOI: 10.1007/3-540-36110-3_6.

[40] M. Palacios, J. García-Fanjul, J. Tuya, and C. de la Riva, "A proactive approach to test service level agreements," in *The Fifth International Conference on Software Engineering Advances, ICSEA 2010, 22-27 August 2010, Nice, France*, J. G. Hall, H. Kaindl, L. Lavazza, G. Buchgeher, and O. Takaki, Eds., IEEE Computer Society, 2010, pp. 453–458. DOI: 10.1109/ICSEA.2010.77.

[41] M. Di Penta, G. Canfora, G. Esposito, V. Mazza, and M. Bruno, "Search-based testing of service level agreements," in *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007*, H. Lipson, Ed., ACM, 2007, pp. 1090–1097. DOI: 10.1145/1276958.1277174.

[42] M. A. R. Gonzalez, P. Chronz, K. Lu, *et al.*, "G-SLAM – the anatomy of the generic SLA manager," in *Service Level Agreements for Cloud Computing*, P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, Eds., New York, NY: Springer, 2011, pp. 167–186, ISBN: 978-1-4614-1614-2. DOI: 10.1007/978-1-4614-1614-2_11.

[43] J. Happe, W. Theilmann, A. Edmonds, and K. T. Kearney, "A reference architecture for multi-level SLA management," in *Service Level Agreements for Cloud Computing*, P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, Eds., New York, NY: Springer, 2011, pp. 13–26, ISBN: 978-1-4614-1614-2. DOI: 10.1007/978-1-4614-1614-2_2.

[44] "The TrustCoM project. deliverable 64: Final TrustCoM reference implementation and associated tools and user manual," Jun. 2007.

[45] K. Mahbub and G. Spanoudakis, "Monitoring *WS-Agreement* s: An event calculus-based approach," in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds., Springer, 2007, pp. 265–306. DOI: 10.1007/978-3-540-72912-9_10.

[46] G. Spanoudakis and K. Mahbub, "Non-intrusive monitoring of service-based systems," *Int. J. Cooperative Inf. Syst.*, vol. 15, no. 3, pp. 325–358, 2006. DOI: 10.1142/S0218843006001384.

[47] M. Comuzzi and G. Spanoudakis, "Dynamic set-up of monitoring infrastructures for service based systems," in *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal, and C. Hung, Eds., ACM, 2010, pp. 2414–2421. DOI: 10.1145/1774088.1774591.