Reference Architecture for an SLA Management Framework

Whitepaper

Version 1.0; December 10, 2010

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 216556.

Contributors

Partner	Contributors
SAP	Wolfgang Theilmann, Jens Happe, Tariq Ellahi
ENG	Francesco Torelli, Keven Kearney
TID	Juan Lambea, Beatriz Fuentes
XLAB	Miha Vuk
PMI	Sam Guinea
Intel	Andrew Edmonds, John Kennedy, Michael Nolan
FZI	Giovanni Falcone, Franz Brosch
UDO	Costas Kotsokalis, Peter Chronz, Edwin Yaqub

Notices

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. Copyright 2009 by the SLA@SOI consortium.

* Other names and brands may be claimed as the property of others.

This work is licensed under a Creative Commons Attribution 3.0 License.

Document History			
Version	Date	Author	Changes
1.0	10 December 2010	SLA@SOI consortium	Initial public version of this whitepaper.

Executive Summary

This document provides the specification of the reference architecture for an SLA management framework as developed by the EU FP7 project SLA@SOI.

This architecture directly responds to the mission of SLA@SOI: "to deliver and showcase an innovative open SLA Management Framework that provides holistic support for service level objectives - enabling an open, dynamic, SLA-aware market for European service providers. SLAs will be managed autonomously throughout the complete service lifecycle, spanning the entire services stack from the business layer through to infrastructure. Arbitrary domains will be supported, as demonstrated by evaluations in wide-ranging, grounded, use cases". Furthermore it integrates the contributions on the four top-level technical objectives of (1) Consistent SLA-management framework, (2) Adaptive SLA-aware infrastructure, (3) Engineering predictable service-oriented systems, and (4) Comprehensive business management suite for e-contracting into one consistent architecture.

The reference architecture definition was driven by the requirements of four industrial use cases but also by other internal and project-external stakeholders.

The reference architecture represents a key innovation of SLA@SOI as it realizes the first comprehensive architecture of a consistent SLA-management framework. Four main novelties can be highlighted: (1) the architecture supports multi-layered SLA management where SLAs can be composed and decomposed along functional & organizational domains; (2) it supports arbitrary service types (business, software, and infrastructure) and SLA terms; (3) the architecture covers the complete SLA and service lifecycle with consistent interlinking of design-time, planning and run-time management aspects; (4) the actual implementation supports flexible deployment setups, where actual components can be flexibly selected, extended and connected and where founding data models can be extended.

This document provides a complete overview of the framework architecture including a high-level overview, a detailed discussion of relevant foundational concepts, a description of the modelling foundation, and the actual architecture overview with its building blocks, components and interactions.

The reference architecture has been successfully implemented into an SLA management software framework. This framework is provided as open source project under BSD licence [1]. The software framework has been successfully applied against four industrial use cases in the domains of ERP hosting, Enterprise IT management, Telco Service Aggregation and eGovernment.

Table of Contents

1 Introduction 9

2 Context and Motivation 10
2.1 Overall goal 10
2.2 Technical perspective 11

3 Scope and Foundations 13
3.1 Scope 13
3.2 Services, Resources and SLAs 14
3.3 Management of Services and SLAs 16
3.3.1 Definition of "SLA Management" 16
3.3.2 Service Lifecycle 16
3.3.3 Management domains 17
3.4 Building blocks 18
3.4.1 Main components 18
3.4.2 Component setups 20

4 Architecture Specification 23
4.1 Goals and Overview 23
4.1.1 Top-level Architecture 23
4.1.2 Components and Interaction 25
4.2 Foundation Models 27
4.2.1 SLA Model 27
4.2.2 Service Construction Metamodel 27
4.2.3 Open Cloud Computing Interface 28
4.2.4 Palladio Component Model 28
4.3 Components 29
4.3.1 Generic SLA Manager 29
4.3.2 Business SLA Manager 29
4.3.3 Business Manager 29
4.3.4 Software SLA Manager 29
4.3.5 Software Service Manager 30
4.3.6 Infrastructure SLA Manager 30
4.3.7 Infrastructure Service Manager 30
4.3.8 Service Evaluation 30
4.3.9 Manageability Agent 30

5 Interaction Stereotypes 30
5.1 <<customer_relations>> 30
5.2 <<provider_relations>> 33
5.3 <<query_product_catalog>> 34
5.4 <<negotiate/renegotiate >> 36
5.5 <<control/track/query>> 40
5.6 <<negotiate/query/coordinate>> 41
5.7 <<prepare software services>> 42
5.8 <<manage_software_service>> 44
5.9 <<prepare_infrastructure_services>> 47
5.10 <<manage_infrastructure_service>> 48
5.11 <<evaluate>> 51
5.12 <<subscribe_to_event>> 52
5.13 <<publish_event>> 59

6 Interaction Sequences 60
6.1 Negotiation interaction 60
6.2 Provisioning interaction 63
6.3 Adjustment & Reporting interactions 66

6.4 Renegotiation interactions 67

7 Manageability approach 68
7.1 Manageability 70
7.2 Monitoring 71
7.3 Adjustment 78

8 Adoption Examples 80
8.1 ERP Hosting 80
8.2 Enterprise IT 82
8.3 E-Government 83

9 Conclusions 86
9.1 Summary 86
9.2 Further reading 86
9.3 Outlook on Future Work 86

10 References 88

Appendix A: Glossary 90

Appendix B: Abbreviations 93

Table of Figures

Figure 1: Envisioned interaction. 11
Figure 2: Service concepts and their relations to each other. 15
Figure 3: Service Lifecycle. 17
Figure 4: Generic building blocks and their relations. 18
Figure 5: A possible domain split. 21
Figure 6: Building blocks in the envisioned interaction. 21
Figure 7: Top-level view of the SLA@SOI framework. 24
Figure 8: Component Specialisation Hierarchies. 25
Figure 9: << customer_relations :ICustomerRelations>> sequence diagram 31
Figure 10: << customer_relations: IReporting>> sequence diagram 32
Figure 11: << IProviderRelations>> sequence diagram 34
Figure 12: << IQueriableProductCatalogue: getProducts>> sequence diagram. 35
Figure 13: << IQueriableProductCatalogue: getTemplates >> sequence diagram 36
Figure 14: Interaction Overview 37
Figure 15: Interaction showing successful negotiation, renegotiation & termination of SLA. 38
Figure 16: Interaction showing negotiation & cancelling negotiation 39
Figure 17: << control/track/query: Track >> sequence diagram 41
Figure 18: <<Prepare_Software_Services>> sequence diagram 43
Figure 19: Managed_Software_Services interaction sequence 46
Figure 20: Prepare Infrastructure Services Sequence Diagram 48
Figure 21: Sequence Diagram for <<Manage_Infrastructure_Service>> 50
Figure 22: High-level Interaction of POC (via SLAManager) and ServiceEvaluator 52
Figure 23: Generic sequence diagram showing the subscription mechanism that allows the communication between SLAManager, ServiceInstance in order to send/receive monitoring events through the MonitoringEventChannel. 54
Figure 24: Class diagram for <<subscribe_to_event>> 55
Figure 25: ISubscription API 56
Figure 26: Full sequence diagram of <<subscribe_to_event>> interaction 58
Figure 27: Negotiation interactions on the business layer. 61
Figure 28: Negotiation interactions between software and infrastructure layer. 62
Figure 29: Interactions during the creation of an agreement. 63
Figure 30: Explicit provisioning triggered by the customer. 64
Figure 31: Implicit provisioning of infrastructure services triggered by a timer inside the Infrastructure Manager. 64

Figure 32: Implicit provisioning of software services triggered by the Software Service Manager. 65
Figure 33: Interactions for reporting the status of SLAs and their violation. 66
Figure 34: Interactions for the renegotiation of SLAs. 67
Figure 35: Manageability, Monitoring and Adjustment system architecture 69
Figure 36: Layers of Monitoring System 72
Figure 37: LLMS inner structure, interfaces and interactions 73
Figure 38: LLMS Interaction sequence diagram 75
Figure 39: Sensor interface 76
Figure 40: Reasoning component gateway interface 77
Figure 41: Reasoning component gateway methods 77
Figure 42: Adopted framework architecture for ERP Hosting use case. 81
Figure 43: Adopted framework architecture for Enterprise IT use case. 83
Figure 44: Adopted framework architecture for eGovernment use case. 85

1 Introduction

This document describes the reference architecture for the SLA management framework created within the second year of the European research project SLA@SOI. The architecture definition includes foundational concepts on terms and lifecycles, a modelling foundation and the actual architecture in terms of components and interactions.

The framework's architecture mainly focuses on separation of concerns, related to SLAs and services on the one hand, and to the specific domain (e.g., business, software, and infrastructure) on the other. Service Managers are responsible for all management activities directly related to services. This includes the management of information about available services, supported types of services, as well as their offered functionality and their dependencies. SLA Managers are responsible for all actions that are related to the service-level agreements. They are involved in the negotiation with customers and they are responsible for the planning and optimization of new services that are to be provisioned. Furthermore, they monitor the terms a provider and customer have agreed upon and react in case of violations. SLA Managers can negotiate with each other in order to find the best offer for a customer. The provisioning of a service is a joined effort of all SLA Managers and Service Managers involved. In order to support multiple domains with our framework, multiple SLA managers and multiple Service Managers can collaborate inside the framework as well as across framework boundaries. Thereby, each SLA Manager and Service Manager is responsible for SLAs and services of a particular domain.

The remainder of this document is structured as follows. Chapter 2 provides an overview of the context and the motivation under which the architecture has been developed. Chapter 3 introduces the actual scope and foundational concepts, and provides an overview of the main architectural concepts and its modelling foundation i.e. the most important meta-models that are shared between different components of the architecture. Chapter 4 describes the reference architecture with its main components and interactions. Chapter 8 provides a few examples how the architecture is applied in industrial use cases, and Chapter 9 concludes with a short summary and outlook on future work.

2 Context and Motivation

In order to put the specification of the reference architecture into context we first summarize the general goals of our approach and its intended results.

2.1 Overall goal

The research project SLA@SOI aims at providing a major milestone for the further evolution towards a service-oriented economy, where IT-based services can be flexibly traded as economic goods, i.e. under well defined and dependable conditions and with clearly associated costs. Eventually, this will allow for dynamic value networks that can be flexibly instantiated thus driving innovation and competitiveness

The project SLA@SOI envisions

a business-ready service-oriented infrastructure

empowering the service economy in a flexible and dependable way

where business-readiness requires the following three major characteristics

- *Predictability and Dependability*: The quality characteristics of services can be predicted and enforced at run time.
- *Transparent SLA management*: Service level agreements (SLAs) defining the exact conditions under which services are provided/consumed can be transparently managed across the whole business and IT stack.
- *Automation*: The whole process of negotiating SLAs and provisioning, delivery and monitoring of services will be automated allowing for highly dynamic and scalable service consumption.

A motivating business scenario highlighting the project idea is a service provider who is enabled to offer services with differentiated, dependable and adjustable SLAs, and can negotiate concrete SLAs with (individual or groups of) customers in an automated fashion. This business goal imposes additional requirements on software providers (to provide components with predictable non-functional behaviour) and infrastructure providers (to support an SLA aware management of resources).

This vision maps to the overarching challenge for a service-oriented infrastructure (SOI) that supports consistent SLA management across all layers of an IT stack and across the various stakeholder perspectives. Noteworthy, the SLA characteristics may span across multiple non-functional domains such as security, performance, availability, reliability.

Figure 1 gives a simplified overview on how such a systematic SLA management process may look like. As today's business systems typically consist of complex layered systems, user-level SLAs cannot be directly mapped onto the physical infrastructure. Services might be composed of other more fundamental services that could be also provided by external parties. Consequently, a stepwise mapping of higher-level SLA requirements onto lower levels and the aggregation of lower-level capabilities to higher levels is crucial for grounding user-level SLAs to the infrastructure. This vertical information flow must carefully reflect service interdependencies as well as the originating business context. In addition to SLAs, the vertical information flow also covers monitoring, tracking, and accounting data and must support brokering and negotiation processes at each layer. As shown in the figure, the overall SLA management process may include different stakeholders, namely customers, service and infrastructure providers, and also

various business steps such as business assessment, contracting and sales. The overview is intentionally simplified in the sense that no service chains are visualized. Such chains would represent all cases where service providers rely on other external providers.

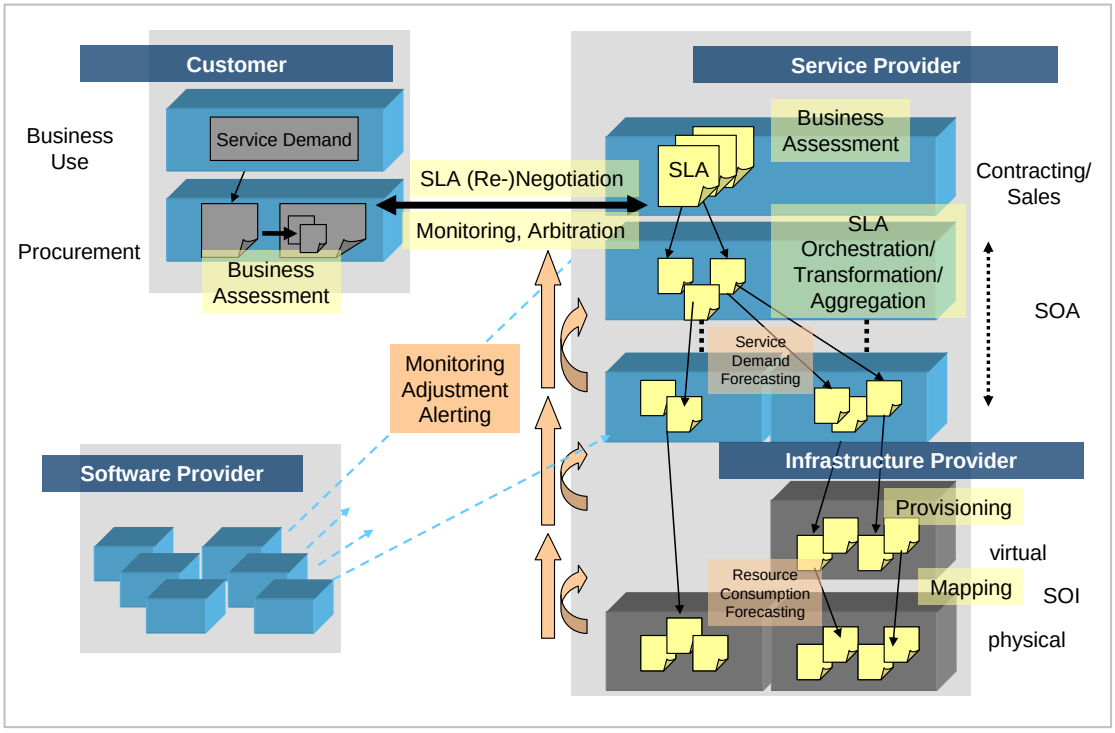


Figure 1: Envisioned interaction.

2.2 Technical perspective

Service Level Agreements (SLAs) are a common way to formally specify the exact conditions (both functional and non-functional behaviour) under which services are or shall be delivered. However, the current SLAs in practice are just specified at the top-level interface between a service provider and a service customer. Top-level SLAs can be used by customers and providers to monitor whether an actual service delivery complies with the agreed SLA terms. In case of SLA violations penalties or compensations can be directly derived.

However, top-level SLAs do not allow service providers to either plan their IT landscapes according to possible, planned or agreed SLAs. Nor do they allow understanding why a certain SLA violation might have occurred. The reason for this is that SLA guarantee terms are typically not explicitly or directly related to actual performance metrics or configuration parameters. This makes it difficult for service providers to derive proper configuration parameters from top-level SLAs and to assess (lower-level) monitoring metrics against top-level SLAs.

Overall, the missing relation between top-level SLAs and (lower-level) metrics and parameters is a major hurdle for managing IT stacks in terms of IT planning, prediction or adjustment processes and in accordance with possible, planned or actual SLAs.

The technical vision is to use the paradigm of SLAs for managing a complete IT stack in correlation with top-level SLAs which are agreed at business level. This complies very well with the technical trend to apply the paradigm of service-orientation across the complete IT stack (infrastructure/platform/software as a service) but also with the organizational trend in IT companies to organize different departments as service departments (providing infrastructure resources,

middleware, applications or composition tools as a service). SLAs will be associated with multiple elements of the stack at multiple layers, e.g. SLAs for elements of the physical infrastructure, virtualized infrastructure, middleware, application level and process-level. Such internal SLAs describe the contract between the lower-level entity and a higher-level entity which consumes the lower one. More precisely, the SLAs specify the required or agreed performance metrics but also the related configuration parameters.

The fundamental challenge for realizing this vision is how to properly correlate the different SLAs in such a scenario so that they form a well synchronized SLA hierarchy.

3 Scope and Foundations

This chapter provides the foundation for the architecture discussion as it sets the basic scope and introduces the main concepts/terms and their relationship. Note that more detailed definitions around the notion of services, SLAs, roles and IT systems can be found in the SLA@SOI glossary [2]. A summary, of those terms is given in Appendix A.

3.1 Scope

Functional Goals

The primary functional goal of our SLA management framework is to provide a generic solution for SLA management that

1. supports SLA management across multiple layers with SLA composition and decomposition across functional and organizational domains;
2. supports arbitrary service types (business, software, infrastructure) and SLA terms;
3. covers the complete SLA and service lifecycle with consistent interlinking of design-time, planning and runtime management aspects; and
4. can be applied to a large variety of industrial domains and use cases.

In order to achieve these goals, the reference architecture is based on three main design principles. First, we put a strong emphasis on a clear separation of concerns, by clearly separating service management from SLA management and by supporting a well layered and hierarchical management structure.

Second, a solid foundation in common metamodels for SLAs as well as their relation to services and the construction of actual service instances is an important aspect to support clear semantics across different components of the framework.

Third, design for extensibility/adaptability is a key aspect in order to address multiple domains. Therefore, we clearly distinguish between generic solution elements and places where domain specific logic/models need to be provided. Furthermore, we seek for an architecture where even generic parts can be replaced by domain specific versions, which might be dictated by already existing (legacy) management functionality.

Technical goals

A set of technical requirements and goals has been collected from various industrial use cases and external stakeholders. They fall into the four categories of Framework Configuration & Setup, Framework Model Configuration, Framework Operation, and Framework Access.

Model-related requirements are mainly about model extensibility and are addressed by the design of the SLA model and the service construction model.

Other requirements relate to the usage of certain technology standards and are taken into account by the actual framework development [5].

Full details about these requirements and their evaluation can be found in deliverable D.B1a [15].

Use-Case driven Approach

Both, functional and technical aspects have been developed based on requirements collected from other stakeholders. The main stakeholders considered are: the four industrial use cases within the project, project partners contributing direct insight from their organization, and project-external organizations interested in the topic of SLA management. Further details on this process are described in [15].

3.2 Services, Resources and SLAs

A first and fundamental concept for the architecture is the clear distinction between resources, services and SLAs.

Following the ITIL definitions [9] and in accordance with the SLA@SOI glossary we can define them as follows:

- **Service**: A means of delivering value to Customers by facilitating Outcomes Customers want to achieve without the ownership of specific Costs and Risks.
- **Resource**: A generic term that includes IT Infrastructure, people, money or anything else that might help to deliver an IT Service. Resources are considered to be Assets of an Organisation.
- **SLA**: An Agreement between a service provider and a customer. The SLA describes the service, documents service level targets, and specifies the responsibilities of the service provider and the customer. A single SLA may cover multiple services or multiple customers.

To further stress the distinction between these concepts we can state that services are about the activity to bring value to customers. They are not about the artefacts needed at the provider side to deliver that value. However, it's probably safe to assume that services always require some resource(s) as means for their delivery. The following 2 examples show this distinction:

- A hotel booking service listed in a public registry requires hotel rooms and some software as resources for the actual service delivery (and possibly also human resources). A service endpoint may refer to a hotel room but neither the hotel nor a hotel room is a service.
- A compute service provided by Amazon requires infrastructure resources but also some management software resources. A service endpoint may refer to an infrastructure entity (such as a computer) but the entity is not a service.

In order to close the gap between abstract services and concrete resources that are eventually needed for a deliverable, we further refine the concept of services and their concreteness via 3 specializations:

- **Service Type**: Specifies the service as a fully abstract entity via its external interface.
- **Service Implementation**: describes specific resources or artefacts (such as software components, or appliances) which allow for instantiating the

service. Service implementations may still depend on other services. There can be different implementations of a given service type.

- **Service Instance**: is a running and accessible service which is ready for consumption by service users. It has one or more service endpoints (for service consumption) and a management endpoint (for service monitoring & control). Service instances might have multiple service/management endpoints if their service type specifies a bundled service.

As an example for these concepts we can take a database service.

The abstract type of such a service is mainly specified by the fact that the service is exposed via an SQL interface.

Different service implementations may exist for such a database service, for example a MySQL database or an IBM DB2 database. These implementations may rely on other services such as a storage service.

For each implementation, multiple services instances might be created. These may differ from each other either in their concrete configuration. For example, one instance might be configured for optimized read access, another one for fast write access.

The following diagram shows now the main concepts coined for services and SLAs as well as their relationships.

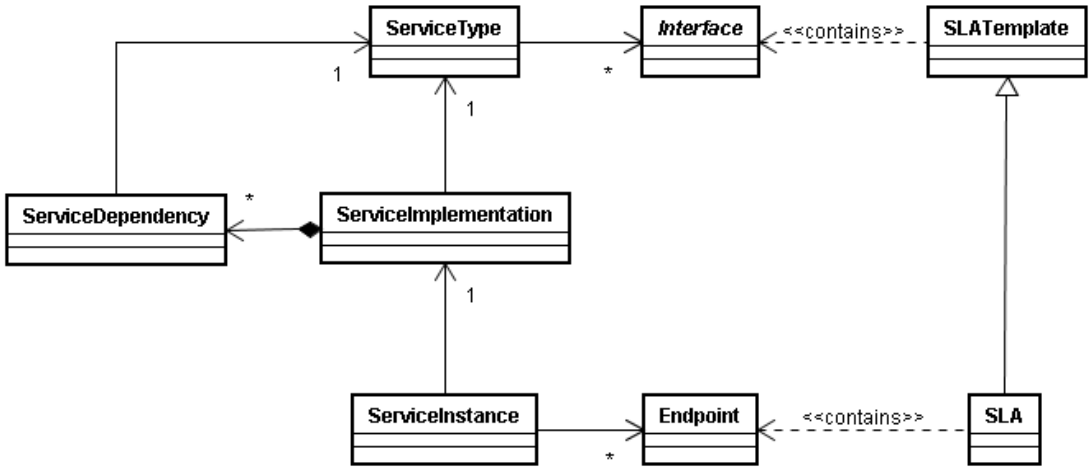


Figure 2: Service concepts and their relations to each other.

A few further explanations:

SLA Templates specify the types of SLA offers a service provider advertises it is willing to accept.

An SLA represents a potential agreement between a service provider and a customer that describes the service, documents service level targets, and specifies the responsibilities of the service provider and the customer. An agreed SLA also refers to the endpoints of exactly one service instance. For example, an SLA for an instance of the address validation service contains the Web service endpoints for invoking the validation functionality.

Service dependencies relate to service types a given implementation relies upon. These dependencies must be resolved to concrete instances of the depending sub service in order to instantiate the higher level services.

3.3 Management of Services and SLAs

Following the core concepts, we now briefly sketch our notion of management and the related lifecycles of SLAs and services.

3.3.1 Definition of "SLA Management"

The term management is interpreted here as "control" - in the classic control-theory sense; synonymous with "applied constraint". In particular, the relation between 'manager' & 'managed' is defined as a control relation – formally:

> **Given a "managed" system, S, with n degrees of freedom, the "manager" M "manages" S if M acts upon S to reduce the degrees of freedom of S to m<n.**

We also assume that the management actions enacted by M upon S are goal-based – i.e. they serve to satisfy one or more management objectives, which are in some way dependent on the state of S. To this end, the management relation is necessarily bi-directional; the application of management entails a continuous feedback loop in which M observes the dynamic state of S, and acts upon S in order to constrain its state dynamics in some way. Both observation (sensing) and action are necessarily mediated by information exchange (appropriate interfaces are defined in Appendix B). Finally, management systems can be:

- **hierarchically organised**: each level operates under the constraints imposed by higher levels, and serves in turn to constrain lower levels.
- **distributed**: to the extent permitted by the communication channels supporting the management relation.

We interpret "SLA Management" as the management of service delivery systems in order to meet the QoS objectives (goals) specified in SLAs. SLA management covers all stages in the SLA lifecycle:

- SLA Template design : ensuring that offered QoS guarantees are realistic,
- SLA negotiation : ensuring that agreed QoS guarantees are realisable,
- SLA runtime (effective period of SLAs) : ensuring that QoS guarantees are satisfied,
- SLA(Template) archiving : ensuring that previous experience is available to future cycles

3.3.2 Service Lifecycle

The management of SLAs happens in the context of the overall service lifecycle, which is depicted in the following figure.

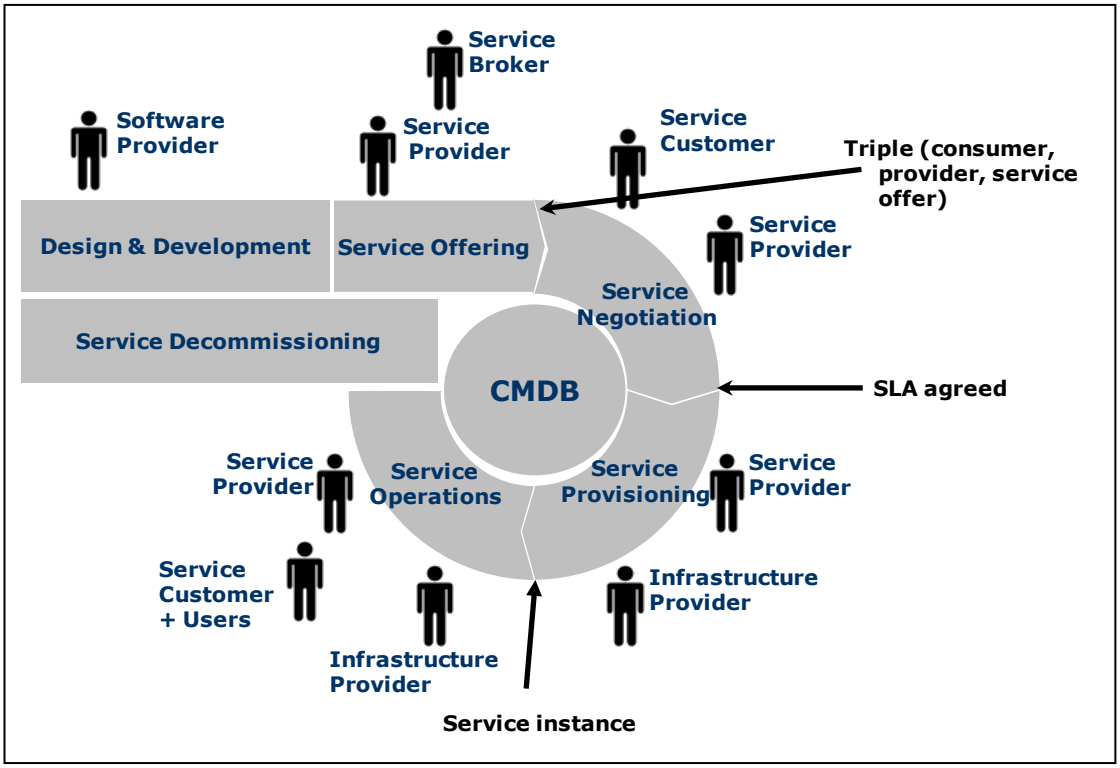


Figure 3: Service Lifecycle.

The lifecycle consists of the following stages:

- Design and Development: development of artefacts needed for service implementation
- Service Offering (incl. SLA template design): offering a service (type) to customers; results include specification of SLA templates
- Service Negotiation (incl. parts of SLA negotiation): actual negotiation between customer and provider; results in an agreed SLA
- Service Provisioning (incl. parts of SLA negotiation): all activities required in system preparation and setup for allowing service operation, including booking, deployment (if needed), and configuration. Note that provisioning does not necessarily imply deployment as for example in a multi-tenant environment the provisioning of a new tenant might be a simple reconfiguration of the running system.
- Service Operations (incl. SLA runtime): an actual service instance is up and running; it might be adjusted in order to enforce an SLA
- Service Decommissioning: the service instance is stopped and can no more be accessed by the service customer

A more detailed view on this service lifecycle is provided in [4].

3.3.3 Management domains

Another important aspect of management is the notion of management domains. So far we distinguish two main kinds of domains, the first driven by business considerations, the second driven by technical considerations.

1. **Administrative domains** are areas of organizational coherence, e.g. an independent organization or a department that operates largely as a profit

centre. Within an administrative domain two main views can be considered:

a. The business view representing basically a sales department, i.e. the activity to sell services via SLAs.

b. The management view that oversees all the offered or active SLAs within a certain domain and which is responsible for the eventual SLA operation.

2. **Technical domains** are areas where a certain kind of resources or artefacts can be coherently managed, e.g. domains for infrastructure artefacts, software artefacts, business artefacts or even subdivisions of these.

Technical domains could be understood as horizontal layers within a business/IT stack, while administrative domains relate more to vertical, cross-cutting pillars within an organization (though they can form a hierarchy as well). Section 3.4.2 gives insight on how the notion of domains impacts the architecture.

3.4 Building blocks

In this chapter we introduce the main building blocks that constitute our framework, explain their responsibilities, show how they can be specialized for specific domains, and explain how they can be combined in order to serve different scenarios and setups.

3.4.1 Main components

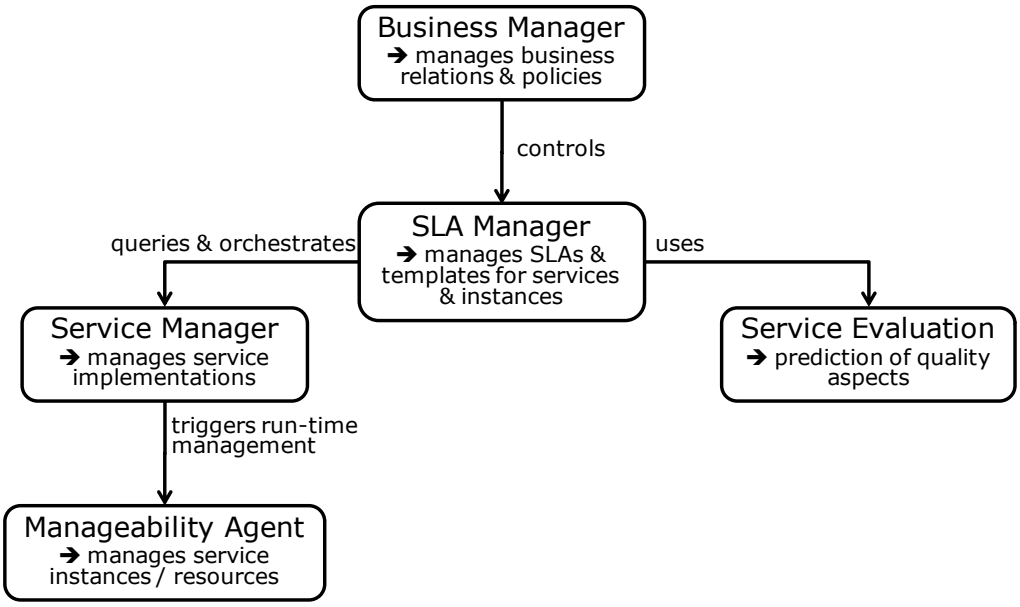


Figure 4: Generic building blocks and their relations.

Figure 4 gives an overview of the framework's main components and their relations. The "leading" component is the Business Manager which is responsible for business related information and business-driven decisions. It controls the SLA Manager which is responsible for SLA templates and actual SLAs. It uses the Service Manager for querying service implementations and orchestrating

provisioning activities. The Service Manager is responsible for managing actual service implementation. It uses Manageability Agents for triggering run-time management activities. Last, the SLA Manager also relies on the Service Evaluation for retrieving predictions of service qualities. More detailed discussions of these components and their relations follow below.

Taking the business-rooted ambition of an SLA management framework, the root of the management hierarchy is the **Business Manager** component. It is responsible for asserting overall business constraints on the system in order to meet business objectives and for maintaining customer and provider relations. To that extent, it captures knowledge about pricing schemes (incl. rewards, promotions and discounts), customer profile information, 3rd party service provider profiles and business rules for taking cost/profit-aware decisions. Business managers may contain sensible data that must not be shared among components. The actual functionality of a Business Manager includes:

- searching and publishing of products
- customers & service providers management
- negotiation and establishment of agreements/contracts with customers and service providers
- notification of bills & penalties to customers and service providers

Following the concepts introduced in Section 3.1, we consider the following concepts as core to our architecture: SLA, service, and resource. Consequently the architecture contains dedicated management components for all three of them.

The **SLA Manager** component is responsible for managing a set of SLA Templates and SLAs in its domain. Furthermore, it captures knowledge about negotiation and planning goals (such as utility functions or policies). Depending on the specific context/requirements of a use case a separate SLA Manager may be set up for a complete organization, per department, or for each individual service. The actual functionality of an SLA Manager includes:

- searching and publishing of SLA templates
- negotiation of SLAs with customers and 3rd parties including conversion between different SLA formats
- SLA planning and optimization
- SLA provisioning and adjustment

The **Service Manager** component is in charge of managing the elements necessary to instantiate a service. In particular it knows about the structure of service implementations and keeps track of existing service instances. Service Managers can be created for any technical domain which needs consistent management. The actual functionality of a Service Manager includes:

- publishing of service implementations
- maintenance of a service landscape, incl. elements required for instantiating a service implementation
- reservation and booking of service instances
- mediation of management/adjustment operations to service instances and Manageability Agents
- triggering of actual service provisioning

The **Manageability Agent** component acts as gateway to actual resources. It knows about the available sensors and effectors that can be used for managing a

certain service instance and its resources. Manageability Agents may exist per resource, per service instance or per collections of these. The actual functionality of a Manageability Agent includes:

- sensing/monitoring the status of service instances and resources
- searching for and executing manageability actions

Finally, to support pro-active management decisions, at all levels, the framework also provides a **Service Evaluation** component. It relies on background information (from design-time, run-time or historical) about the quality characteristics of services. It provides functionality for a priori quality evaluations of services – depending on influencing factors such as customer behaviour or lower-level service qualities. Service Evaluation components may exist per SLA Manager or even for sets of these.

While all these components have clearly distinct responsibilities, they also need to have some common understanding of the overall problem domain.

- Service types must be commonly understood by SLA managers, service managers and service evaluation.
- The identity of service implementations must be commonly understood by service managers and service evaluation, though both components may rely on completely different data models in order to deal with service implementations.
- SLA terms and their quality aspects must be commonly understood by SLA managers and service evaluation.
- SLA terms and available monitoring handles must be commonly understood by SLA managers and service managers.
- The SLA negotiation process with customers and thirds parties must be commonly understood by business managers and SLA managers.

3.4.2 Component setups

As stated in the design goals for the framework architecture, a key goal is the support for flexible configurations and setups where different domain cuts can be realized.

Our architecture supports cuts along the two main criteria mentioned in Section 3.3.3, i.e. via administrative domains and technical domains. Administrative domains are characterized by having a dedicated SLA Manager, which is in charge of all the SLAs within that domain. Technical domains are characterized by having a dedicated Service Manager, which is in charge of all the artefacts needed for a (set of) service implementation(s).

The following figure shows an example of how such a domain cut can be realized for a single service provider organization that interacts with customers and 3rd parties. Basically, the service provider organization is split into 2 main administrative domains: one might be for application services, the other for infrastructure services. Furthermore, there is a split into 3 technical domains, each represented by a service manager. For example, one might be for application artefacts, one for middleware artefacts and the last one for infrastructure artefacts.

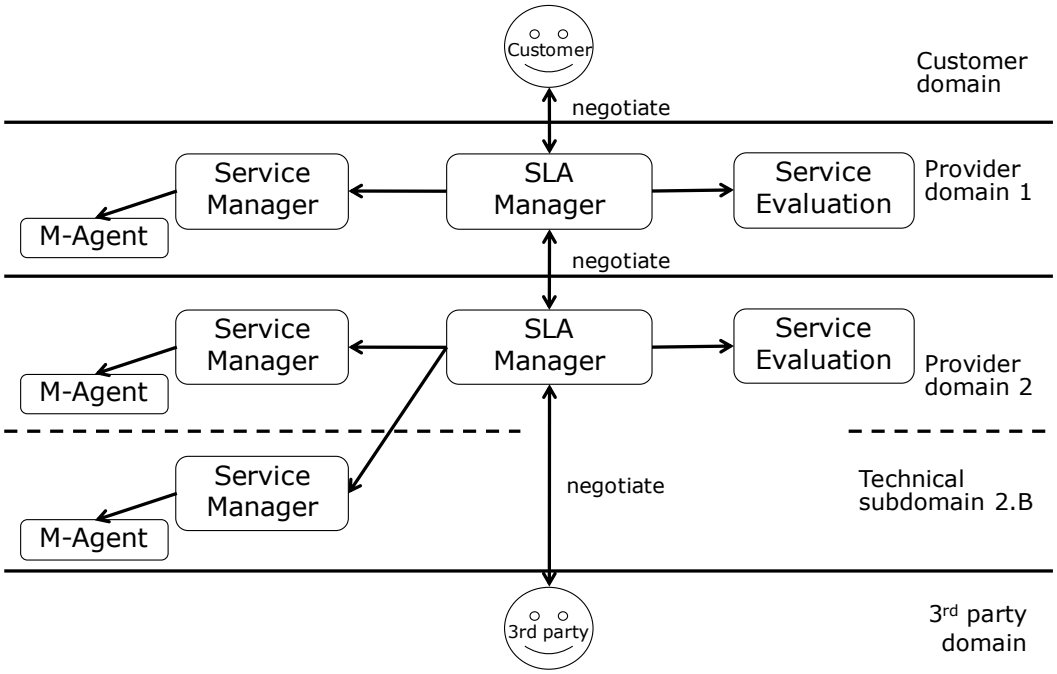


Figure 5: A possible domain split.

Other examples for domain splits are shown in Chapter 8, where we sketch the adoption architecture of the four main use cases pursued in our project.

Last, we want to show the relation between the introduced building blocks and the overall vision as stated in Section 2.1. The following figure shows exactly the envisioned interaction diagram and extends it by displaying where the concepts of Business Manager, SLA Manager, Service Manager and Manageability Agent apply.

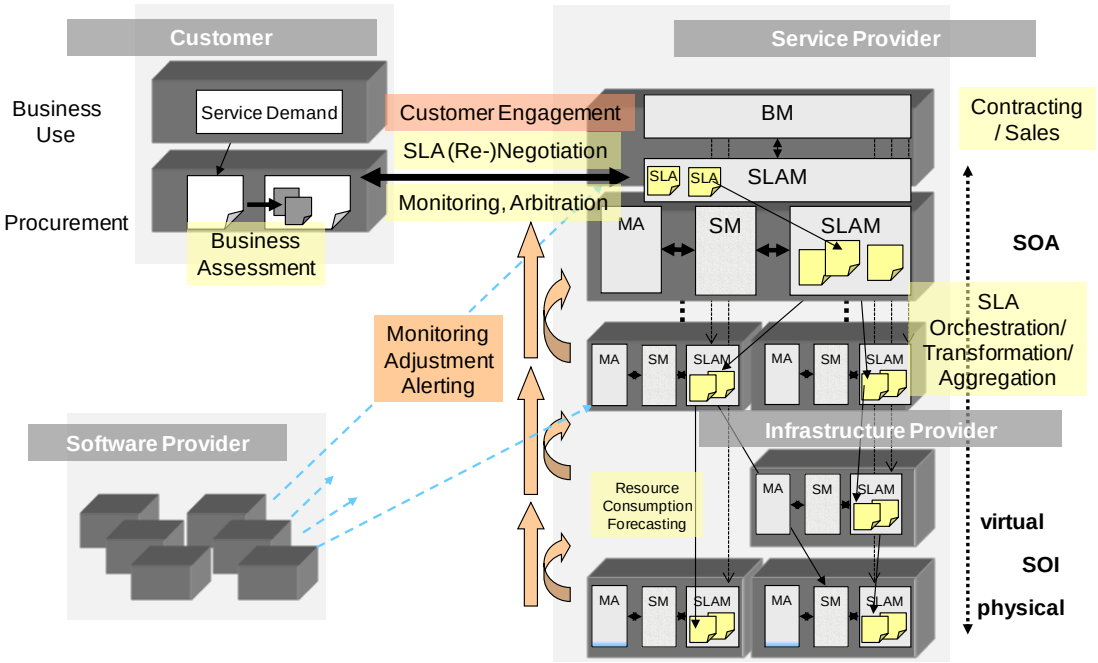


Figure 6: Building blocks in the envisioned interaction.

Note, that this figure does not show Service Evaluation components in order to keep it at reasonable complexity. Nevertheless, Service Evaluation components can be associated in here to each SLA Manager.

4 Architecture Specification

In this chapter, we present the top-level view of the SLA@SOI framework architecture, including its components and interactions.

After a first overview in Section 4.1, Section 4.1.2 provides the introduction to all top-level components. The role and purpose of the interaction stereotypes is explained in Section 5 in the context of broader interaction sequences. Last, this chapter provides details a cross-component aspects, namely the common Manageability approach (Sections 7).

4.1 Goals and Overview

The overall SLA@SOI Framework is conceived as a possibly distributed, hierarchical management system providing consistent SLA management across the service delivery stack.

At the highest level, we assume the operation of the SLA@SOI Framework serves ultimately to satisfy the goals of some business entity. Consequently, all management activities supported by the framework should eventually relate to the needs of the business entity.

Technically, the framework architecture is build along the following design goals:

1. Clear separation of concerns, e.g. having different components with clear responsibility on business, software, infrastructure, SLAs or 3rd party aspects.
2. Flexible configuration / setups, which can support a variety of scenarios and domains which
    a) require different setups of the framework, and
    b) already have (legacy) management functionality in place.
3. Simplicity, which is important to make the whole architecture understandable to a large audience and to make the actual framework adoptable for industrial use cases.

Design goals 1 and 2 are indispensable in order to support the different scenarios that are introduced by the different industrial use cases. Goal 3 is a more pragmatic goal but may in some cases also conflict with goal 2 as flexibility typically increases complexity. The main approach taken to resolve such conflicts is by providing default implementations of respective components or interaction channels.

In the remainder of the section, we first provide an overview of the architecture, its main components (Section 4.1.2) and interaction flows (Section 5). Finally, we elaborate the Manageability Fabric System (Section 7).

4.1.1 Top-level Architecture

In the following, we present the top-level view of the SLA@SOI framework architecture. It is derived from the general motivation and approach. For its representation, we chose a simplified version of UML component diagrams. Boxes represent components and connections represent stereotyped dependencies that translate to specific sets of provided and required interfaces.

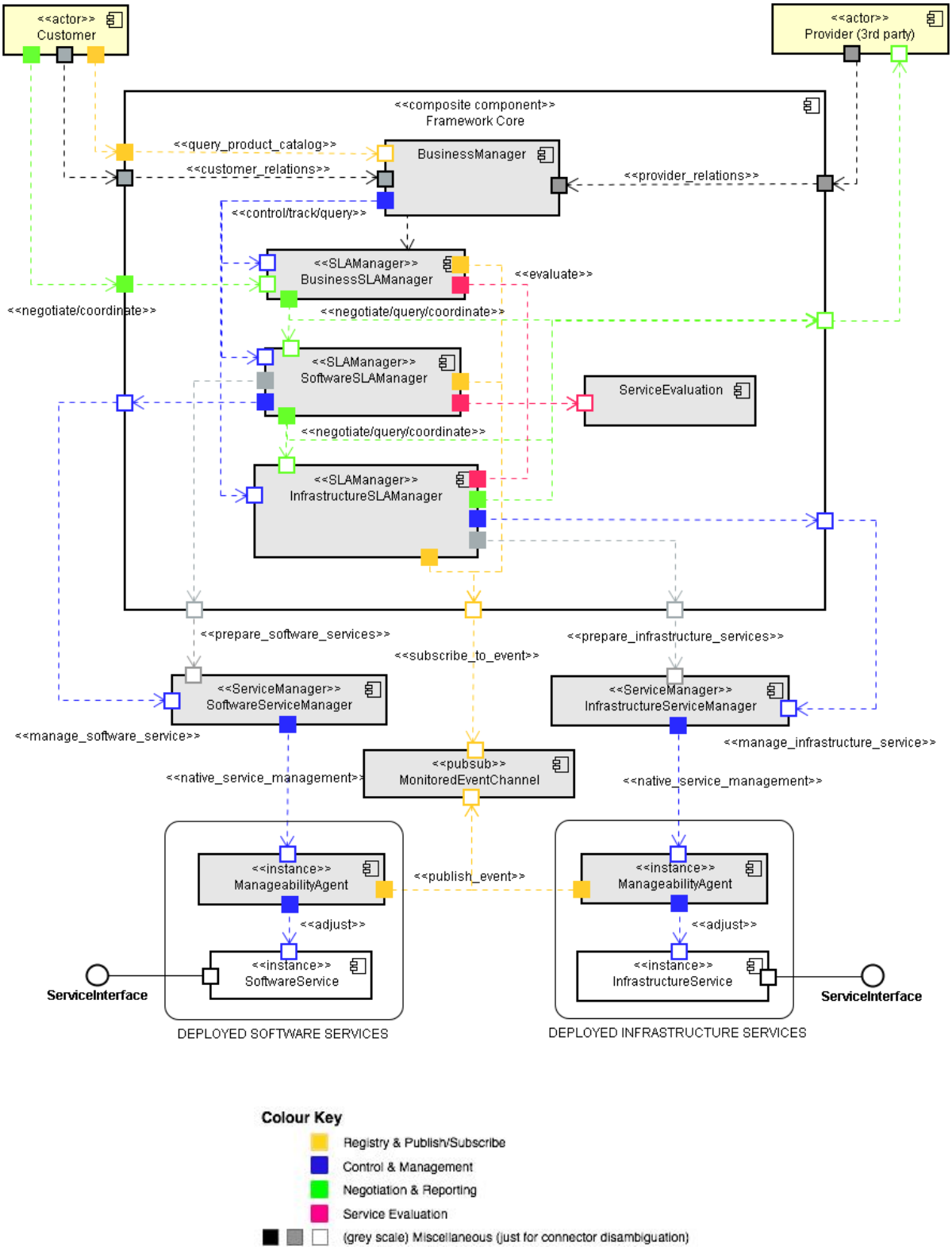


Figure 7: Top-level view of the SLA@SOI framework.

Figure 7 illustrates the main components of the SLA@SOI framework and their relationships. On the highest level, we distinguish between the core framework, Service Managers (infrastructure and software), deployed service instances with their Manageability Agents, and Monitoring Event Channels. The core framework

encapsulates all functionality related to SLA management. Infrastructure- and Software Service Managers contain all service specific functionality. The deployed service instance is the actual service delivered to the customer, and is managed by the framework via Manageability Agents. Monitoring Event Channels serve as a flexible communication infrastructure that allows the framework to collect information about the service instance's status. Further details are described in Section 5. In order to achieve a good generalization of the framework architecture, several components are realized as specialization of abstract components, namely SLA manager components and service manager components. The component hierarchy assumed for the top-level view is depicted in Figure 8.

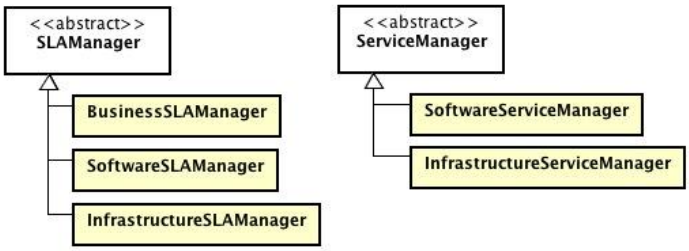


Figure 8: Component Specialisation Hierarchies.

Please note that Figure 7 shows only one possible instantiation of the framework architecture with its types of components and its possible interactions. Thus, where appropriate, there will be different instantiations of the framework architecture for each use case of the B-line. For example, use cases might choose another way to structure the SLA/service hierarchy, to have specific managers for non-IT SLAs/services, or to create managers specialized for IT areas such as BPEL.

4.1.2 Components and Interaction

In the following, we briefly describe the main components of our framework. Detailed specifications of the components can be found in the deliverables of the respective work packages.

The **Business Manager** is responsible for controlling all business information and communication with customers (<<customer_relations>>) and providers (<<provider_relations>>). For example, it realizes the customer relation management (CRM) necessary to efficiently sell the offered services. Furthermore, the Business Manager governs the (Business-, Software-, and Infrastructure-) SLA Managers (<<control/track/query>>). For this purpose, SLA Managers have to adhere to business rules defined by the Business Manager ('control') and have to inform the Business Manager about their current status and activities ('track').

The **(Business-, Software- & Infrastructure-) SLA Managers** are responsible for the management of all SLA related concerns. The Business SLA Manager, Software SLA Manager, and Infrastructure SLA Manager are specializations of an abstract generic SLA Manager (cf. Figure 8). SLA Managers are responsible for the negotiation of SLAs, and for the SLA Management of services (subject to SLAs). All SLA Managers can act as "service customers"; negotiating SLAs with other SLA Managers inside the *same* framework, or with *external* (3rd party) service providers (including other framework instances). As "service providers" all SLA Managers can negotiate SLAs with other SLA Managers in the *same* framework. Only the Business SLA Manager, however, can negotiate with

customers who are *external* to the framework. To avoid confusion, we refer to *external* customers as "business-customers", and use the term "product" to denote the (SLA governed) services offered by the framework to business-customers. Product descriptions are published in a 'product catalogue' (accessible via <<query_product_catalog>>) maintained by the Business Manager. Once an SLA has been agreed with a customer, it is the responsibility of the Business Manager to send reports on SLA status to the customer. The <<negotiate/query/coordinate>> relation captures all framework internal negotiation and querying interactions.. These negotiation part is equally used at business-level for the customer interaction (<<negotiate/coordinate>>) where business-level considerations (e.g. billing) are intercepted by the Business SLA Manager into the negotiation protocol. Finally, all SLA Managers can consult Service Evaluation to *a priori* evaluate the potential quality of a service (<<evaluate>>). This evaluation can be based on prediction, historical data, or predefined quality definitions, and supports the SLA Manager in finding service realisations with an appropriate quality.

Infrastructure- and Software Service Managers encapsulate all service-specific details. Both are specialisations of the abstract Service Manager concept (cf. Figure 8). Service Managers provide information about the service implementations supported, such as the service's realization and its dependencies with other services (<<prepare_infrastructure_services>>, <<prepare_software_services>>). SLA Managers provision services using the management functionality of their corresponding ServiceManager (<<manage_software_service>>, <<manage_infrastructure_service>>). Furthermore, Service Managers control the service instances they have provisioned. SLA Manager can manipulate service instances via generic management functions provided by the Service Manager.

The **Monitoring & Adjustment Management System** provides the underlying fabric across different layers (i.e. across software and infrastructure layer) supporting the monitoring and management of actual service instances. The Monitoring Event Channel is the basic component via which arbitrary monitoring events (e.g. SLO violations) can be propagated to relevant SLA managers. Access to this channel is realized via the <<publish_event>> and <<subscribe_to_event>> interaction stereotypes. Manageability Agents support the actual configuration and management of service instances. The access to manageability agents for SLA managers is always mediated via a specific service manager. Due to the domain specific nature the interactions between service managers and manageability agents is represented by the <<native_service_management>> stereotype which is not further refined by this architecture. It should be noted that Manageability Agents need not necessarily run within the same administrative domain as the service instance but importantly the sensors and effectors, which are part of the Manageability Agent must reside in the same administrative domain of the service instance and have access to their related Manageability Agent via some communication mechanism.

Due to the distributed nature of the overall monitoring & adjustment management system, the overall system is specified in an integrated way in Section 7.

Please note that Figure 7 shows only one possible setup of the architecture. The framework itself supports a much more flexible combination of different SLA Managers and Service Managers. In general, the following component cardinalities can be assumed per framework instance:

Component	Cardinality
SLA Manager	1..*
Service Manager	1..*
Service Evaluation	0..*
Business Manager	0..1

The minimal framework setup contains a single SLA Manager and its Service Manager. In such a scenario, no business functionality (such as CRM) is supported but the only focus lies on SLA negotiation, provisioning as well as monitoring and adjustment. Further setups are described in Section 8.

4.2 Foundation Models

Models play a central role in the SLA@SOI framework. Different parties use them to negotiate and plan services and their respective SLAs. They hold information needed for a priori quality evaluation, for runtime monitoring of SLAs, and for provisioning services.

The four most important models are

- the SLA model,
- the service construction model,
- the Open Cloud Computing Interface, and
- the Palladio Component Model.

4.2.1 SLA Model

The SLA & SLA Template Model, henceforth SLA(T) Model, captures all kinds of information needed for specifying a service level agreement. As such it contains data primitives (e.g. notions of time, capacity, ...) ground expressions (encapsulating constraints, events & functions), service descriptions: (encapsulating the functional properties of services, such as interfaces & operations), standard terms (e.g. availability, response time), actual SLA terms (e.g. on effective dates, involved parties), and business terms (for pricing, billing, etc.).

The detailed specification of the SLA model can be found in the SLA@SOI open source project [1] at http://sourceforge.net/apps/trac/sla-at-soi/wiki/SlaModel

A complete account of the SLA(T) modelling objectives & approach can be found in [20].

4.2.2 Service Construction Metamodel

The Service Construction Metamodel is an integral part for the planning and evaluation of services. It is motivated by the need to store and manage information about services inside the SLA@SOI framework. ServiceManagers have to provide data about the types of services offered, about alternative realizations of these service types, and about the service instances that have already been provisioned. Furthermore, SLAManagers require information about the dependencies of a service on other services, about features of the service itself, and about features of its associated monitoring system. Based on this information, SLAManagers can plan and negotiate SLAs with their customers and acquire other (external) services that are required.

The detailed specification of service construction model can be found in the SLA@SOI open source project [1] at http://sourceforge.net/apps/trac/sla-at-soi/wiki/ServiceConstructionModel.

A more general introduction can be found in [2].

4.2.3 Open Cloud Computing Interface

The Open Cloud Computing Interface (OCCI) is an open protocol for infrastructure service in a cloud-computing environment. The OCCI working group an open initiative and is always hugely enthusiastic and very supportive to people and groups of all types wishing to get involved with OCCI, whether that is through specification contributions or new implementations of it. OCCI is a REST (Representational State Transfer) based protocol and hence it adopts a resource-oriented architecture (ROA). Each resource i.e. a compute node is identified by URL(s) and has one or more representations. The specification, for reasons of modularity and extensibility, is divided into 3 areas: core model, rendering-specific and extension (domain-specific) models. At the heart of OCCI protocol is a core model, which defines the essential OCCI entities and their relationships. The core model is designed with extensibility in mind to manage other types of cloud resources in the future like storage, networks, application etc. For now, the core model is extended by the Infrastructure model and is used to described IaaS-type resources (Compute, Network & Storage) and their inter-relations.

An interface, the query interface, exposes "kinds" (category of resources distinguished by some common characteristic or quality) which have "attributes" and on which actions can be performed. The attributes are exposed as key-value pairs and applicable actions as links. As well as exposing kinds, the query interface also exposes "mixins". Mixins can be supplied when creating a resource (also at runtime to existing resources) in order to add further functionality to a particular type of resource Kind. An example of this is with the Network resource in the Infrastructure specification. As specified, this resource represents a L2 networking resource. If a client wishes to add L3/L4 functionality in the form of TCP/IP then the IPNetwork mixin should be uses to specify this additional requirement.

The detailed specification of the OCCI model can be found in the SLA@SOI open source project [1] at http://www.occi-wg.org and an implementation of the specification, from SLA@SOI, can be found at https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/trunk/infrastructure-servicemanager/ism

Further details on this model are described in [19].

4.2.4 Palladio Component Model

The PCM is a metamodel and toolset for software quality analysis. Its focus lies on software performance evaluation and reliability prediction. As such, it follows the paradigm of model-driven performance (and reliability) engineering. Software architects specify their system in a language specific to their domain, i.e. component diagrams, deployment diagrams, and activity diagrams. These diagrams are annotated with information necessary for performance and reliability prediction. For quality prediction, the architectural model can be transformed into a discrete-event simulation or analytical models, such as Layered Queueing Networks (LQNs), stochastic process algebras, or in case of reliability prediction discrete time Markov chains.

The detailed specification of the PCM model can be found on the Palladio Component Model (PCM) project page at ***http://sdqweb.ipd.kit.edu/wiki/Palladio_Component_Model***

Further details on this model are described in [21].

4.3 Components

4.3.1 Generic SLA Manager

The detailed specification of the GenericSlaManager component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/wiki/GenericSlaManager

4.3.2 Business SLA Manager

The BusinessSLAManager concentrates all the business management activities. It realizes the <<negotiate/coordinate>> interaction with business customer and <<control>> interaction with the BusinessManager. Also it realizes <<negotiate/coordinate>> interaction with other lower layers or external providers.

The detailed specification of the BusinessSLAManager component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/wiki/BusinessSlaManager

4.3.3 Business Manager

The BusinessManager concentrates all the customer and provider management activities and the business control of the framework. At business level controls the different SLAManagers in terms of negotiation rules and violations generated on each layer. It realizes the <<query product catalogue>> interaction with business customer, <<customer relations>> and <<provider relations>> interactions. It realizes the <<track >> interaction with the SLAManagers.

The detailed specification of the BusinessManager component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/wiki/BusinessManager

4.3.4 Software SLA Manager

The Software SLA Manager extends Generic SLA Manager with concrete implementations of Planning and Optimization and Provisioning and Adjustment components. Its prime functionality concentrates on dealing with software services for which it consumes <<prepare_software_services>> and <<manage software service>> interaction provided by SoftwareServiceManager and <<evaluate>> interaction provided by ServiceEvaluation component.

The detailed specification of the Software SLA Manager component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/wiki/SoftwareSlaManager

4.3.5 Software Service Manager

The SoftwareServiceManager encapsulates all management activities specific to software services. It realizes the <<prepare_software_services>> and <<manage_software_services>> interactions.

The detailed specification of the SoftwareServiceManager component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/wiki/SoftwareServiceManager

4.3.6 Infrastructure SLA Manager

The detailed specification of the InfrastructureSlaManager component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/wiki/InfrastructureSlaManager

4.3.7 Infrastructure Service Manager

The detailed specification of the InfrastructureServiceManager component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/wiki/InfrastructureServiceManager

4.3.8 Service Evaluation

The detailed specification of the ServiceEvaluation component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/wiki/SoftwareServiceEvaluation

4.3.9 Manageability Agent

The detailed specification of the ManageabilityAgent component can be found in the SLA@SOI open source project [1] at

http://sourceforge.net/apps/trac/sla-at-soi/

5 Interaction Stereotypes

This section details the interaction stereotypes introduced in the top-level architecture (cf. Figure 7). For each stereotype the involved component interfaces are specified as well as some considerations on the expected interaction flow.

5.1 <<customer_relations>>

The interaction **<<customer_relations>>** realizes the customer relationship management with the BusinessManager component of the framework for necessary to efficiently sell the offered services. It offers different functionality to the business customer. The functionality allows the customer to be registered into the framework, modify this registration information, authentication against the framework once he has been registered (to buy some product and also to retrieve report of the behaviour of the product and corresponding SLAs).

Involved Components

- BusinessManager
- Customer

Overview

The relation **<<customer_relations>>** subsumes functionality that is necessary for a customer in order to be able to interact with the framework to consume the products and services.

Interfaces and Operations

<<customer_relations>> subsumes functionality that is necessary for a customer in order to be able to interact with the framework to consume the products and services. This relation is comprised of the following interfaces: ICustomerRelations and IReport.

ICustomerRelations

- Customer registration, allows the business customer to be register in to the framework in order to be able to interact with it. This allows the customer to authenticate against the framework, once he has been registered
- Customer can modify his registration details
- Customer can authenticate in the framework

Overview of Interaction Sequence

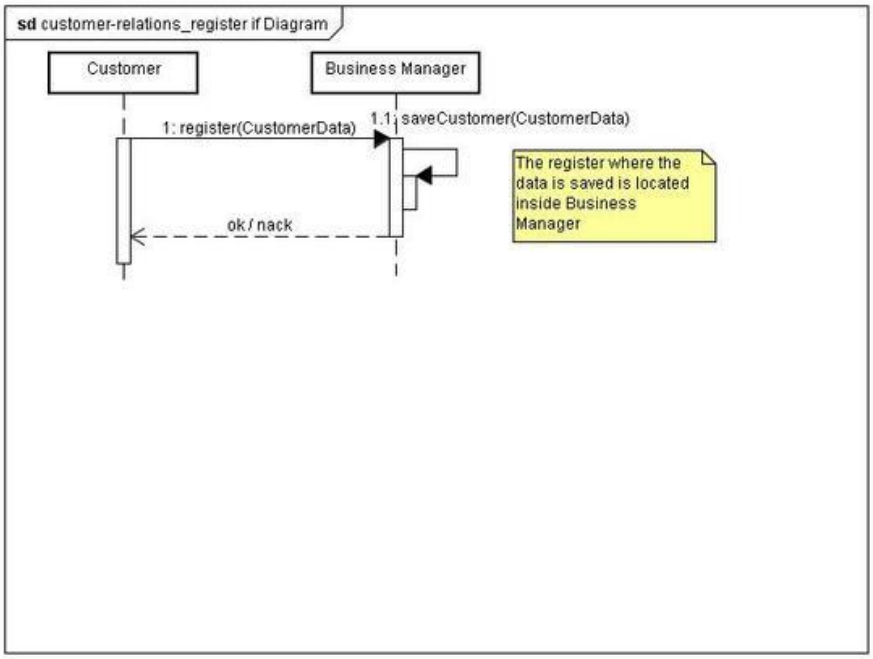


Figure 9: << customer_relations :ICustomerRelations>> sequence diagram

These interactions are described as following:

- The first interaction depicts the way to be register into the framework. The customer requests to the BusinessManager to be registered into the framework given his information. The BusinessManager will do the necessary actions to check if it is possible to do it and finally will save his information in a register inside the BusinessManager.
- Finally, the BusinessManager confirms the customer registration.

IReporting

- Customer can obtain a report about a specific SLA that he signed in the consumption of a service. Customer has to interact with the BusinessManager to obtain this reports.
- Customer can receive periodic reports sent by the BusinessManager about the SLA status and information (for instance a violation occurs). Customer has to implement this interface in order he will be able to see these reports.

Overview of Interaction Sequence

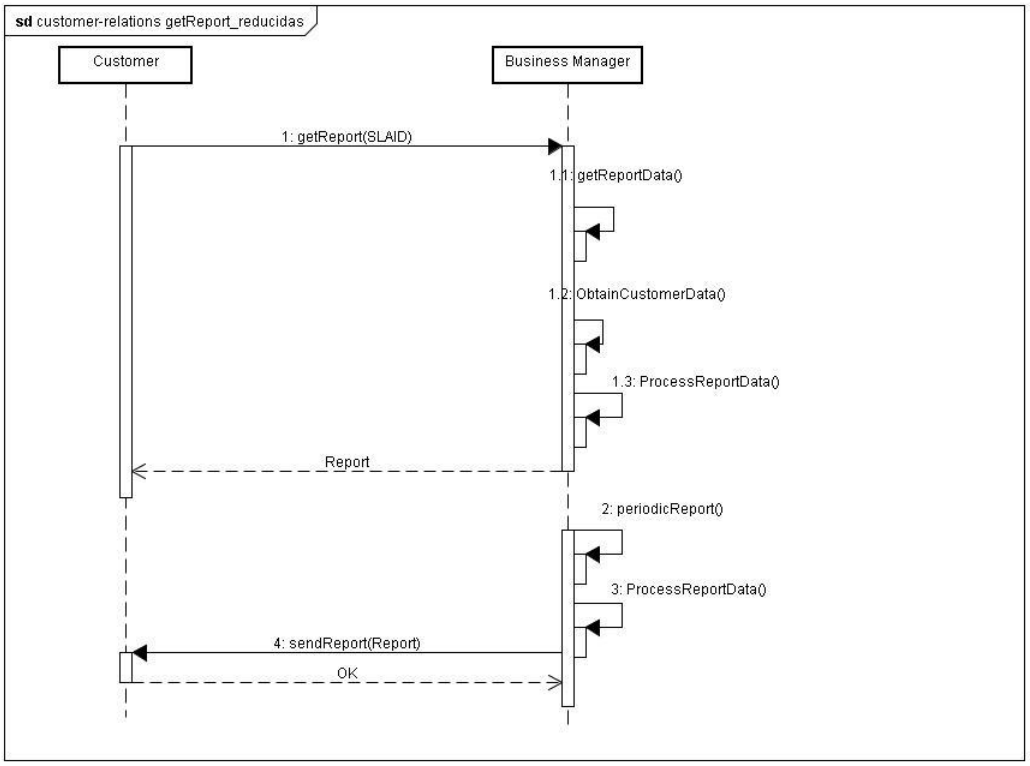


Figure 10: << customer_relations: IReporting>> sequence diagram

These interactions are described as following:

- The first interaction shows the way to obtain the reports related to one SLA. The customer queries the BusinessManager with the identification of the SLA he is interested on get a report. Then the BusinessManager will collect all the necessary information and build the report. Finally, the report will be sent to the customer.
- The second interaction shows the way to send the reports related to one SLA. Periodically, the BusinessManager could have to send reports about a SLA. When it is time to do it, the BusinessManager will collect all the

necessary information and build the report. Finally, the report will be sent to the customer.

Detailed Specification

The detailed specification of the listed interfaces and the involved datatypes can be found in the SLA@SOI open source project [1] under the specification of the BusinessManager component: http://sourceforge.net/apps/trac/sla-at-soi/wiki/BusinessManager

5.2 <<provider_relations>>

This interaction allows the provider to register with the framework, and to modify their registered information. This allows the provider to authenticate against the framework in order to interact with it, once he has been registered provider is able to execute available operations.

Involved Components

- BusinessManager
- Provider

Overview

The relation **<<provider_relations>>** subsumes functionality that is necessary for a provider in order to be able to interact with the framework to publish the services.

Interfaces and Operations

<<provider_relations>> subsumes functionality that is necessary for a provider in order to be able to interact with the framework to publish the services. This relation is comprised of the following interfaces.

IProviderRelations

- Provider registration, allows the provider to be register in to the framework in order to be able to interact with it. This allows the provider to authenticate against the framework, once he has been registered
- Provider can modify his registration details
- Provider can authenticate in the framework

Overview of Interaction Sequence

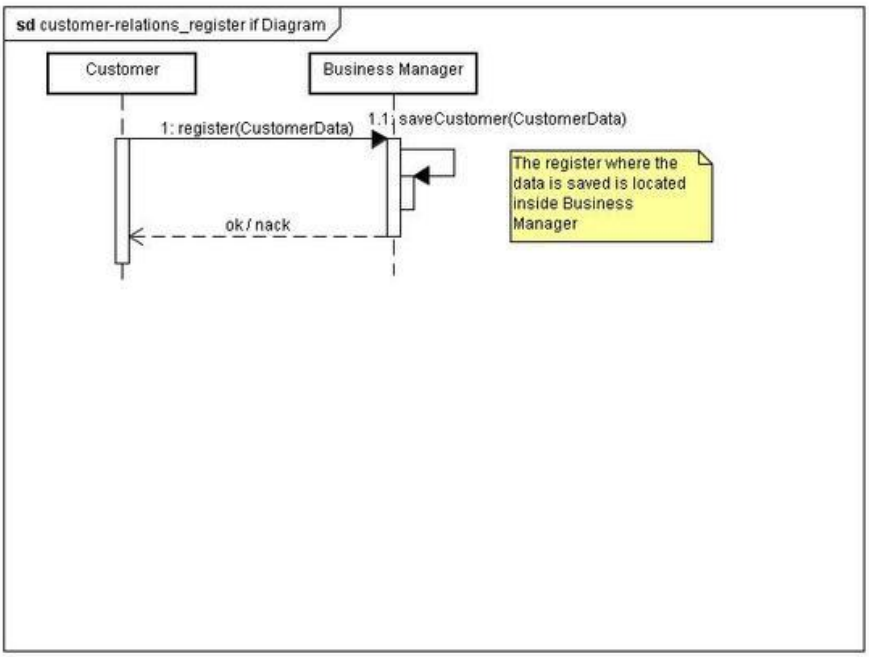


Figure 11: << IProviderRelations>> sequence diagram

These interactions are described as following:

- The first interaction depicts the way to be register into the framework. The provider requests to the BusinessManager to be registered into the framework given his information. The BusinessManager will do the necessary actions to check if it is possible to do it and finally will save his information in a register inside the Business Manager.
- Finally, the BusinessManager confirms the provider registration.

Detailed Specification

The detailed specification of the listed interfaces and the involved datatypes can be found in the SLA@SOI open source project [1] under the specification of the BusinessManager component: http://sourceforge.net/apps/trac/sla-at-soi/wiki/BusinessManager

5.3 <<query_product_catalog>>

This interaction allows business customers to query the accessible product catalogue in order to search products. Customers define the characteristics of the products they want and with this interaction this information arrives. Furthermore, this interface provides access to the business templates, in order to build a SLAOffer to achieve an agreement (SLA) for the consumption of services. The business templates are related to the one product obtained by the query to the product catalogue and match with the characteristic desired by the customer.

Involved Components

- BusinessManager
- Customer

Overview

The relation **<<query_product_catalog>>** subsumes functionality that is necessary for a customer in order to be able to recover the products published in the framework.

Interfaces and Operations

<< query_product_catalog >> subsumes functionality that is necessary for a customer in order to recover the products published in the framework and the business SLA templates associated to each product. This relation is comprised with the following interface.

IQueriableProductCatalog

GetProducts

- The functionality allows the customer to access the product catalogue and find products that fulfil his criteria.
- It is possible customer provide a key-word/category search for searching products.

GetTemplates

- The functionality enables the customer to obtain the templates associated to a product from the catalogue.

Overview of Interaction Sequence

GetProducts

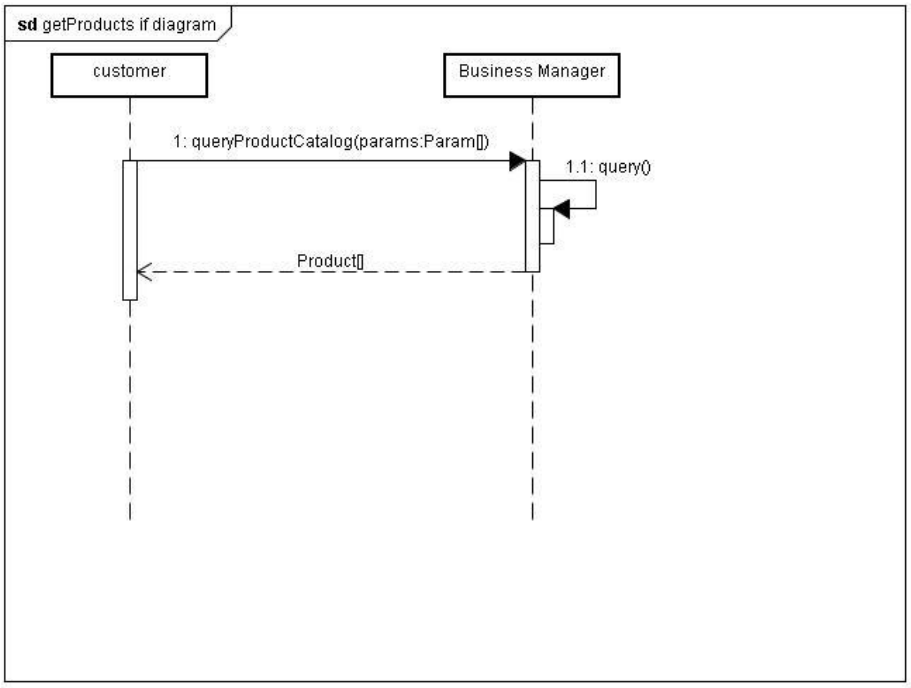


Figure 12: << IQueriableProductCatalogue: getProducts>> sequence diagram

These interactions are described as following:

- The customer queries the Product Catalogue giving some criteria over the characteristic of the products he is interested in.
- The framework returns all the products that match with the given criteria.

GetTemplates

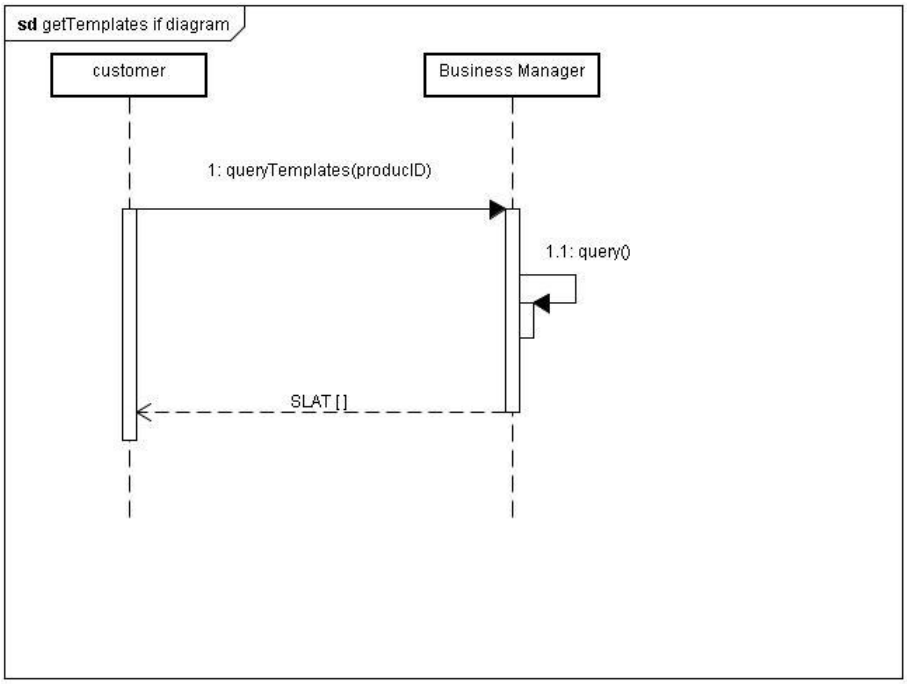


Figure 13: << IQueriableProductCatalogue: getTemplates >> sequence diagram

These interactions are described as following:

- The customer queries the Product Catalogue giving the product he wants and he is interested in.
- The framework returns the business SLA template associated to the given product.

Detailed Specification

The detailed specification of the listed interfaces and the involved datatypes can be found in the SLA@SOI open source project [1] under the specification of the BusinessManager component: http://sourceforge.net/apps/trac/sla-at-soi/wiki/BusinessManager

5.4 <<negotiate/renegotiate >>

This interaction <<negotiate/renegotiate>> allows service customers and business customers to negotiate with each other over a target service. If the negotiation succeeds, an SLA is created.

Overview of Interfaces and Operations

The interfaces for <<negotiate>> and <<renegotiate>> are exposed to external parties through the SyntaxConverter which acts as a proxy to the ProtocolEngine.

The internal parties also have access to <<renegotiate>> directly on the ProtocolEngine.

ProtocolEngine interface

The ProtocolEngine provides the following operations:

- start negotiation: request to start a negotiation session.
- make an offer: submit an offer or counter-offer to the other party.
- request agreement creation: request the other party to accept the submitted offer and return the SLA.
- start renegotiation: request to start renegotiation over an already created SLA.
- cancel negotiation: cancel ongoing negotiation.
- terminate: end the service level agreement with the other party.

Overview of Interaction Sequence

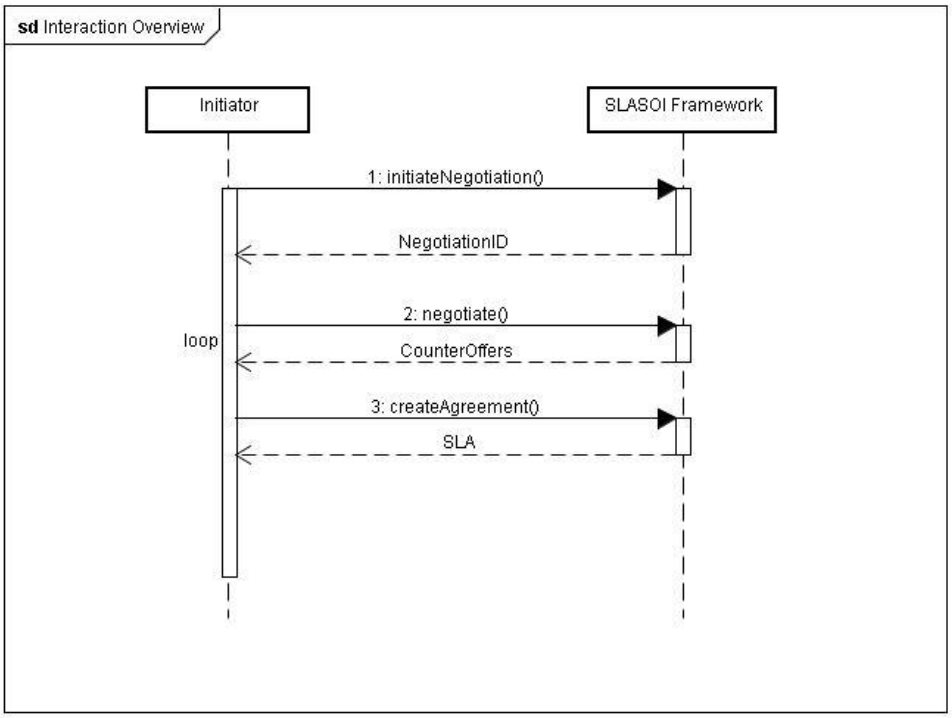


Figure 14: Interaction Overview

The above (simplified) sequence diagram gives an overview of the interaction that the initiator will make with the SLASOI framework in general. The initiator starts negotiation, conducts multiple rounds of offer and counter-offer exchanges and at some point, concludes the negotiation by requesting to create an agreement. If successful, the SLA comes into existence.

Detailed Specification

Negotiation takes place between two kinds of parties: a customer and a provider. Negotiation starts formally when the Customer invokes initiate operation on Provider. A negotiation session is established per negotiation. If the negotiation succeeds, an SLA is created as a result. For year 2, we consider negotiation between 1 Customer and 1 Provider. Both parties use a common protocol that describes a negotiation model as a state machine, the sequence of messaging and transitioning between states.The protocol is orchesterated by the ProtocolEngine, which exposes its interface through the SyntaxConverter. Since SyntaxConverter just acts as a proxy to ProtocolEngine, the operations for both are the same.

Out of various possible interaction scenarios, the most common ones are expressed by the sequence diagrams below:

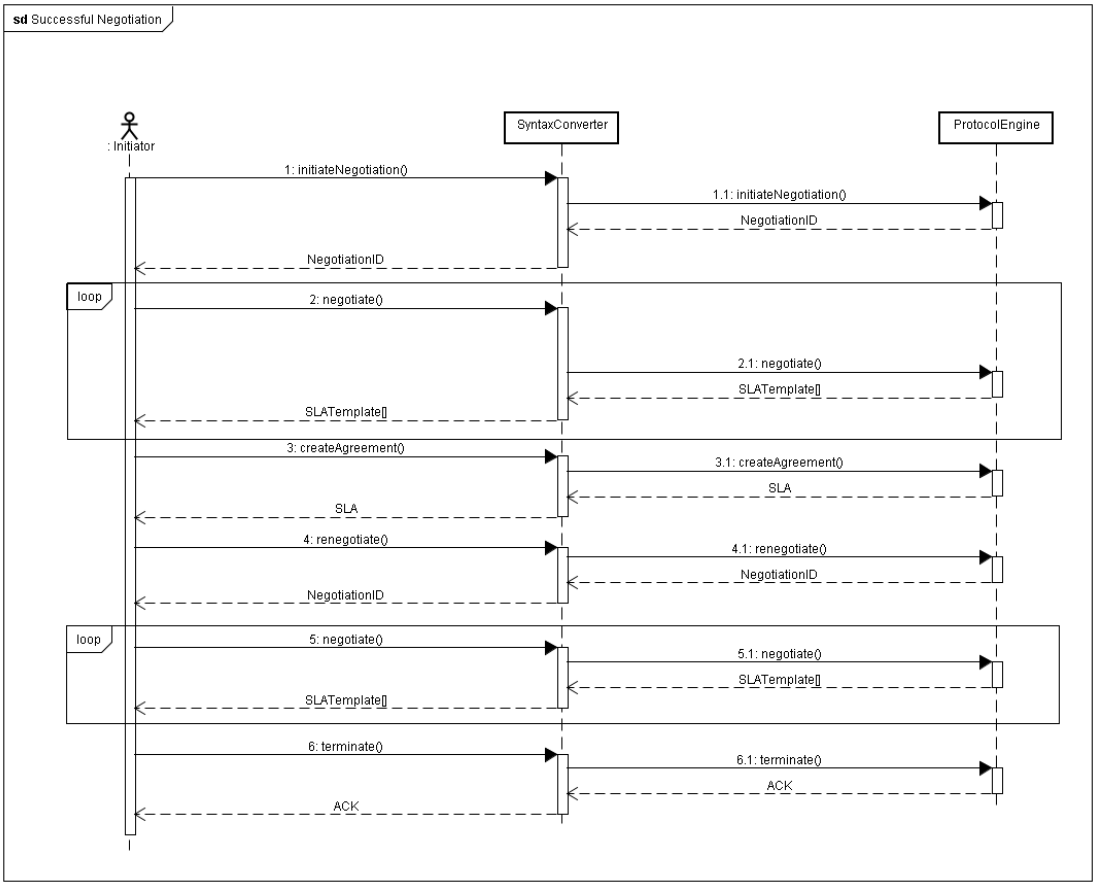


Figure 15: Interaction showing successful negotiation, renegotiation & termination of SLA

Above interaction demonstrates probably the most common interaction scenario. 1) A customer initiates a negotiation session with a provider. The provider produces a negotiation identifier and returns back to the customer who then uses it for any further negotiation that follows. 2) In this example, negotiation involves multiple steps (shown by the loop). In each step, the customer submits an offer through the negotiate operation and the provider returns back counter-offers. 3) At some time, the customer invokes createAgreement operation accompanying an offer which he would like to be accepted as such. The provider accepts it and returns back an SLA.

4) It happens that the customer at some stage realizes that he may need to renegotiate the SLA (e.g., due to increase or decrease in demand for service). The provider creates a new negotiation session, gives it an identifier and returns it back to the customer for further correspondence. 5) The customer then starts (re)negotiating which could also be multistep (as was negotiation). 6) The parties may or may not reach an agreement. Here, the customer decides (for whatever reason) to terminate the SLA. The provider obliges and takes care of the implications.

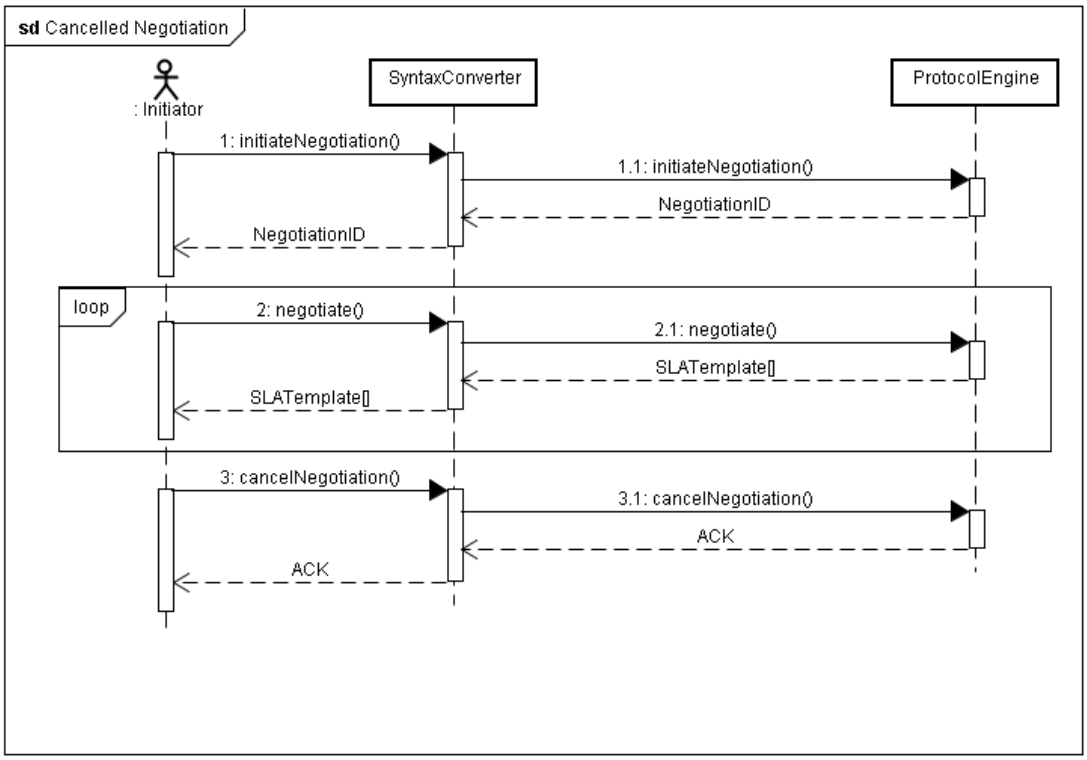


Figure 16: Interaction showing negotiation & cancelling negotiation

In above interaction, 1) A customer initiates a negotiaiton with a provider. 2) Multi-round negotiation follows but due to some reason, the customer decides to cancel negotiation. 3) For this, the customer invokes the cancelNegotiation operation and the provider obliges.

The detailed specification of the listed interfaces and the involved datatypes can be found in the SLA@SOI open source project [1] under the specification of the ProtocolEngine component: http://sourceforge.net/apps/trac/sla-at-soi/wiki/ProtocolEngine

5.5 <<control/track/query>>

This interaction is divided in two parts: control/track interaction and query. The relation **<<control/track>>** give the method for SLAManagers to adhere to business rules defined by the BusinessManager ('control') and to inform the BusinessManager about their current status and activities ('track').

Involved Components

- BusinessManager
- GenericSLAManager(BSLAM/SSLAM/ISLAM)

Overview

The BusinessManager controls all business information and communication over the whole architecture. So it is necessary a way to govern the negotiation used in the SLAManagers (Business-, Software-, and Infrastructure-).

The Track interface is implemented by the BusinessManager in order to receive violations from the SLAManagers.

Interfaces and Operations

<< control/track >> subsumes functionality that is necessary for the business manager to control the negotiation protocol, and monitors the violations of the SLAManagers. This relation is comprised with the following interfaces: Control, and Track.

Control

The interaction <<control>> allows the SLA managers to require and send control information and ask for action to take during the complete lifecycle of the SLA.

Overview of Interaction Sequence

These interactions are described as following:

- The Control interface is implemented by the SLA Managers and provides the ability to configure and retrieve the policies from SLA Managers. It is needed a method for BusinessManager to know about the policies that Business SLA Manager is applying and to communicate them the new ones.
- The kind of Policies that can be configured are: Adjustment policies, Planning and negotiation policies.

Track

The interaction <<track>> allow the SLA managers to inform to the Business Manager about actions, violations and required monitoring information in order to evaluate the behaviour of the services contracted.

Overview of Interaction Sequence

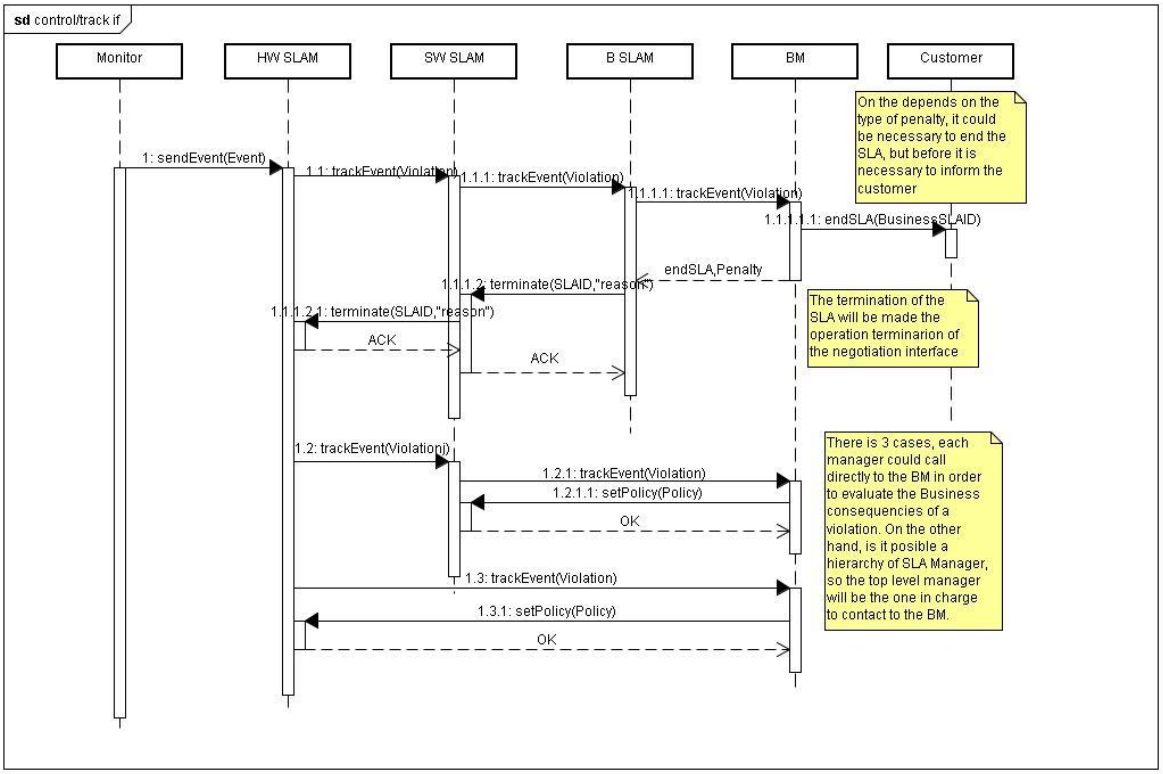


Figure 17: << control/track/query: Track >> sequence diagram

These interactions are described as following:

- Receive Monitoring and Violation information
- Inform to the Business Manager about the actions executed by the SLA Managers and the violations generated in order to take it into account for the penalties calculation at Business Manager.
- Authorization of customers: During negotiation process it is needed to know if the customer has the corresponding rights to do it.

Detailed Specification

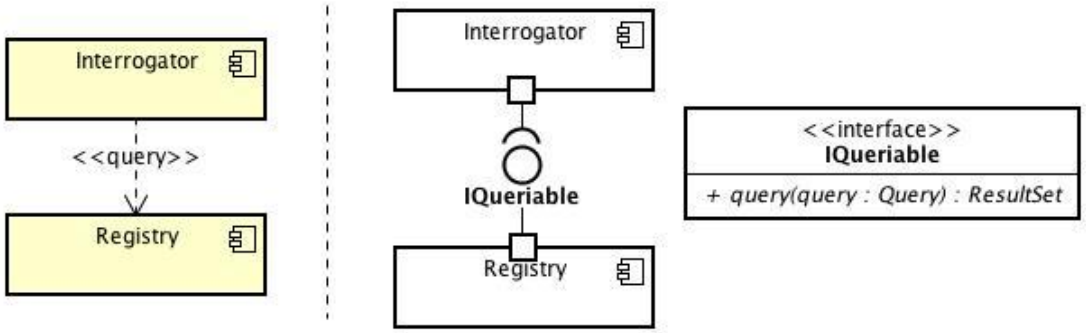
The detailed specification of the listed interfaces and the involved datatypes can be found in the SLA@SOI open source project [1] under the specification of the BusinessManager component: http://sourceforge.net/apps/trac/sla-at-soi/wiki/BusinessManager

5.6 <<negotiate/query/coordinate>>

The negotiate interaction provides an interface for generic negotiation capabilities. It is realized in the subtype <<negotiate>> as implemented by the ProtocolEngine.

The <<query>> interaction is a high-level interaction covering all forms of database/registry querying and encapsulating the posting of queries and receipt of query results. In the context of <<negotiate/query/coordinate>>, the <<query>> interaction permits BusinessSLAManager instances to access

information on the service offerings of third parties. In the case that the third party is an instance of the SLA@SOI framework, then the query operates over that framework's business product catalogue (via <<query_product_catalog>>). Otherwise, it is assumed that service offering information is exposed in the form of SLA Templates, available from an SLA Template Registry (SLATemplateRegistry::<<query>>).



The <<coordinate>> interaction is used so that the customer can initiate the provisioning of a service, according to a previously established SLA. Although it may be possible that this invocation is avoided, and some other mechanism is used (e.g. using lead times for service initialization), we consider this method to be the most reliable and as such it is the recommended one.

The coordination of provisioning is handled by the ProtocolEngine, in particular by the provision(UUID slaId) method exposed by the ProtocolEngine interface. To avoid repetition of content, this interaction is specified along with the interactions supporting SLA negotiation at ProtocolEngine::<<negotiate>>.

5.7 <<prepare software services>>

The relationship **<<prepare_software_services>>** allows SoftwareSLAManagers to search for service implementations provided by a SoftwareServiceManager and retrieve information about their dependencies. Furthermore, it provides a unified access point to the provisioning operation of software services including reservation and booking. The provisioning and management of a service is done via the **<<manage_software_services>>** relationship.

Involved Components

- SoftwareSLAManager
- SoftwareServiceManager

Overview

The relation **<<prepare_software_services>>** subsumes functionality that is necessary to find implementations for a particular service type as well as retrieving information about a service implementation or service instance.

Interfaces and Operations

<<prepare_software_services>> subsumes functionality that is necessary to find implementations for a particular service type as well as retrieving information

about a service implementation or service instance. This relation is coprised of the following interface.

IPrepareSoftwareServices

- Get all available implementations of a particular service type. The implementation descriptors include information about the service dependencies, monitorability, configuration options, and provisioning information such as lead times.
- Create a ServiceBuilder that allows the SLAManager to configure the service instance.
- If necessary, check for the availability of (limited) resources such as software licenses and make a time restricted reservation of these. Inform the caller about the success or failure of the reservation.
- If necessary, book/confirm a previously successful reservation and acquire all necessary resources for the specified time. Inform the caller about the success or failure of the booking

Overview of Interaction Sequence

An overview on the interaction sequence is given in Figure 18.

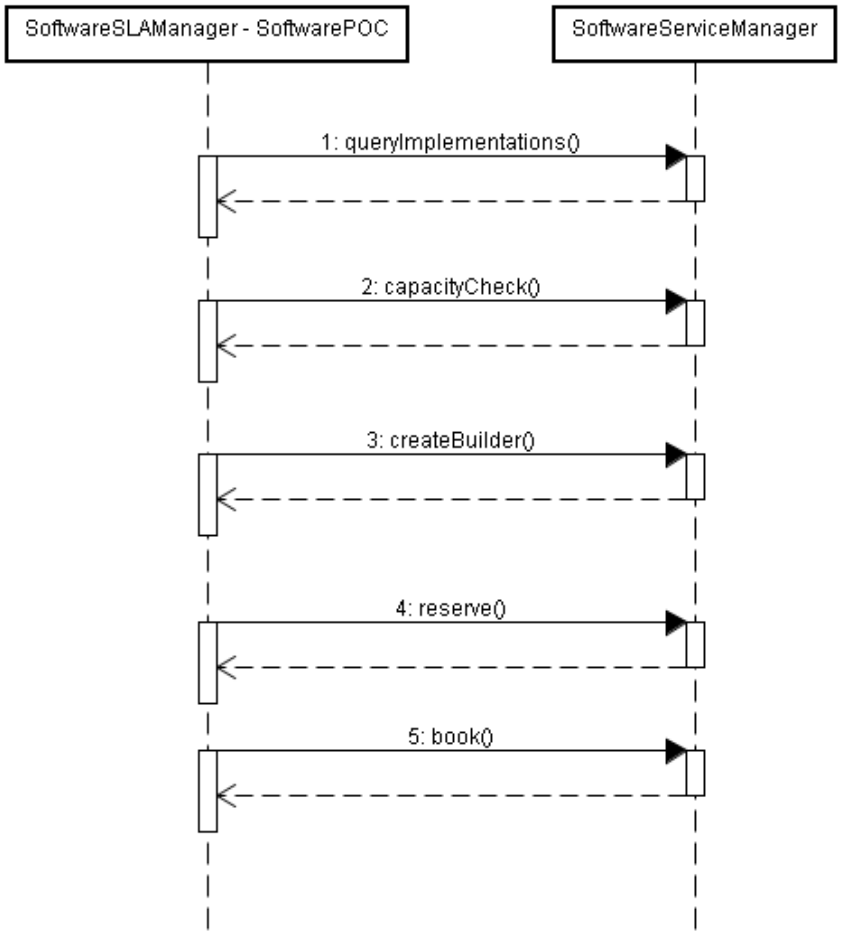


Figure 18: <<Prepare_Software_Services>> sequence diagram

These interactions are described as following:

- The SoftwarePOC queires for the available service implementations for a particular service type. SoftwareServiceManager returns an array of the service implementation. The SoftwarePOC selects the appropriate ServiceImplementation.
- SoftwarePOC invokes capacityCheck to determine the availability of software resources required for the realization of the service provisioning process
- SoftwarePOC based on the results of capacityCheck requests SoftwareServiceManager to create ServiceBuilder instance for the specific service implementation. This service builder instance will then be used by the SoftwarePOC to configure the service instance
- SoftwarePOC reserves the required resources for materialization of the SLA. The reservation is time bound and times out automatically unless specifically confirmed by the SoftwarePOC
- Finally, the SoftwarePOC confirms the previously reserved resources to be booked for later usage during the service provisioning process.

Detailed Specification

The detailed specification of the listed interfaces and the involved datatypes can be found in the SLA@SOI open source project [1] under the specification of the SoftwareServiceManager component: http://sourceforge.net/apps/trac/sla-at-soi/wiki/SoftwareServiceManager

5.8 <<manage_software_service>>

The relationship **<<manage_software_services>>** allows SoftwareSLAManagers to create new service instances and carry out service lifecycle management related operations. This relation complements the functionality enabled through the relation <<prepare_software_service>>

Involved Components:

- SoftwareSLAManager
- SoftwareServiceManager

Overview

The relationship <<manage_software_services>> enabled SoftwareSLAManager to interact with SoftwareServiceManager to create new software service instances. Additionally, SoftwareSLAManager can execute service lifecycle management related operations through this relationship.

Overview of Interfaces and Operations

The relationship <<manage_software_services>> enabled SoftwareSLAManager to interact with SoftwareServiceManager to create new software service instances. Additionally, SoftwareSLAManager can execute service lifecycle management related operations through this relationship.

IManageSoftwareServices

The interface IManagerSoftwareservices subsumes all functionality related to the provisioning and management of a software service.

- trigger the provisioning process that executes the necessary actions to provide access to a booked service instance
- release and cleanup of a provisioned service instance. Disables access to a provisioned service instance
- change the implementation or dependencies of an instantiated service
- (re-)configuration of the service
- (re-)configuration of the monitoring system
- query possible change actions

Overview of Interaction Sequence

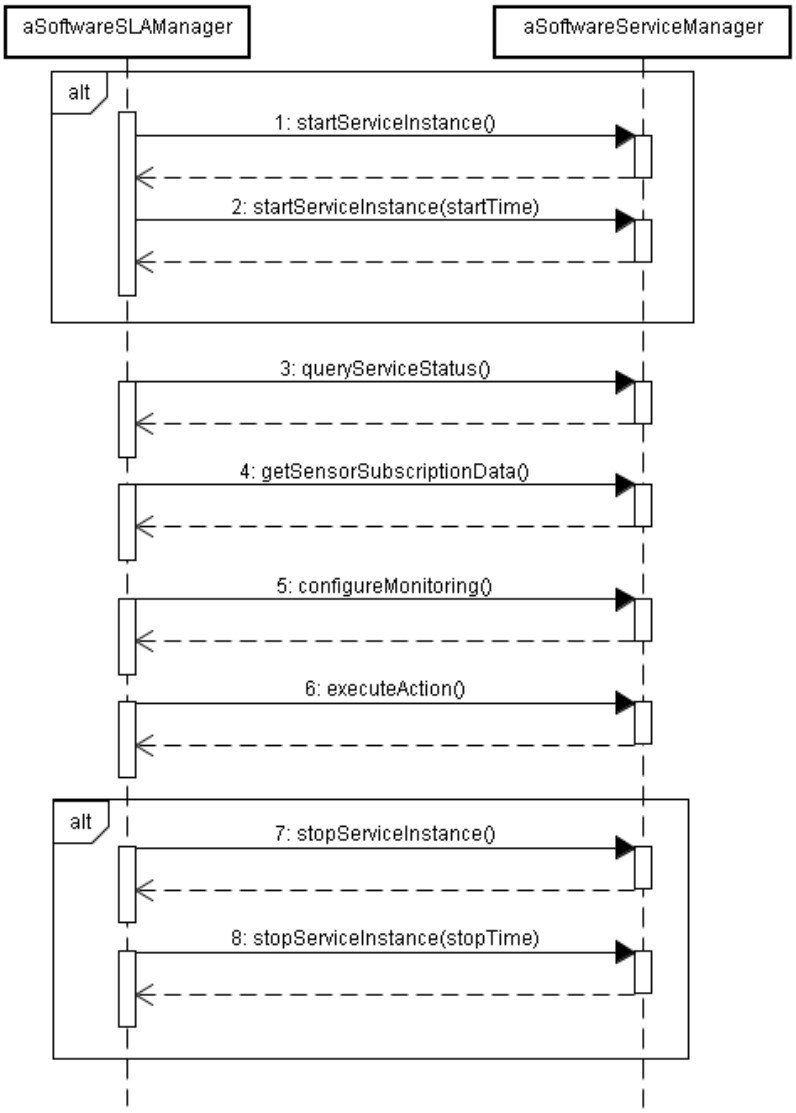


Figure 19: Managed_Software_Services interaction sequence

The summarized description of the interaction sequence is given below:

1. After SLAs have been established with the customer, SoftwareSLAManager invokes the startServiceInstance operation to trigger the software service provisioning. The operation invocation can take one of two forms: 1) either provisioning should be triggered immediately or 2) provisioning is scheduled to be triggered at a later time. In this case a start time is provided as an argument to startServiceInstance operation.
2. Once the service is provisioned and operational, SoftwareSLAManager will invoke queryServiceStatus to find out the status of the service instnace.
3. If the system is to be monitored the operation getSensorSubscriptionData provides information what information can be monitored (i.e. which sensors are avaliable) and where monitoring events are published.
4. If the monitoring system is to be adjusted, this can be done via the method configureMonitoring.

5. In case of some potential SLA violation, some management operations need to be performed on the service instance. Depending on the required operation to bring service instance in compliance to SLAs, SoftwareSLAManager invoked the executeAction operation which will perform the operation provided as an argument.
6. At the end of the service lifecycle, SoftwareSLAManager will use the stopServiceInstance to trigger the unprovisioning of the software services. Again, this operation can take two forms and the appropriate operation is invoked by the SoftwareSLAManager

Detailed Specification

The detailed specification of the listed interfaces and the involved datatypes can be found in the SLA@SOI open source project [1] under the specification of the SoftwareServiceManager component: http://sourceforge.net/apps/trac/sla-at-soi/wiki/SoftwareServiceManager

5.9 <<prepare_infrastructure_services>>

Overview

This stereotype provides a single interface. It allows potential infrastructure solutions to be queried, and reservations for specifc infrastructure to be made. Reservations can also be released.

Overview of Interfaces and Operations

The <<preprovisioning>> sub-interaction supports all interactions required in advance of committing to a provisioning request. It has one interface.

IPrepareInfrastructureServices

- Given an infrastructure request, returns a list of available infrastructure configurations that could realise that request.
- Given an infrastructure configuration, reserves this infrastructure and returns a reservation ID and the duration for which this reservation will hold.
- Given a reservation ID, cancels the reservation if it exists.

Overview of Interaction Sequence

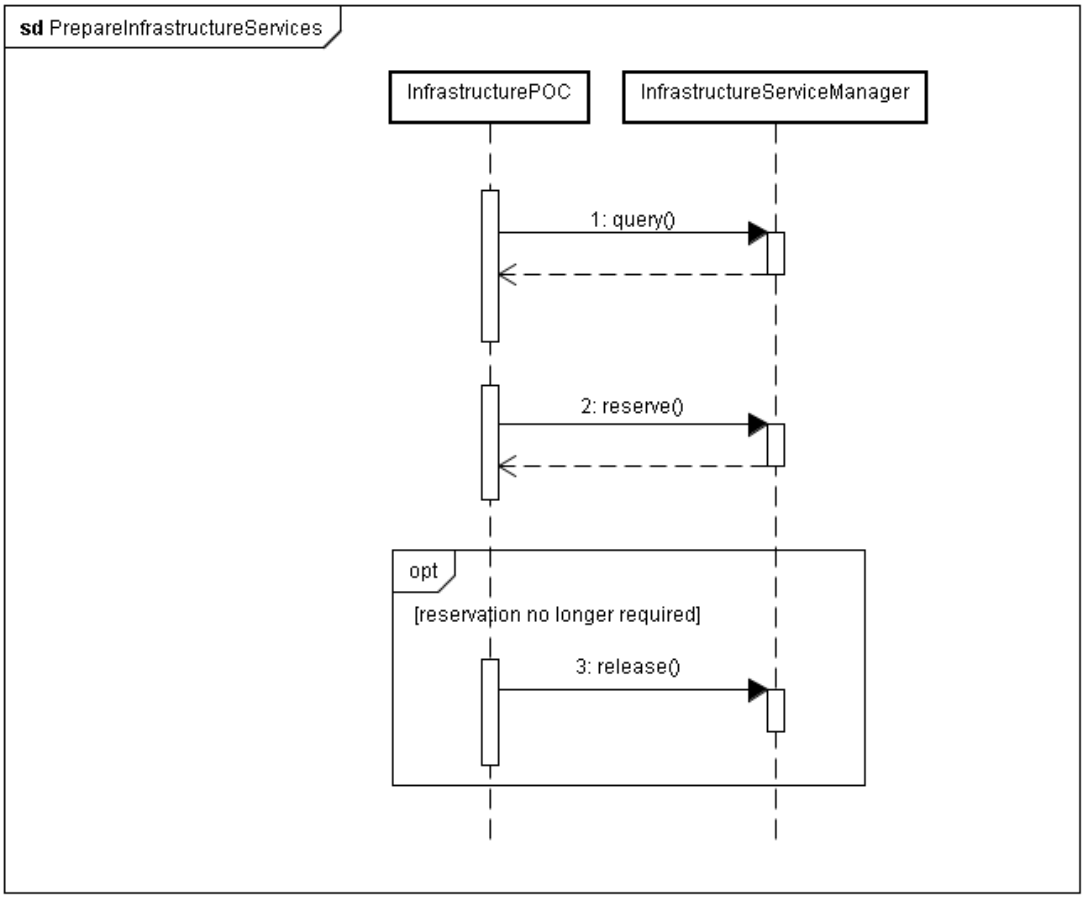


Figure 20: Prepare Infrastructure Services Sequence Diagram

The interaction sequence can be summarised as:

1. The InfrastructurePOC queries to see what are the actual provisioning requests that the InfrastructureServiceManager can satisfy that best match the given provisioning request.
2. The InfrastructurePOC reserves the actual provisioning request that the InfrastructureServiceManager may be able to satisfy.
3. The InfrastructurePOC may release the reservation if the reservation is no longer required.

5.10 <<manage_infrastructure_service>>

Overview

The interaction **<<manage_infrastructure_service>>** is provided to manipulate infrastructure resources. Infrastructure reservations made by **<<prepare_infrastructure_services>>** can be committed. Provisioned resources can be queried, modified and deleted. This stereotype provides a single interface. It allows infrastructure solutions to be actually provisioned, and provisioned infrastructure to be adjusted, queried, and manipulated.

Overview of Interfaces and Operations

The IManageInfrastructureService interface supports all interactions required to provision requested infrastucture, and to manipulate it once provisioned.

- Given an infrastructureID of an active reservation, or complete details of a new request, commits to this reservatio and scheduling the infrastructure for provisioning.
- Returns the details of an infrastructure request
- Given an existing provisioning request, and an updated provisioning request, replaces the former with the later, if possible.
- Given a particular resource, allows that resurce to be started or stopped
- Given a particular provisioning request, terminates the entire provisioning request, tearing down any live VMs and removing any resource commitments previosuly scheduled

Overview of Interaction Sequence

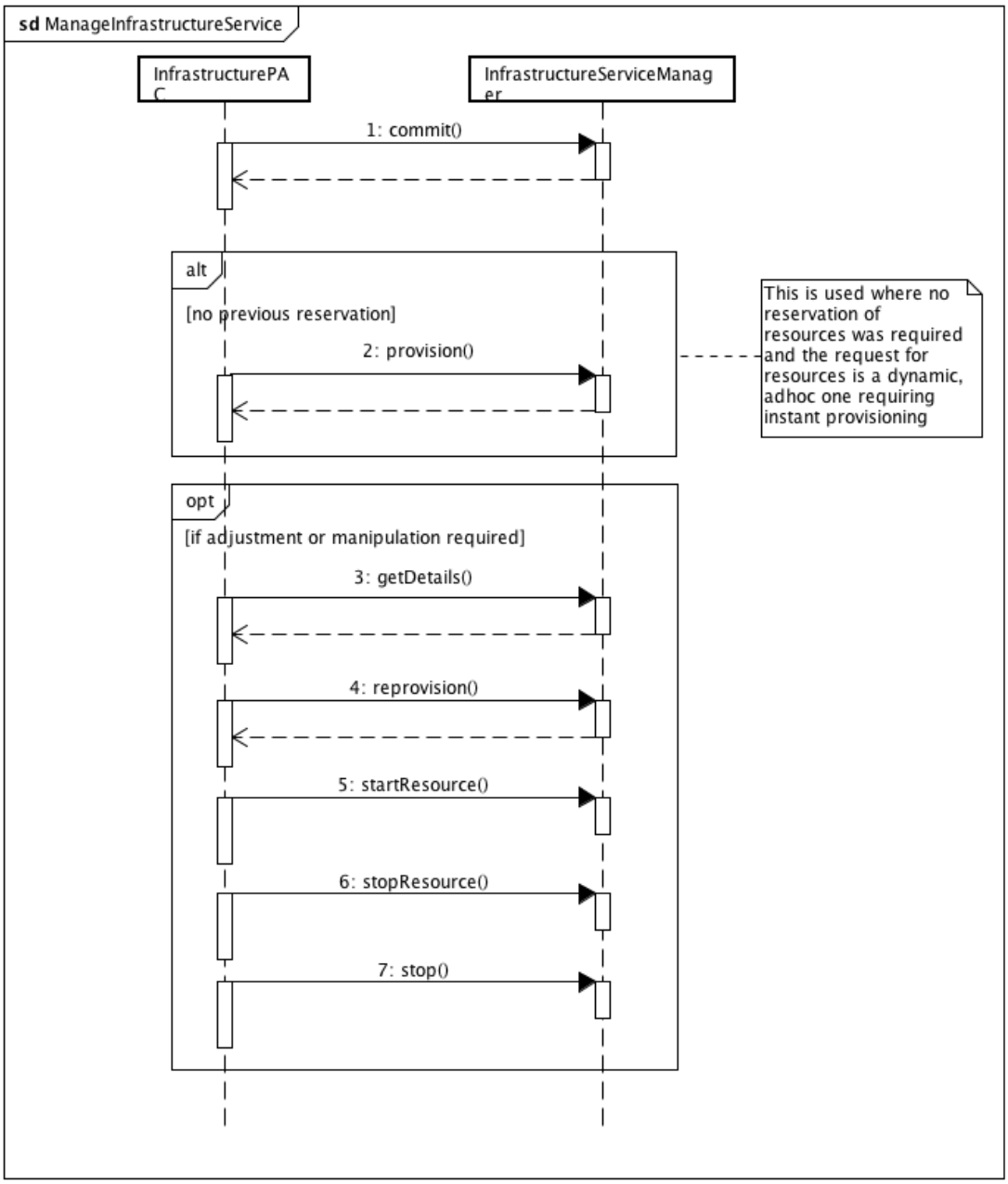


Figure 21: Sequence Diagram for <<Manage_Infrastructure_Service>>

The interaction sequence can be summarised as:

1. An existing reservation can be committed to.
2. Alternatively, a new provsioning request can be made directly without the need for a previous reservation
3. Details of a provisioning request and its current status can be queried
4. An existing provisioning request can be reprovisioned.
5. An individual resource can be started (assumming stopped)
6. An individual resource can be stopped (assumming started)

7. All resources associated with the provisioning request can be stopped.

5.11 <<evaluate>>

Overview

Involved Components

- PlanningAndOptimization (POC) sub component of the SLAManager (for business, software, infrastructure)
- ServiceEvaluation (SE)

The <<evaluate>> interaction allows the PlanningAndOptimization (POC) sub component of SLAManagers to evaluate possible realizations of a service regarding their expected quality. The evaluation is done by the ServiceEvaluation (SE) component. The general concepts of service evaluation are explained in detail on http://sdqweb.ipd.kit.edu/wiki/Palladio_Component_Model. In the following we give a brief overview on how the ServiceEvaluation is accessed by the SLA Manager.

Overview of Interfaces and Operations

The <<evaluate>> interaction specifies an interface IServiceEvaluation, which provides an operation for evaluation of the quality of a target software service. The target service may depend on a list of other services. The quality of the required services may influence the quality of the target service. A set of possible realizations of the target service, including concrete quality descriptions for required services, is given as an input to evaluation.

Overview of Interaction Sequence

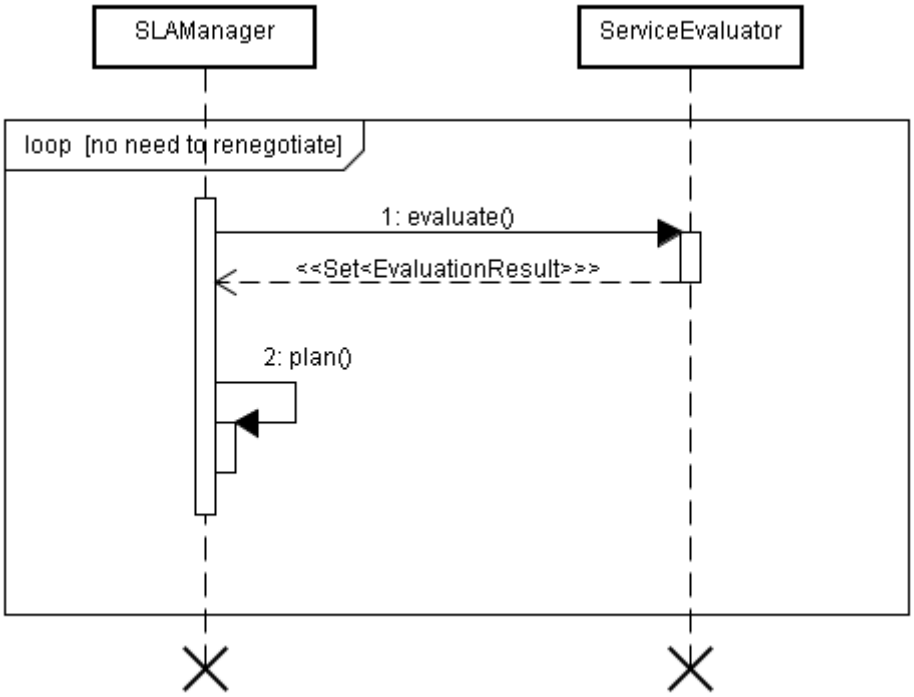


Figure 22: High-level Interaction of POC (via SLAManager) and ServiceEvaluator

The interaction consists of one call evaluate() that is issued by the POC and triggers SE to determine quality estimates for a set of ServiceRealizations. SE returns these realizations in a list, together with the associated quality estimates, back to POC. POC can use these results for its planning functionality (which is only sketched here) and possibly reinvoke the evaluate() method to get quality estimates for further ServiceRealizations.

Detailed Specification

The detailed specification of the listed interfaces and the involved datatypes can be found in the SLA@SOI open source project [1] under the specification of the SoftwareServiceEvaluation component: http://sourceforge.net/apps/trac/sla-at-soi/wiki/SoftwareServiceEvaluation

5.12 <<subscribe_to_event>>

The <<subscribe_to_event>> interaction allows the different components to register in the monitoring event bus (MonitoringEventChannel) specifying the type of message or the channel of its interest. The components that will act as subscribers are:

- Manageability Agents, to receive events emitted by Sensors.
- Provisioning and Adjustment (PAC) sub-component of the SLAManagers, to receive SLA Violation events emitted by the Manageability Agents (*).
- Effectors, to receive commands sent by Manageability Agents

This interaction is closely related to the <<publish_event>> interaction. In order to allow the reusability of the pubsub mechanism implemented in year 1, all the class names and interactions are taken from the common.messaging module.

Involved components:

- SLAManager
- ManageabilityAgent
- MonitoredEventChannel

Overview

This stereotype provides a single interface. It allows the registration and subscription to a given channel of an event bus.

Overview of interfaces and operations

The relation <<subscribe_to_event>> subsumes the functionality that a component needs to connect to the monitoring event bus and register itself to receive a specific type of event, or to receive the events sent to a given channel.

ISubscription

This interaction must allow:

- connection/disconnection to the monitoring event bus
- creation/deletion a channel in the event bus
- checking whether a channel already exists or not
- subscribe/unsubscribe to a channel

Overview of Interaction Sequence

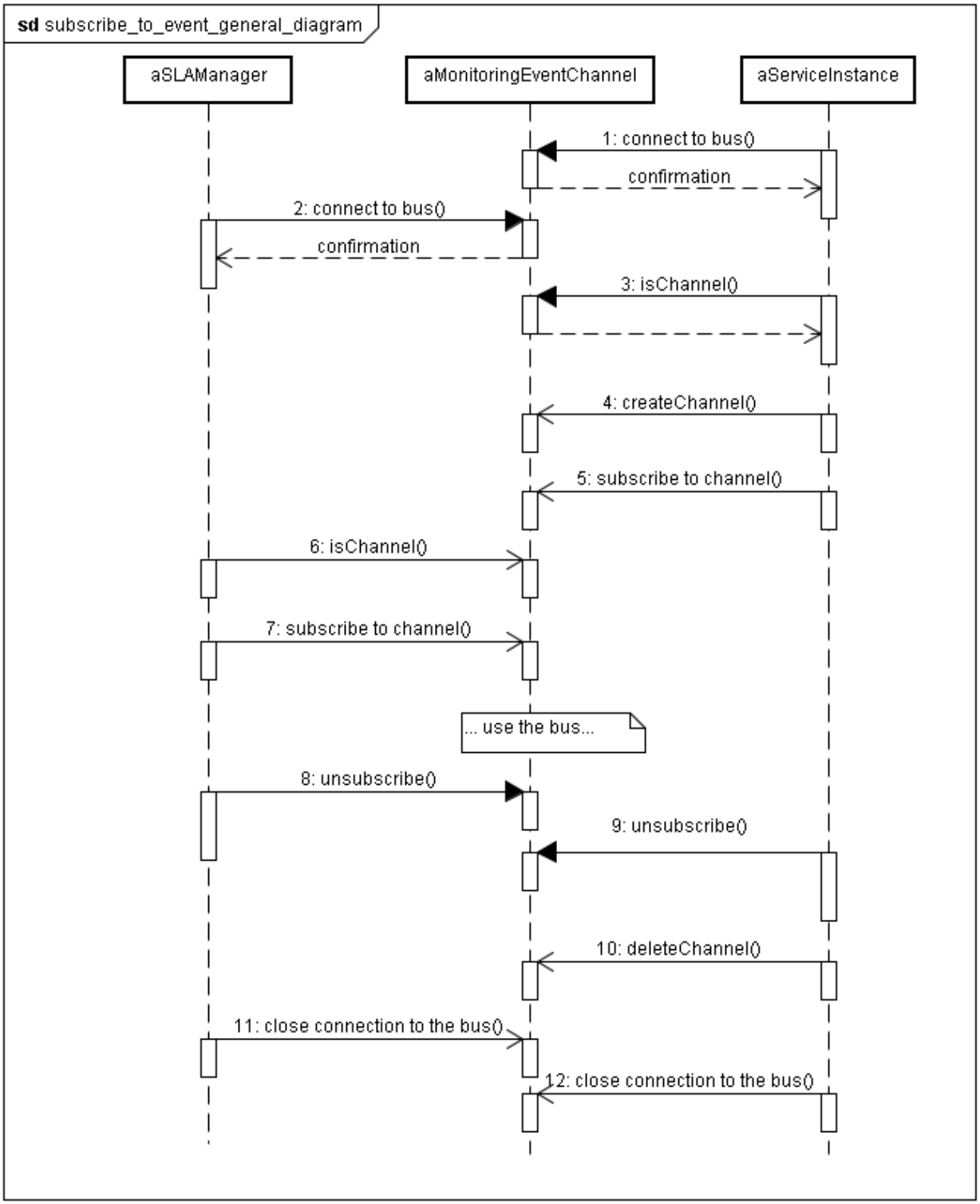


Figure 23: Generic sequence diagram showing the subscription mechanism that allows the communication between SLAManager, ServiceInstance in order to send/receive monitoring events through the MonitoringEventChannel.

Figure 23 illustrates the communication between a SLAManager and a ServiceInstance in a generic fashion. Details about the data types passed and exact method names are omitted.

First, both, sender and receiver must connect to the same instance of the event bus. Before using a given channel, the method isChannel() (steps 3 and 6) is used to confirm whether the channel already exists. If the channel does not exist, it must be created (step 4). If the channel is successful created, the receivers of the messages can subscribe to it (steps 5 and 7). The sender can then start publishing messages in this channel (see the <<publish_event>> interaction).

Finally, when the communication is not more needed, senders and receivers unsubscribe from the channel (steps 8 and 9), delete it (step 10) and disconnect from the bus (steps 11 and 12).

Detailed Specification

Local data types

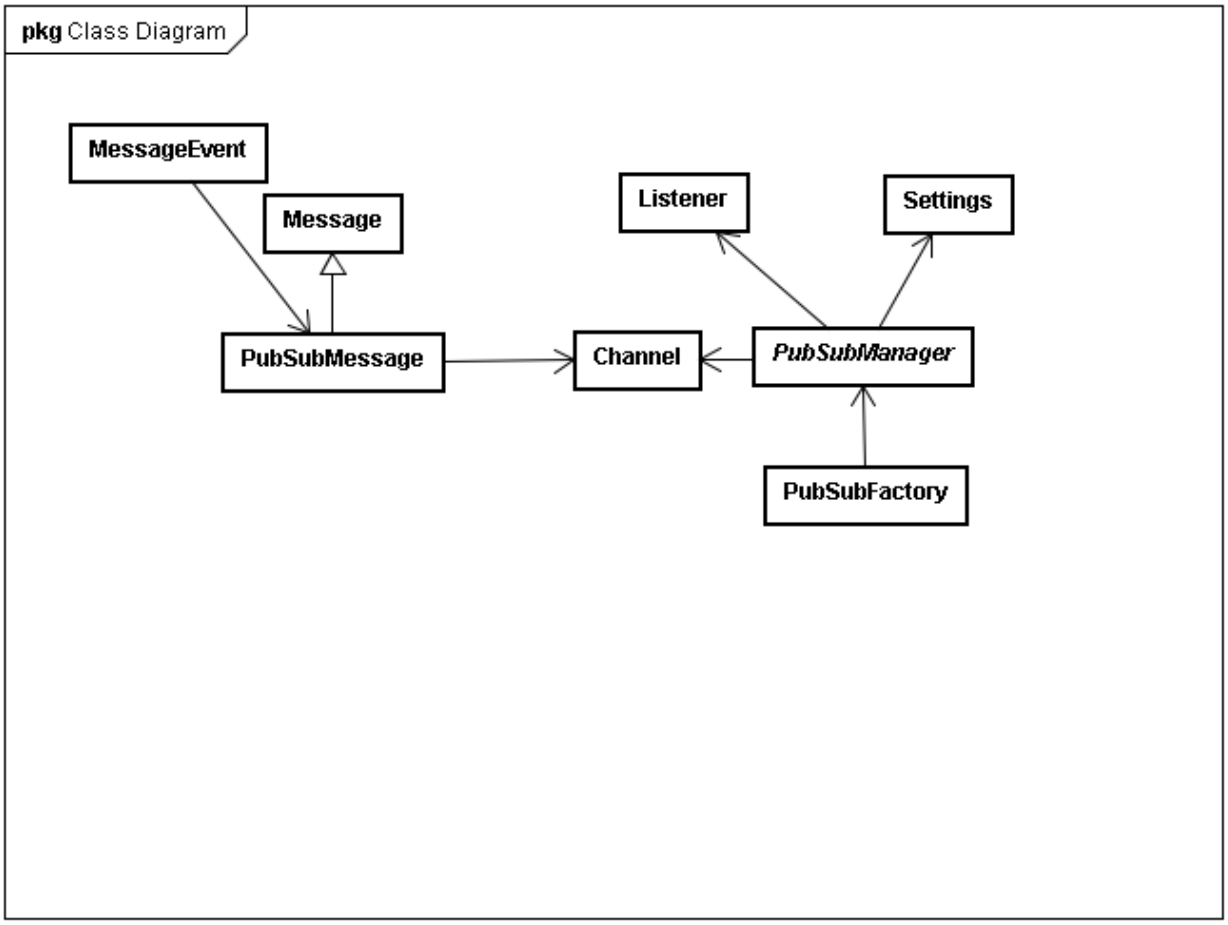


Figure 24: Class diagram for <<subscribe_to_event>>

The main class is the PubSubManager, that offers all the methods needed for the interaction with the bus and the channels (connect/disconnect, subscribe/unsubscribe, create/delete channels). When a component needs to subscribe to a channel, it creates a listener and adds it to the PubSubManager. The PubSubManager stores all the listeners attached to it in a Listener object.

The messages exchanged through a given Channel are PubSubMessage objects. MessageEvents received through the listeners are wrappers of PubSubMessages.

Exceptions

- MessagingException

Detailed Specification of Interfaces

ISubscription

ISubscription
+ connect() : void + createChannel(Channel channel : int) : void + isChannel(String channelName : int) : boolean + deleteChannel(String channelName : int) : void + subscribe(String channelName : int) : void + unsubscribe(String channelName : int) : void + close() : void + addMessageListener(MessageListener listener : int, String[] channelNames : int) : void + addMessageListener(MessageListener listener : int) : void + removeMessageListener(MessageListener listener : int) : void

Figure 25: ISubscription API

- connect(): void: connect to a given event bus using specified Settings.
- createChannel (Channel channel) : void: create the specified channel in the event bus
- isChannel (String channel) : boolean: checks whether a channel already exists
- deleteChannel (String channel) : void: delete the specified channel
- subscribe (String channel) : void: subscribes to a specific channel of the event bus
- unsubscribe (String channel) : void: unsubscribes from the channel
- close(): void: closes the connection to the event bus
- addMessageListener(MessageListener messageListener, String[] channelNames) : void: adds a listener so the subcriber is notified each time a event is sent to any of the channels specified in the second argument.
- addMessageListener(MessageListener messageListener) : void: This method is used when the subscriber has previously subscribed itself to a given channel. Then, it adds a listener so the subscriber is notified each time an event is sent to that channel.
- removeMessageListener(MessageListener messageListener) : void: removes a previously created listener.

This interface was implemented by the PubSubManager class in year 1.

Detailed Sequence Diagram

The following diagram is a detailed view of the general diagram depicted above, with all the involved actors. Yellow classes (PubSubFactory, PubSubManager) are part of the messaging implementation. The diagram uses one event bus, and three channels:

- Channel1

    - Publisher: ManageabilityAgent
    - Subscriber: Provisioning and Adjustment Component (PAC) of a SLAManager
    - Messages: SLA Violations or SLAWarnings
- Channel2
    - Publisher: Sensor
    - Subscriber: ManageabilityAgent
    - Messages: Instrumentation events
- Channel3
    - Publisher: ManageabilityAgent
    - Subscriber: Effector
    - Messages: Commands to trigger certain actions

Even this is not mandatory, we have applied the convention that a given channel is created and deleted by the publisher.

The interaction begins with a subscriber and a publisher invoking the createPubSubManager operation of the PubSubFactory that triggers the allocation of a new PubSubManager, which realizes the connection to the physical bus. A PubSubManager objects in the end wraps the commands with a particular instance of the event bus.

Once the connection has been successfully achieved, the publisher checks whether the channel exists, and, if not, it creates the channel. The subscriber then can subscribe and add a listener so it will receive all the messages sent to that channel.

Finally, in order to disconnect the objects from the event bus, both publisher and subscriber must unsubscribe from a given channel, the channel should be deleted and finally the connection to the event bus must be closed.

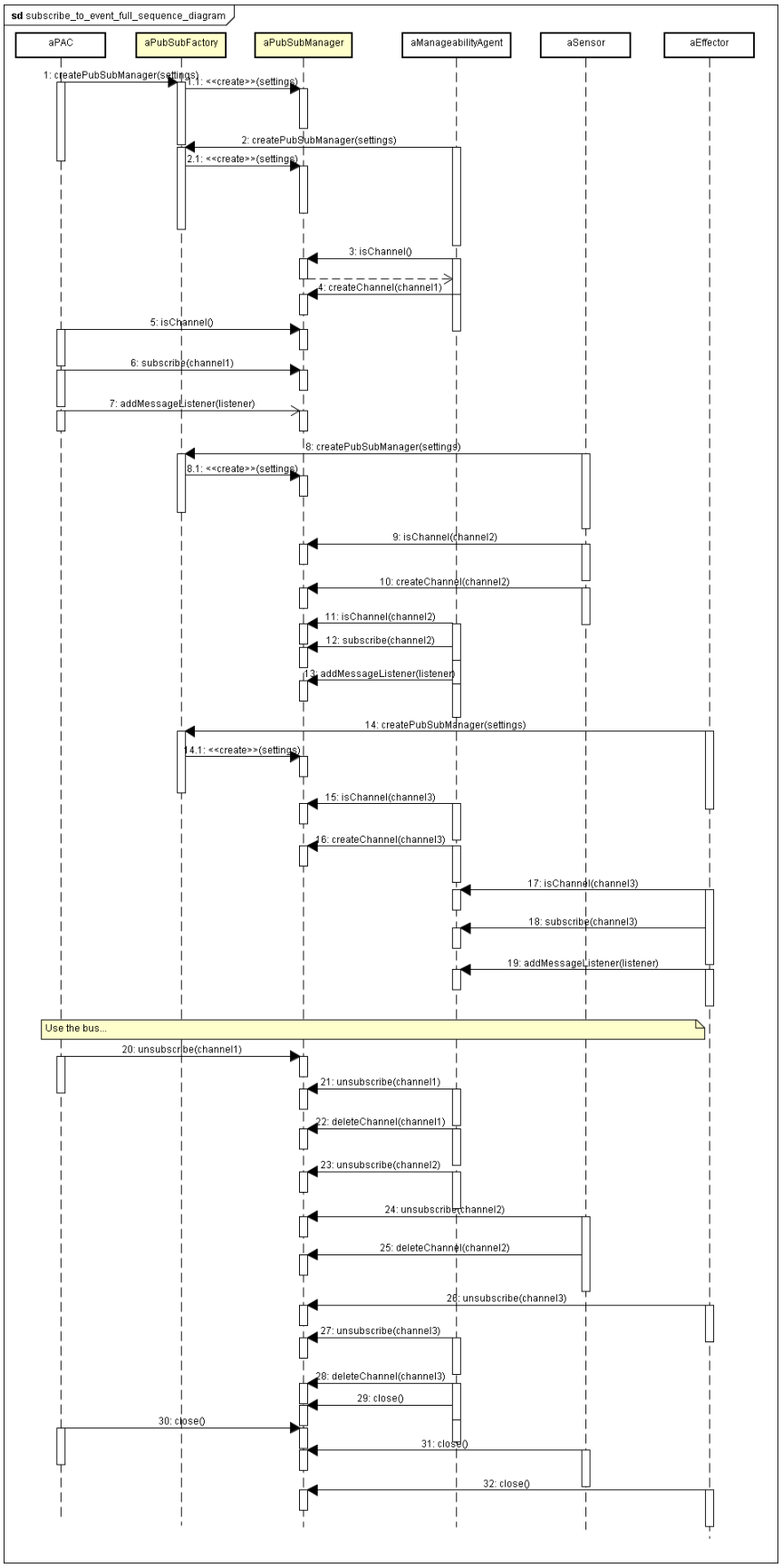


Figure 26: Full sequence diagram of <<subscribe_to_event>> interaction

5.13 <<publish_event>>

Overview

This stereotype is provided to allow publishing events to all interested (subscriber) components. Events provide some general information about lifecycle of monitored resource and are not by any mean subscriber specific. Events are published even if no one subscribes to them.

Events are published through the channels identified by channelName. Subscribes use <<subscribe_to_event>> stereotype to connect to a given channel and receive all the events on this channel.

Events have a predefined structure and type.

Involved components

This stereotype allows the Monitoring System to publish resources related events based on their resource specific configuration.

All components interested in general information about monitored resources, their status, violations, actions, errors, etc. can collect these data.

NOTE: The amount and the content of published event fully depends on Monitoring System configuration. It might be configured that some resources publish no events at all.

Overview of interfaces and operations

Events are published by method publish implemented by PubSubManager:

void publish(PubSubMessage message) throws MessagingException;

It sends the message with the following parameters:

- channelName,
- payload (event).

The payload contains the Event with the following structure:

- EventId - unique for each publish
- EventType - from predefined set (ex. VmCpuOverLimit)
- EventClass - warning, error, etc.
- Timestamp
- List of arbitrary parameters with:
    - name
    - value (can be structured)
    - Typical parameters:
        - SLA ID or Infrastructure ID
        - RelatedResourceId
        - Violated constraintID

- Related metrics (with values)

6 Interaction Sequences

In this section, we provide an extended example of how actual interaction sequences **may** take place. For this purpose, we assume that the framework is instantiated so that all components exist once as shown in Figure 7. The interactions are described in terms of sequence diagrams. The chosen representation is semi-formal as it tries for an individual interaction step to clarify (a) the relation to the top-level interaction stereotype and (b) the relation to the more detailed activity step (as it is specified within the respective specification of an interaction) that takes place. Both aspects are separated by a ":" resulting to the following description style: <<name_of_top-level_stereotype: name_of_activity_step>>.

The sequence diagrams shown in the following are still on an abstract level. Appendix C gives a detailed example for the specification of interactions involved in the integrated planning cycle for software services.

6.1 Negotiation interaction

In the following, we describe the basic steps of a customer-initiated negotiation process. We distributed the process among three sequence diagrams that show the interaction on the business layer (Figure 27), the negotiation process between software and infrastructure layer (Figure 28), and the creation of an agreement (Figure 29).

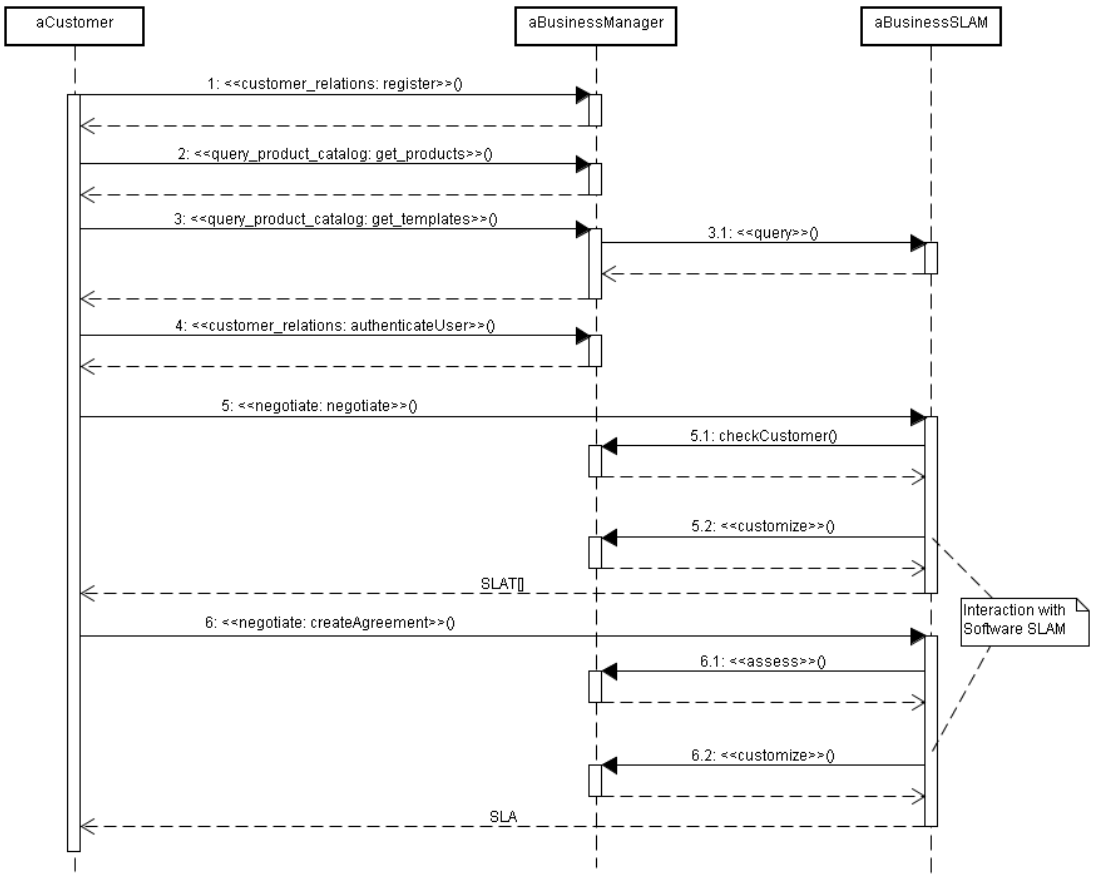


Figure 27: Negotiation interactions on the business layer.

The sequence diagram in Figure 27 shows the interactions between a customer, a Business Manager and a Business SLA Manager. As a first step, the customer registers at the business manager, for example using the framework's web portal. Then the customer browses through the product catalogue and inspects the SLA Templates of specific products. If the customer found an interesting product, he or she authenticates at the business manager and can now start the negotiation process. For this purpose, a negotiation request regarding a specific product is sent to the Business SLA Manager. The Business SLA Manager first checks the credentials of the customer before it passes his or her request to the other framework components for further negotiation with other SLA managers (cf. Figure 28). Once the Business SLA Manager received a set of potential SLA Templates from the other components of the framework, it calls back the Business Manager to adjust the templates for the particular customer. For example, power users get a discount of 20%. The customised templates are then returned as an individual offer to the customer. Finally, the customer selects one of the offers, adds his or her individual requests and triggers the creation of an agreement. During this process, the Business SLA Manager first calls the Business Manager to assess the incoming request. If the request is valid, the framework internal process for the creation of an agreement is triggered (cf. Figure 29). Once all necessary steps, such as the reservation of resources, have been taken the final SLA is again adjusted for the particular customer and an SLA is returned. The SLA contains all information necessary for the customer, including the service's endpoint and the date and time when the service will be available. In the following, we elaborate the negotiation process between the software and infrastructure layer in more detail.

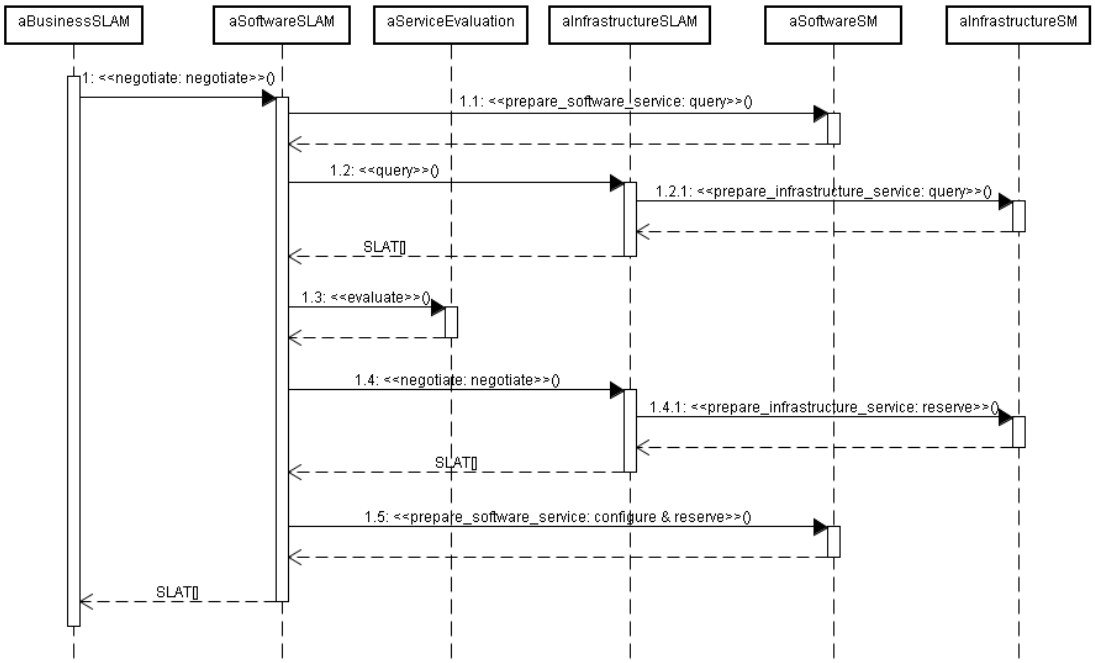


Figure 28: Negotiation interactions between software and infrastructure layer.

The sequence diagram in Figure 28 illustrates how the negotiation process continues from the Business SLA Manager across the different layers starting with the Software SLA Manager. To provide a set of possible SLA Templates to the Business SLA Manager, the Software SLA Manager queries the available service implementations from its associated Software Service Manager (SSM). In order to determine the quality of service that can be provided for the customer's request, the Software SLA Manager needs to resolve all dependencies that the implementation may have. For example, one of the implementations may rely on an infrastructure service hosting its appliance. Therefore, the Software SLA Manager queries available SLA Templates from an Infrastructure SLA Manager to identify potential alternatives for the service needed by the service implementation. Based on the alternatives for the dependent services, the Software SLA Manager starts its internal planning. For this purpose, it calls the Service Evaluation to estimate the quality of service that can be achieved by a particular set of external services used by an implementation. Judging the achieved quality of service against the required one, the Software SLA Manager decides which alternatives are to be taken into account. For these alternatives, it starts the negotiation process with the Infrastructure SLA Manager which in turn reserves the necessary resources. Finally, the Software SLA Manager itself reserves all necessary software resources calling the corresponding Software Service Manager. When the negotiation process has been completed, the customer can select one of the returned SLA Templates, add his or her adjustments, and create an agreement described in the following.

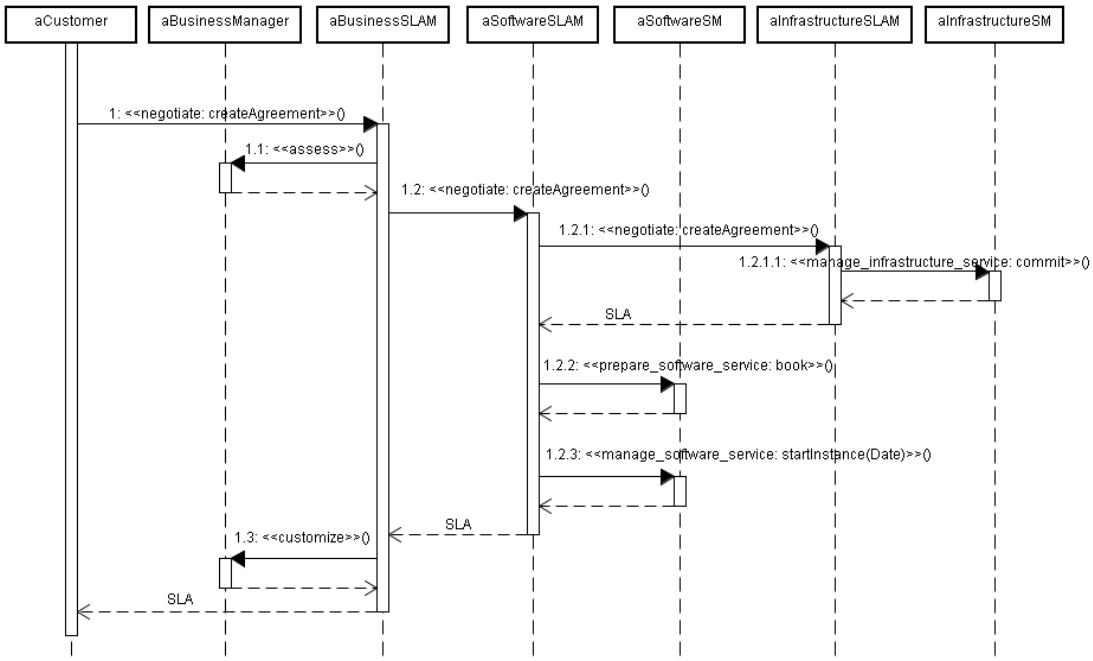


Figure 29: Interactions during the creation of an agreement.

Figure 29 shows a sequence diagram for the interactions necessary to create an agreement among all involved parties. The customer triggers the creation of an agreement. Once the Business Manager has assessed its validity, the Business SLA Manager propagates the request to the Software SLA Manager. The Software SLA Manager first needs to ensure that all dependent services are available. For this purpose, it tries to create an agreement with the connected Infrastructure SLA Manager which (if the request can be handled successfully) commits the previous reservation and makes the corresponding infrastructure available. Once all dependent services have been acquired, the preparation on the Software Service Manager's side can start. It books all the resources needed to provision the software (such as software licenses) and then schedules the provisioning of the service calling startInstance with the date and time when the service must be available. Finally, the agreed SLA is returned to the Business SLA Manager that returns it after a customization by the Business Manager to the customer. This concludes the negotiation process. Next the service needs to be provisioned which is described in the following section.

6.2 Provisioning interaction

In this section, we describe two variants of the provisioning process: explicit provisioning (Figure 30) and time triggered provisioning (

Figure 31 and Figure 32). In the first case, a customer explicitly triggers provisioning. In the second case, it is executed implicitly by SLA managers based on the times specified in the SLA.

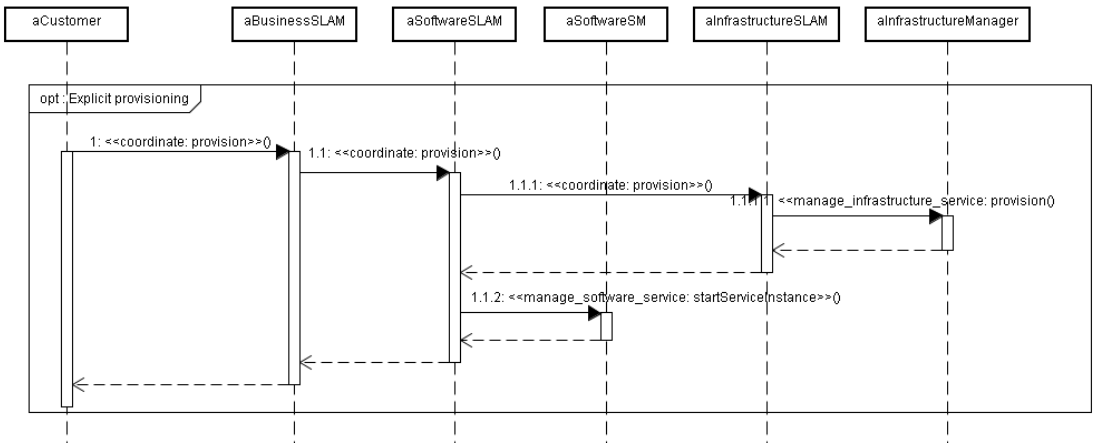


Figure 30: Explicit provisioning triggered by the customer.

Figure 30 shows a sequence diagram illustrating how provisioning can be triggered explicitly by the customer. Once triggered, the provisioning request is propagated from the Business SLA Manager, to the Software SLA Manager, and to the Infrastructure SLA Manager. In the sequence diagram shown, the infrastructure services required for the software service are provisioned first. Once they are available the Software SLA Manager can proceed and provision its services. Finally, the Business SLA Manager can finish the provisioning sequence. When the request returns to the customer, he or she can immediately use the requested service. However, provisioning is often a lengthy and time consuming process. Therefore, customers do not want to trigger provisioning manually, but require the service to be available at a particular time. This can be achieved by the implicit provisioning described in the following.

In order to enable implicit, time-triggered provisioning, detailed planning of the provisioning process by the SLA Managers is essential. SLA Managers must communicate the time necessary to provision the requested services. Based on these times, a provisioning schedule is determined. Each SLA Manager schedules its provisioning actions according to this schedule. Furthermore, SLA Managers have to notify each other as soon as individual services are available. The notifications are necessary to cope with potential delays during provisioning.

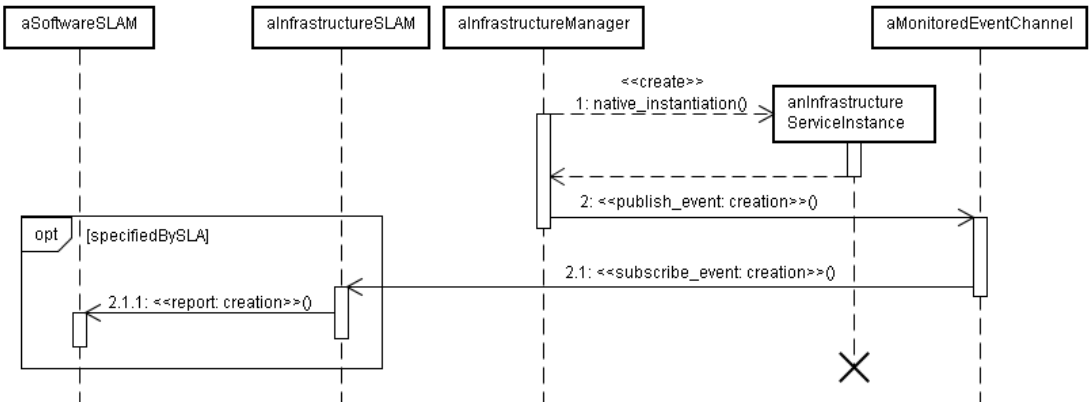


Figure 31: Implicit provisioning of infrastructure services triggered by a timer inside the Infrastructure Manager.

Figure 31 shows a potential interaction flow for the time-triggered provisioning of an infrastructure service. At the point in time that has been determined by the Infrastructure SLA Manager, the Infrastructure Manager starts the creation of a

new instance of the requested infrastructure service. This can translate, for example, into booting a virtual machine. Once the creation as well as the internal book keeping has been finished the Infrastructure Manager publishes a message on the event bus notifying all involved parties that the infrastructure service (e.g. the virtual machine) is available now. In Figure 31, the Infrastructure SLA Manager receives the event and in turn notifies the Software SLA Manager (as it has been agreed on in the SLA).

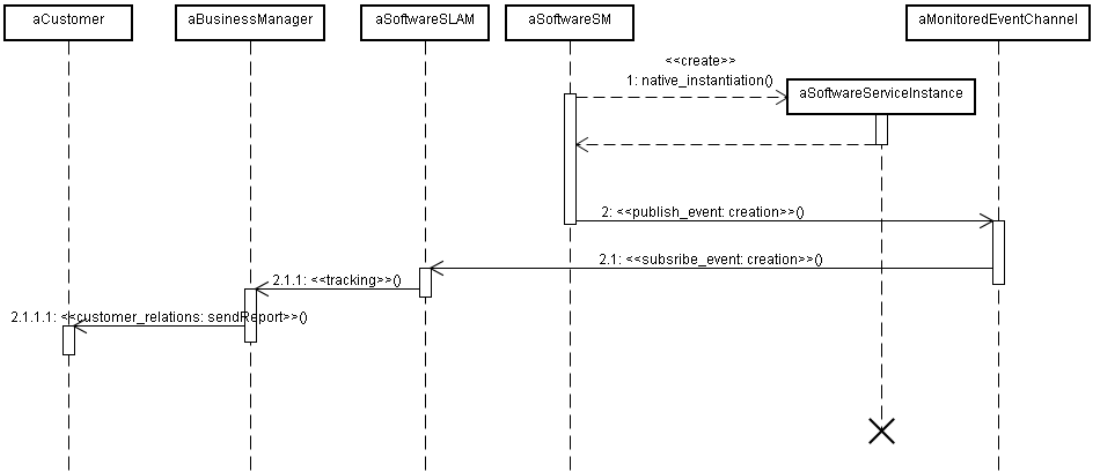


Figure 32: Implicit provisioning of software services triggered by the Software Service Manager.

The process is similar for the provisioning of software services. In Figure 32, the provisioning of the software service is triggered at a particular point in time. Alternatively, it may be triggered by an event of the Infrastructure Manager. In any case, the Software Service Manager creates a new instance of the software service. The creation of the new service instance can be realised by very different means. For example, it can be done by deploying and activating an application on a middleware platform or by creating a new tenant in a multi-tenant system. Once the service has been instantiated and configured, the Software Service Manager puts an event on the message bus signalling the successful creation of the software service to all involved parties. In Figure 32, the Software SLA Manager is triggered by this event and notifies the Business Manager about the successful service creation using its tracking interface. The Business Manager can propagate this information further to the customer. The customer now knows that his or her service is accessible and ready for usage.

In the following sections, we illustrate the main scenarios at runtime, namely the customer or framework triggered reporting, the processing of SLA violations, and customer-initiated renegotiation.

6.3 Adjustment & Reporting interactions

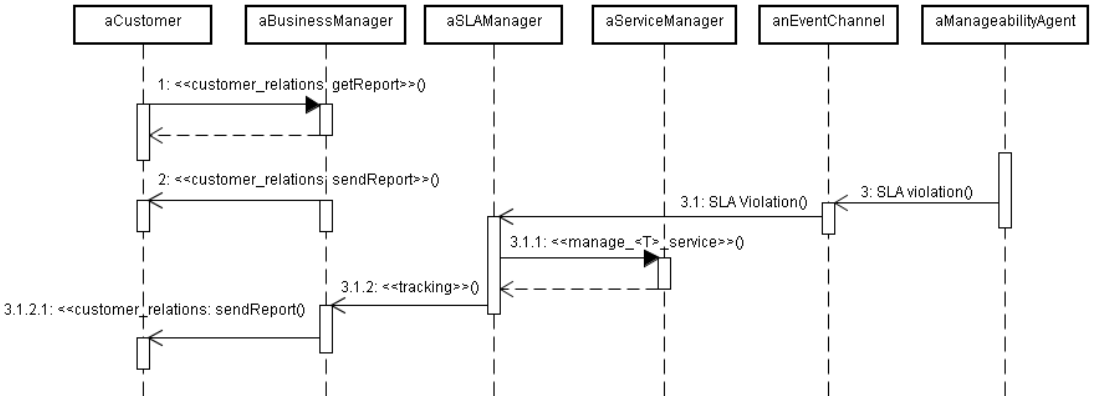


Figure 33: Interactions for reporting the status of SLAs and their violation.

Interaction 1 and 2 in Figure 33 illustrate two different ways of how information about a service's status can be brought to the customer. In the first case the customer pulls information from the Business Manager using an explicit call (getReport). In the second case, the framework pushes information about a service's status to the customer in regular intervals or in case of specific events. Both scenarios are valid for the framework. However, the terms and conditions have to be defined explicitly in the SLA.

Interaction 3 in Figure 33 illustrates the notification of the customer in case of an SLA violation observed by the framework. The Manageability Agent and Monitoring System continuously observe the service's quality. If the observed quality is worse than the quality agreed in the SLA, it publishes the resulting violation on an Event Channel. In the scenario depicted here, the SLA Manager retrieves the notification and initiates countermeasures (i.e., calls manage_<T>_service, where <T> can either be infrastructure or software in this case). If successful, the countermeasures allow the framework to restore the quality of service agreed on in the given SLA. The violation and the taken countermeasures are reported to the Business Manager via the <<tracking>> interaction. The Business Manager records the violation and decides based on the severity of the violation and the success or failure of the countermeasures whether to report the violation to the customer or not. Sometimes the violation cannot be handled by the adjustment of the service stack and renegotiation with the customer is necessary. For example, this can be the case if the customer violates the SLA by putting too much load on the system. In the following, we describe the renegotiation process in more detail.

6.4 Renegotiation interactions

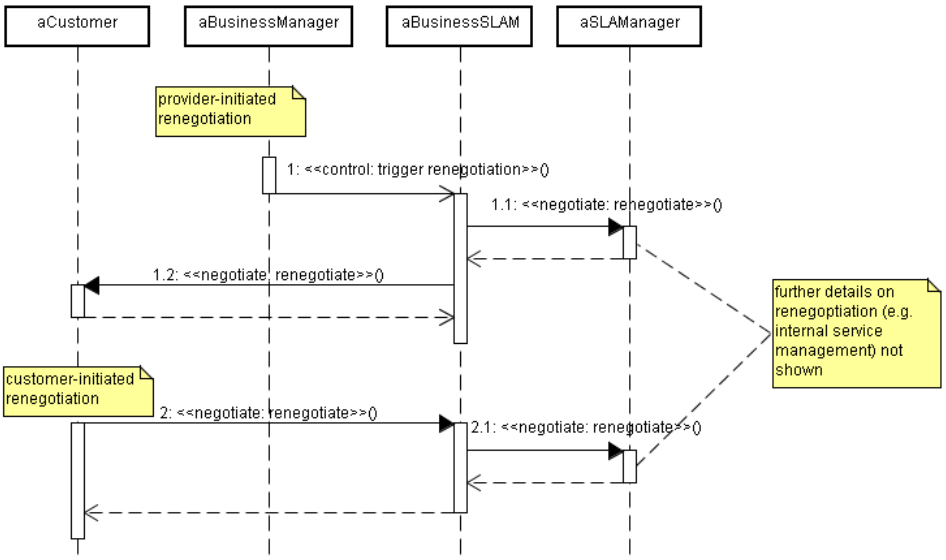


Figure 34: Interactions for the renegotiation of SLAs.

Renegotiation can be triggered either by the Business Manager or the customer. In the first case, the Business Manager may observe an SLA violation (caused by the framework or the customer) that cannot be handled by the framework itself. In this case, the Business Manager triggers an internal (re-)negotiation process of the framework and its connected SLA Managers to solve the problem. Based on the possible solutions found, the renegotiation with the Customer is initiated.

In the second case, the customer requires changes in the quality agreement made in the SLA. Therefore, he or she triggers renegotiation with the Business SLA Manager. Basically, the interactions for renegotiation are similar to the ones shown in Figure 27 and Figure 28.

7 Manageability approach

In this section, we provide a detailed introduction of the manageability approach employed inside the SLA@SOI framework. Monitoring and adjustment are to a large extent cross-cutting concerns of our framework. As such, a detailed understanding of the manageability part of our architecture is essential. Therefore, this deliverable captures its most important aspects in detail.

Manageability, monitoring and adjustment systems collaborate to provide reliable services to customers, assuring the agreement between customer and service provider. Due to their cross-cutting nature, they serve as a fabric underlying the rest of the framework and support the service and SLA management in a flexible but still harmonized way.

Figure 35 shows the main components of the manageability system.

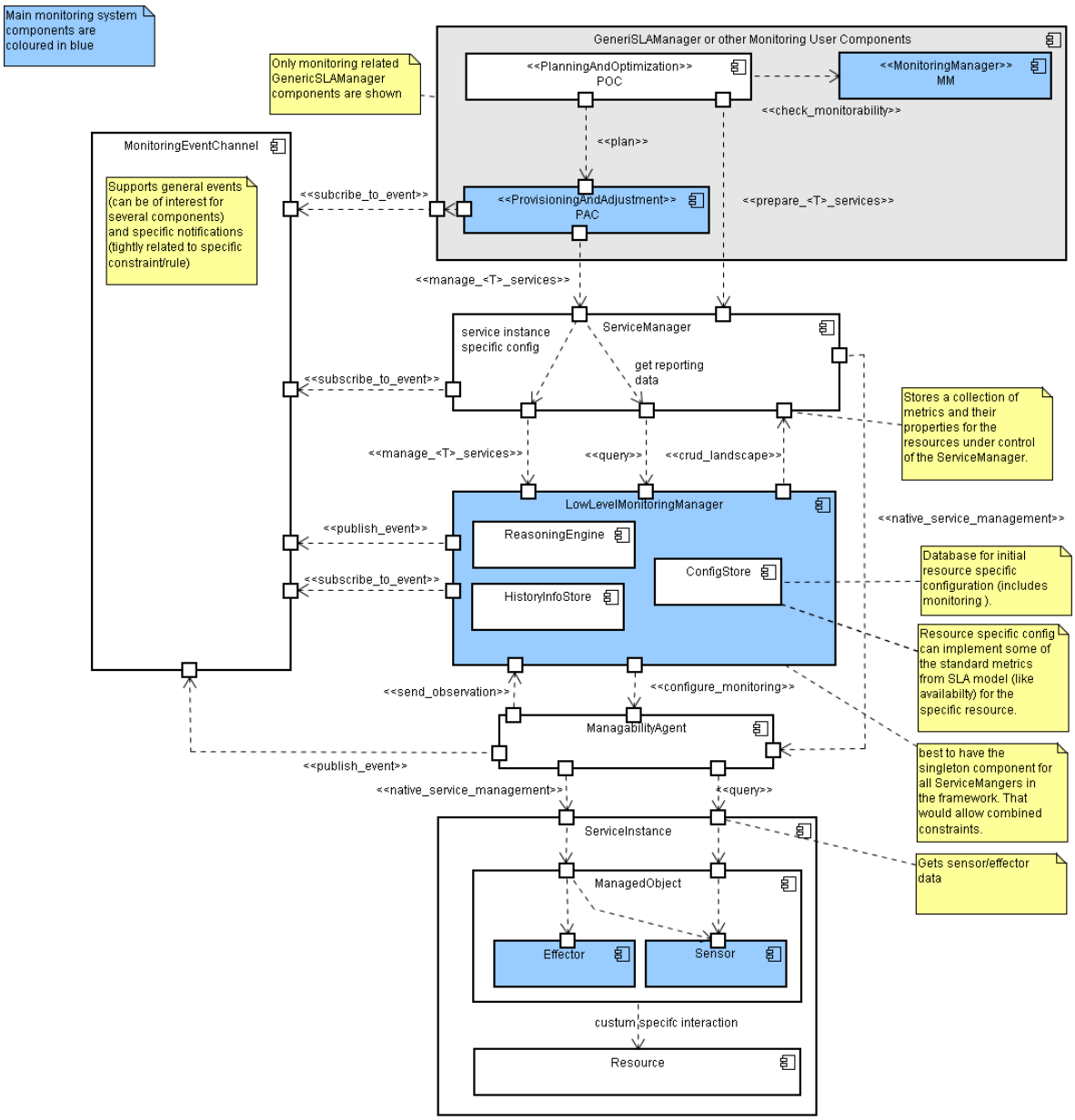


Figure 35: Manageability, Monitoring and Adjustment system architecture

The purpose of the Monitoring and Adjustment Management System (MAMS) is to compliment the service manager by providing the means to monitor a provisioned service instance in the context of the requested guarantee terms contained in the SLA. The system needs to be general enough such that its architecture is applicable to both software and infrastructure service managers. In the following we describe the key components of the MAMS (printed in blue in Figure 35).

The **MonitoringManager** (MM) coordinates the automatic configuration of the monitoring system. It decides, for any SLA it receives, which is the most convenient monitoring configuration according to configurable selecting criteria. A monitoring configuration determines which components are to be configured as well as their configurations.

Adjustment (PAC) collects information from the Low Level Monitoring System, analyzes the incoming events, and triggers the best corrective or proactive action in case of (potential) problems. If the PAC cannot solve the problem at a local level, it escalates the issue to a higher level. In case of an SLA violation, the PAC

can trigger re-planning, re-configuration and/or alerts at higher levels. These capabilities are considered to be important in order to guarantee best user perception preserving underlining resources.

The **LowLevelMonitoringSystem** is a central entity for storing and processing monitoring data. It collects raw observation data, processes them, computes derived metrics, evaluates rules, stores monitoring data, and offers the data to other components (accessible through Service Managers). It is general by design and thus capable of supporting infrastructure and software services as well as other use cases.

Sensors collect information about a service instance. Their design and implementation is domain-specific. Sensors can be injected into a service instance, e.g., by direct instrumentation or it can intercept service invocations outside the service instance. Sensors send the collected data to the communication infrastructure. Furthermore, other components can request (query) information from it. Various kinds of Sensors exist that collect different kinds of information. All sensors implement a common interface that provides methods for starting, stropping, and configuring a Sensor.

Effectors are components for configuring the behavior of a service instance. Their design and implementation is domain-specific. Effectors can be used to configure a service instance during its provisioning as well as during runtime. Effectors are the main means for the execution of adjustment actions.

In the following, we describe the concepts for manageability, monitoring, and adjustment in more detail.

7.1 Manageability

In the SLA@SOI framework we provide a unified manageability interface for managing services through appropriately deployed sensors and effectors. The interface is implemented by a generic manageability agent component, a gateway to managing one or more services, hiding the domain-specific details of the the management itself.

The Manageability Agent provides two generic interfaces: the IManageabilityAgent interface and the IManageabilityFacade interface. The generic Manageability Agent component implements the IManageabilityAgent interface, which provides a gateway for accessing services that need to be managed. In particular it allows the framework to:

- start a new service (method *startServiceInstance*)
- stop a service (method *stopServiceInstance*)
- obtain a list of the services being managed through the particular ManageabilityAgent instance (method *getEndpoints*)
- obtain a service instance specific façade component for managing the service (method *getManageabilityAgentFacade*)

Once a façade object is obtained, it can be used to directly manage a service instance through the IManageabilityAgentFacade interface. In order to do so, the façade's internal implementation is necessarily domain-specific. The façade allows the framework to:

- get a ServiceInstance object for the service instance being managed through the façade (method *getInstance*). The ServiceInstance object is part of the Service Construction Model, and contains information for uniquely identifying the instance.

- configure the sensors that will be used to collect run-time data regarding the service instance, and the effectors that will be used to control the service instance (method *configureMonitoringSystem*)
- deconfigure the sensors and effectors associated with the service instance (method *deconfigureMonitoring*)
- get a list of sensor subscription data for the sensors active on the service instance (method *getSensorSubscriptionData*). The sensor subscription data allow the framework to understand what data will be gathered, and how they can subscribe on the XMPP event bus to get them.
- execute a control action on the service instance (method *executeAction*). This allows the framework to invoke domain-specific adjustment actions on the service instance.

Note that more details regarding the ManageabilityAgent and the interfaces it provides can be found in Section 7 of the deliverable "DA3.a - SLA-aware Service Management**".**

7.2 Monitoring

Monitoring provides the following functionalities:

1. Check whether the SLA can be monitored.
2. Perform instrumentation of standard resources (standard infrastructure and SW).
3. Collect instrumentation measurements (from several sources).
4. Store them (HistoryInfoStore).
5. Calculate QoS terms (defined in service configuration).These terms are
6. Expose all monitoring data to other components (PAC, GUI, etc.)
7. Evaluate agreement terms.
8. Send notifications in case of SLA Violations and Warnings

Functionalities 1,7,8 are implemented by SLA Level Monitoring System (SLMS), while functionalities 3-6 are implemented by Low Level Monitoring System (LLMS). Instrumentation is performed by Ganglia and by custom sensor modules.

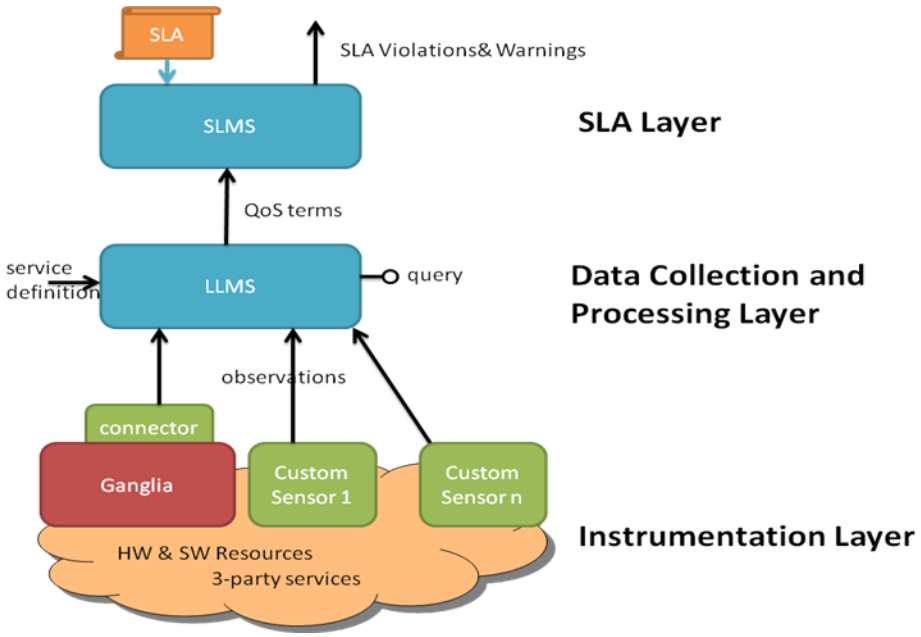


Figure 36: Layers of Monitoring System

Instrumentation

For all further monitoring purposes basic instrumentation data are needed from all monitored resources and services. A single unit of instrumentation data is called **observation** (also measurement). A type of observation is called **metric**. It typically refers to a single resource/service, but can also refer to arbitrary group. Example metrics: cpuUsage on VM1, request/response of a service MyService.

Depending on the type of metric, instrumentation can be done by standard solution or a custom module (called **Sensor**). In both cases the observations has to be send to LLMS by invoking LLMS Remote API.

Instrumentation is a continuous distributed process, so the observations are to be sent asynchronously with minimal delay after they are observed.

Low Level Monitoring System

The inner structure of LLMS, its interfaces and interactions are show on Figure 37.

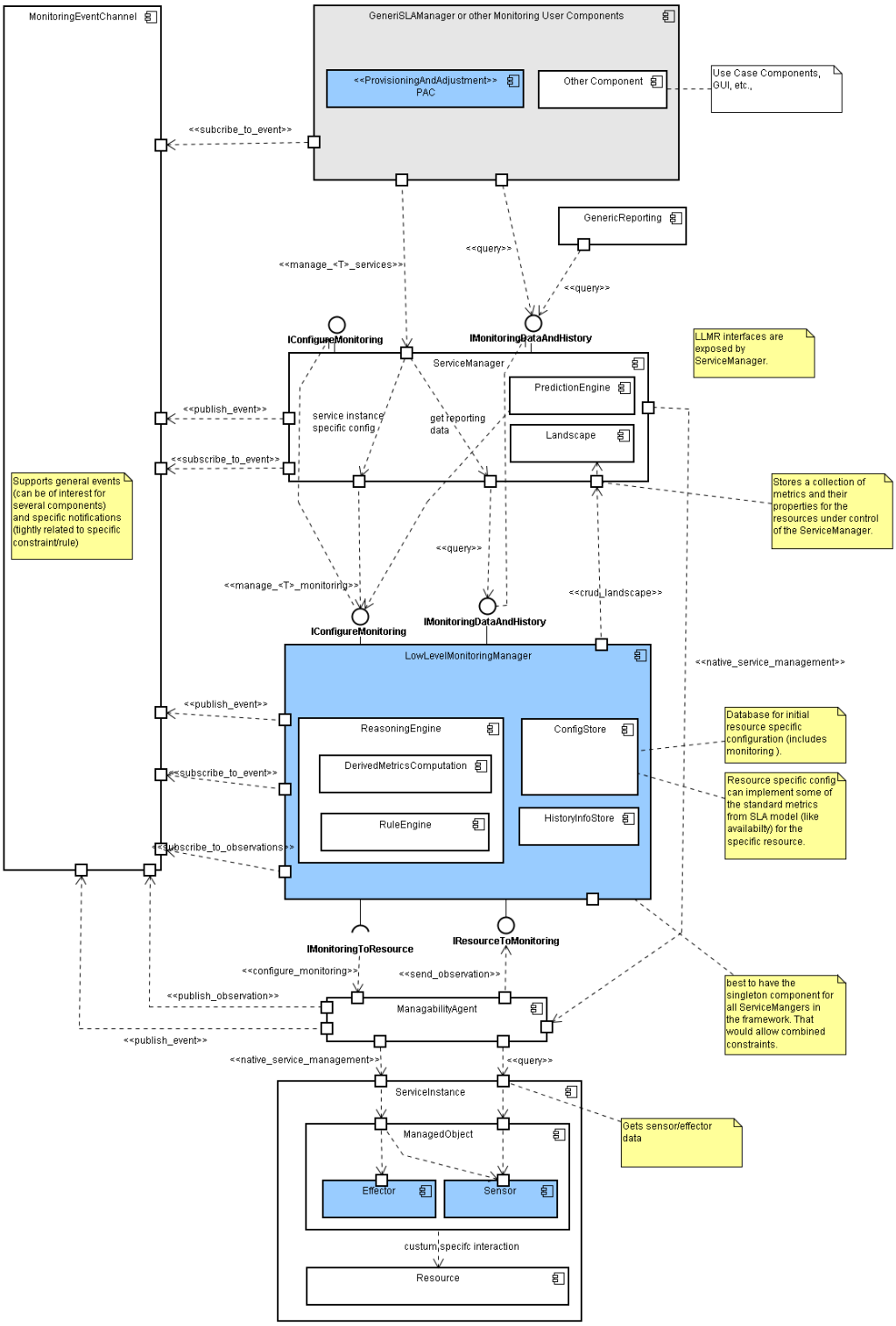


Figure 37: LLMS inner structure, interfaces and interactions

LLMS comprises the following components:

- ResoningEngine
- DerivedMetricsComputation
- HistroryInfoStore
- ConfigStore
- GenericReporting

ReasoningEngine performs reasoning - rule evaluation based on received and historical observations and static parameters. The rules can be data triggered (when new observations arrive) or time triggered (periodic). The action statement of rules can contain arbitrary method call, most typical being sending a (specific) notification or calculating a derived metric observation.

DerivedMetricsComputation is used to define and compute some derived metrics that are computed from basic (observed) metrics. Derived metrics are defined using rule definition and computed in runtime.

HistoryInfoStore stores the history of raw observation, derived metrics values and sent notifications. It also does the history aggregation of old data.

ConfigStore store the resource and service specific configuration (metric parameters and rules).

GenericReporting is the common core of domain aware reporting based on monitoring data.

The following public interfaces are implemented:

- **IConfigureMonitoring** – provision, reprovision, free monitoring request
- **IMonitoringDataAndHistory** – query metrics, their last values and history
- **IMonitoring2Resource** – set metric specific sensor parameters
- **IResource2Monitoring** – register new metrics, send observations

The system is configured through IConfigureMonitoring interface at runtime. ILandscape is used to access the data in ServiceManager Landscape databases. The data is used by LowLevelMonitoringManager and exposed through IMonitoringDataAndHistory interface.

Generic monitoring console is implemented by GenericReporting component that is an independent and optional part of the monitoring system. It collects data though IMonitoringDataAndHistory interface and present them in a user friendly way (GUI). Reports export will be supported through IReporting interface.

For data exchange and other communication common PubSub component will be used, so **MonitoringEventChannel** will be implemented as general **PubSub** component.

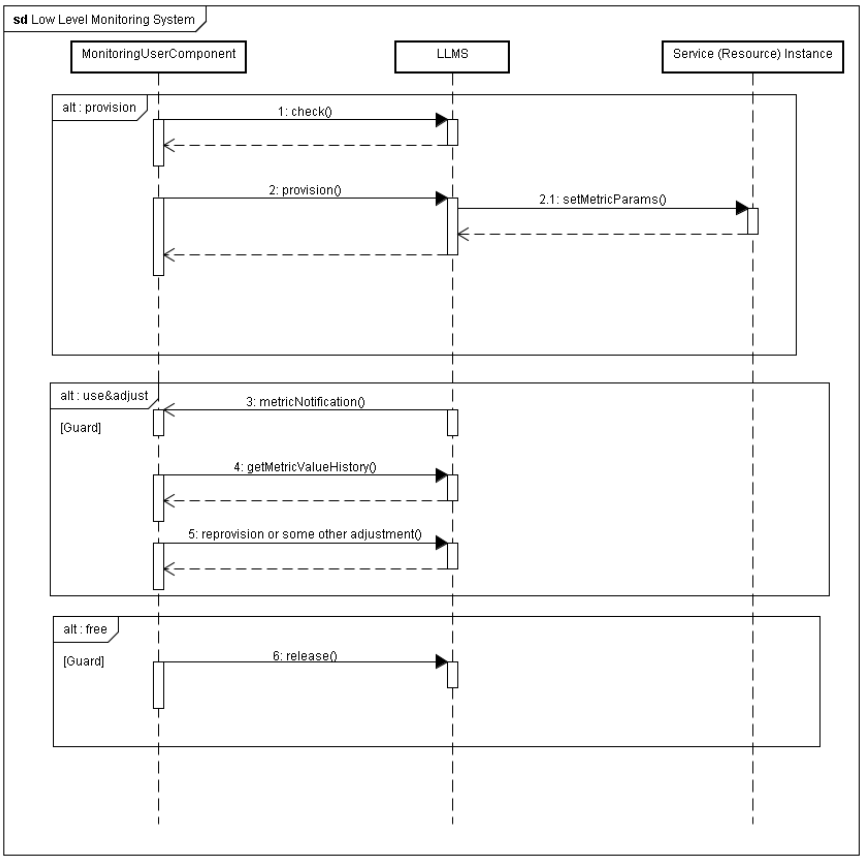


Figure 38: LLMS Interaction sequence diagram

SLA Layer Monitoring System

To provide effective monitoring support when replacements of the services deployed in a service based system (SBS) occur at run-time it is necessary to be able not only to check whether the monitorability of the required SLA terms and conditions is affected by the changes but also to modify the deployed monitoring infrastructure in order to ensure the continuous execution of the required run-time checks. Existing monitoring environments and approaches, however, do not offer these capabilities. To address this gap, SLA@SOI is developing a novel SLA layer monitoring system (SLMS). A key characteristic of this system is the separation of the actual service monitoring from the assessment of SLA monitorability, and the dynamic set up of the monitoring resources (i.e., sensors effectors, and reasoning components) for checking an SLA.

A key characteristic of the approach underpinning the design of SLMS is the distinction between two key layers in service provision, namely the SLA management and service management layers.

- SLA management layer: from a monitoring perspective, it incorporates the mechanisms required for performing the SLA monitorability checks and the dynamic set up of monitoring infrastructures that can enable the monitoring of an SLA.
- Service management layer: it incorporates sensors and reasoning components required for service event capturing and performing the actual SLA checks.

SLMS key components are: Monitoring Manager (MM), Sensors, Effectors, and Reasoning Components.

Monitoring Manager (MM) coordinates the automatically configuration of the monitoring system. It decides, for any SLA it receives, which is the most convenient monitoring configuration according to configurable selecting criteria. A monitoring configuration tells which components to configure and how their configurations. Components receiving monitoring configurations are, typically, Sensors, Effectors, and Reasoning Component Gateways.

The MM check for monitorability function checks whether an SLA is monitorable with respect to the available monitoring system components. The idea underlying the check for monitorability is to delegate, to many reasoners, the monitoring of an SLA in the case in which one reasoner cannot monitor an SLA only. The SLA monitoring can be delegated also in case in which other criteria are taken into account, e.g., the cost of using many reasoners instead one only.

The MM interacts with the POC. The POC requests MM to check for monitorability. All the information required by the MM is passed to it as part of a ServiceBuilder model. In doing so, a query from the MM is not required and the MM simply operates upon the inputs supplied to it by the POC.

For detailed information about MM refer to D.A3b deliverable.

Sensors are components that implement the *SensorInterface* interface. A sensor is a component that gives access to specific infrastructure service information

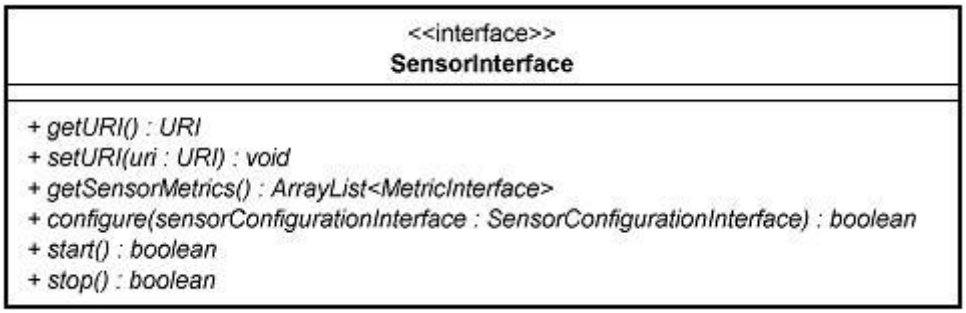


Figure 39: Sensor interface

For instance, let's consider a software service providing a shopping cart for an e-commerce application. One of the operations provided by the service interface is *addItem(Item item):void*. The operation takes as input the item a user want to add to its cart and does not return any value. For monitoring purpose, this method has been instrumented with code for collecting the method's input parameter. The collected data, depending on the instrumentation configuration, can be processed and stored into a database. In the example, the shopping cart service has a configurable instrumentation; the sensor component provides an interface for configuring the service instrumentation, e.g., enabling or disabling the collection of data.

As another example, let's consider the infrastructure service CPU. We are interested in collecting information about its workload. To do that there are many tool available, e.g., the command-line application *top* in UNIX based systems. This application produces information in its own format, therefore, we need to implement a component interacting with the *top* application and collecting the application's output. This component will implement the *SensorInterface* interface.

Reasoning components are monitoring system components implementing the ReasoningComponentGateway (RCG) interface. A RCG mediates the interactions that take place between the SLA@SOI components and the reasoning engine (RE). A RCG provides a set of operations for configuring, starting,

stopping a RE. It also provides operations for retrieving historical monitoring data. A RCG receives monitoring configurations expressed in a *lingua franca* and translates them into RE specific monitoring configurations.

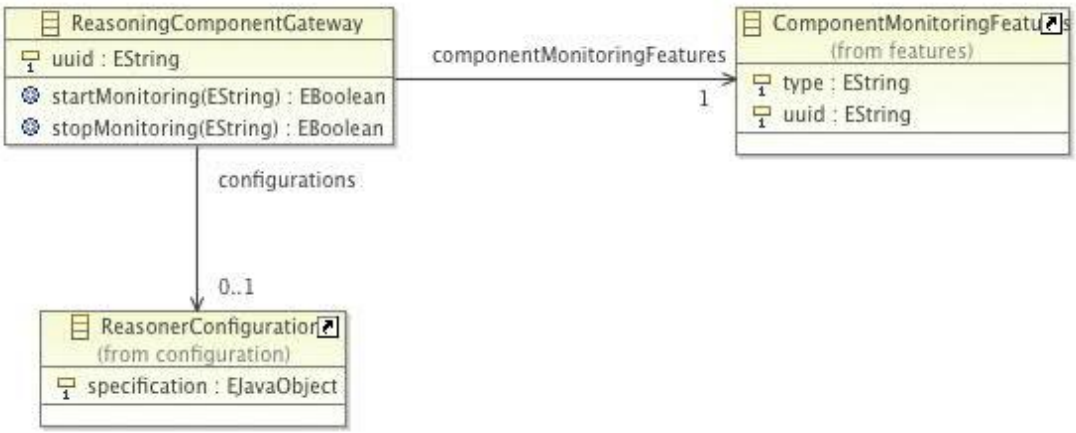


Figure 40: Reasoning component gateway interface

The ReasoningComponentGateway interface exposes the following operations:

```
public ComponentMonitoringFeatures getComponentMonitoringFeatures();
public void setComponentMonitoringFeatures(ComponentMonitoringFeatures
compMonitoringFeatures);
public String getUuid();
public void setUuid(String value);
public ArrayList<ReasonerConfiguration> getConfigurations();
public void removeConfiguration(String configurationId);
public void addConfiguration(ReasonerConfiguration reasonerConfiguration);
boolean startMonitoring(String configurationId);
boolean stopMonitoring(String configurationId);
```

Figure 41: Reasoning component gateway methods

Each RCG has a unique UUID. The UUID is used to retrieve information about a RCG from the software landscape. Each RCG exposes a list of monitoring features through a ComponentMonitoringFeatures object. A monitoring feature expresses the ability to perform an operation or an action.

A RCG receives its configurations via ReasonerConfiguration objects. Each configuration has its own unique configuration identification. The specification field holds the actual monitoring configuration. Ad-hoc translator, if needed can translate this specification to specific monitoring rules. The translation takes place RCG side.

A reasoner configuration already sent to a RCG can be explicitly started/stopped using the startMonitorng/stopMontoring methods that take in input the configuration identification.

7.3 Adjustment

SLAs can be seen as containers of the functional and non-functional properties that both parties, customer and provider, agree specifying its obligations and rights during the service lifetime. However, this also represents a responsibility for the service provider, since it motivates the need of the implementation of an SLA enforcement process. For this reason, the SLA@SOI framework includes a specific component to care about the enforcement aspects of the service: the Provisioning and Adjustment Component (PAC).

Since the quality of the service must be guaranteed at all the levels of the IT stack, this component is placed in all the layers: at software and infrastructure levels, being part of the corresponding SLAManagers, and at the business level as part of the Business Manager. At each level, PACs will receive different inputs, and will trigger different domain-specific actions.

The software and infrastructure PACs subscribe to an event bus (Monitoring Event Channel) in order to receive events from the monitoring system indicating that a SLA has been violated. Upon the reception of these events, the PACs may behave differently: they may take an action on its own given sufficient certainty about the problem and its solution (e.g. reprovisioning of resources, software reconfiguration) or they may request the Planning and Optimization Component to create a new plan, or even to re-negotiate an existing SLA. They also possible inform the business PAC about the problem that has occurred.

On the other hand, the Business PAC receives violations from the lower level adjustment components, and takes decisions based on business criteria, such the re-negotiation of the agreement with the end-customer, the termination of the offered service or even, in a multiprovider environment, may decide to change the 3rd party provider in which it was relying for a given service.

For the implementation of the above described behaviour, the Provisioning and Adjustment Component offers the <<plan>>, <<control>> and <<query>> interactions, and uses <<query>> interface of the SLA registry, <<subscribe_to_event>> of the MonitoringEventChannel, and the <<manage_T_services>> interface offered by the Service Managers. The later is by definition domain specific, and splits into << manage_software_services>> at software level and << manage_infrastructure_services>> at the infrastructure layer.

The **<<plan>> interaction** subsumes the functionality that is necessary for the communication with the Planning and Optimization Component. It includes interfaces for the POC to order the execution of a plan, as well as to inform about the status of the execution of the plan back to the POC. Furthermore, when a problem appears that cannot be solved locally by the PAC, a re-planning can be triggered using the IReplan interface.

The **<<control/track>> interaction** has been designed to take into account business level criteria at the lower levels. For instance, the service provider can decide that it is better to accept violations in some SLAs to give priority to others based on business impact. It is even possible that the service provider decides to prevent breaches for some SLAs or specific QoS metrics while applying reaction upon violations for others. To this end, the <<control>> interaction allows the Business Manager to retrieve the current adjustment policies, as well as to set a new list of policies. On the other hand, the <<track>> interaction allows the communication of violations from the lower software and infrastructure levels to the Business Managers. This allows the business layer to be aware of the

problems in the underlying levels, and to calculate and apply the penalties derived from the malfunctioning of the services under its control.

The **<<query>> interaction** is implemented in the software and infrastructure PACs, and allows an external entity, namely the Business Manager, to query the SLA violations as well as the historical monitoring information. In the second case, the PAC acts as a proxy to retrieve the monitoring data from the Low Level Monitoring System (LLMS) database. This feature is important not only from a business adjustment perspective, but also for the service provider when constructing the reports about the performance of the service.

A more detailed description of the Provisioning and Adjustment Component, also in the context of business, software and infrastructure services, can be found in deliverables D.A2a [17], D.A3a [18], D.A4a [19], and D.A5a [20].

8 Adoption Examples

This chapter serves for illustrating the presented architecture by showing how it can be applied in real-world scenarios.

8.1 ERP Hosting

Background

The Enterprise Resource Planning Hosting (ERP Hosting) solution is targeted at SMEs that cannot afford expensive ERP solutions that include software, hardware and constant support. The service provider provides applications as services (SaaS, Software as a Service) using an online portal. The portal also provides customers with tools for specification of business requirements (providing functional and non-functional information) as well as SLA parameters.

Based on this input, the service provider plans the capacity required to satisfy all requirements, especially the Quality of Service guarantees. Once terms are formally and legally agreed by both entities, the service provider provisions the required infrastructure. It also provides monitoring capabilities for all components (infrastructure, middleware, services) to facilitate appropriate adjustment.

The internal management of the offered SaaS solution is broken down into four separate layers, each of them realizing service offerings of different abstractions and governed via dedicated SLAs. The four layers are:

Solution layer: covering the complete solution offered to the customer including the actual software but also additional support services.

Application layer: covering the offered business applications (software), i.e. the enterprise resource planning functionality.

Middleware layer: covering generic middleware software components such as an application server, database, or enterprise service bus. These can be used across different applications.

Infrastructure layer: covering the actual physical compute infrastructure including virtualized counterparts of resources.

Further details on this industrial use case are provided in [10].

Architecture

The generic framework architecture can be applied for the ERP hosting scenario in the following way. As the use case comprises four different types of services (support, application, middleware, infrastructure) we created dedicated service managers for all of them. From the perspective of the administrative organization, we consider a separation into three different departments, namely business, software and infrastructure department. Consequently, we establish a separated SLA manager for each of these departments. However, for the software part we subdivided the SLA manager into two separate components ones (Application SLA Manager and Middleware SLA Manager) in order to clearly separate the distinct planning and provisioning logic for both areas.

The Business SLA manager is responsible for the overall offering. It uses the support service manager for planning/managing the human support services.

The Application SLA manager is responsible for the SLAs of the complete software application. It uses the Application Service Manager, for managing actual ERP logic and Service Evaluation in order to assess the quality of the different possible software offerings.

The Middleware SLA Manager is responsible for the middleware layer incl. for example the application server.

The Infrastructure SLA Manager is responsible for the in-house IT resources but can also contact external cloud providers to acquire resources for burst scenarios. It uses an infrastructure service manager that deals with all the internal IT resources.

The following figure gives an overview of this adoption architecture. For sake of simplicity we do not show all the actual interaction stereotypes from the generic architecture. Also the manageability and monitoring subsystem is not explicitly shown but considered here as underlying fabric of the service managers.

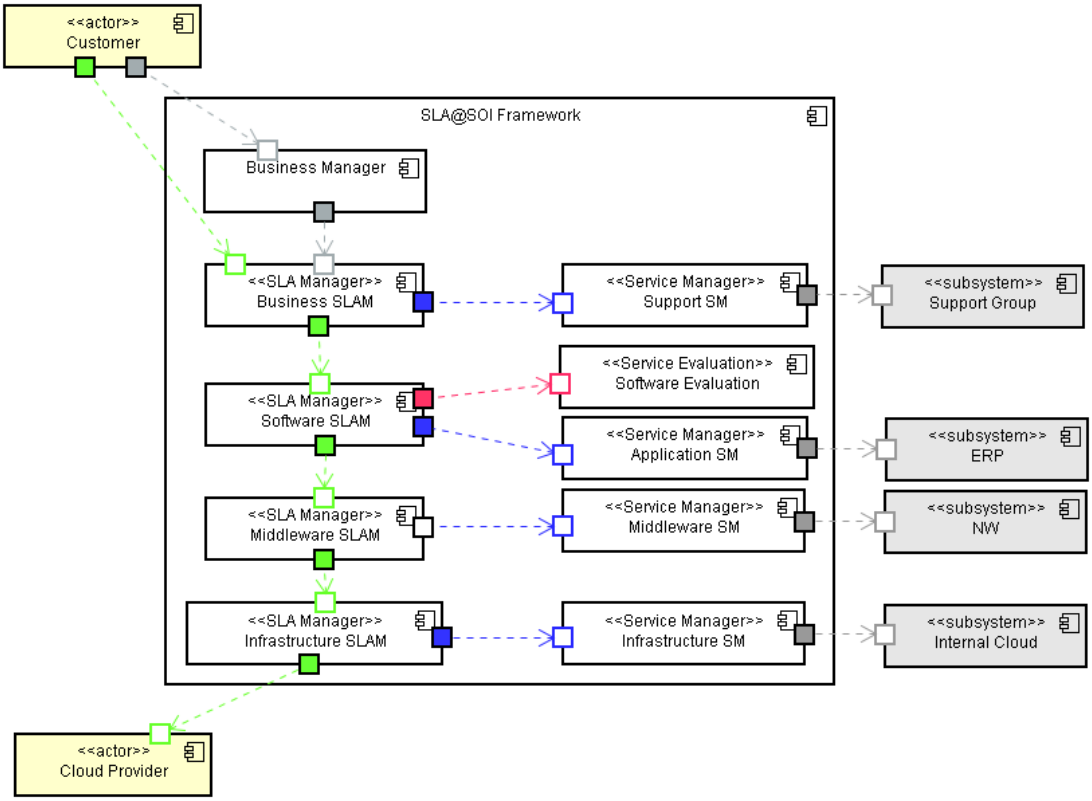


Figure 42: Adopted framework architecture for ERP Hosting use case.

8.2 Enterprise IT

Background

Focusing primarily on Infrastructure as a Service (IaaS), and building on the year one Enterprise IT use case specification [15] the Enterprise IT lab demonstrator aims to address three scenarios, or business challenges, which build on each other and represent real barriers for the enterprise today. Using Key Performance Indicators (KPIs) we evaluate the performance of the lab demonstrator in the areas of:

- IT enabling the Enterprise
- IT Efficiency
- IT Investment/Technology adoption

The three scenarios are briefly re-introduced here.

The first scenario, titled "Provisioning", responds to the issue of efficient allocation of new services on IT infrastructure, SLA negotiation and provisioning of new services in the environment. The second scenario, "Run Time", deals with day-to-day, point in time operational efficiency decisions within the environment. These decisions maximise the value from the infrastructure investment. The final scenario, "Investment Governance" builds on the first two to demonstrate how they feed back into future business decisions. Taking a holistic cost view, it provides fine-grained SLA based data to influence future investment decisions based on capital, security, computational power and energy efficiency.

In year two of the project we have focused on the implementation of a hardware test bed on which to demonstrate and evaluate the effectiveness of the framework in an Enterprise IT setting.

Architecture

The generic framework architecture can be customised and applied to the Enterprise IT use case as shown in Figure 43.

Interaction with the system is achieved through a web based UI where both customers and administrators have an interface to the framework components. The Enterprise IT SLAT defines use case specific agreement terms that are loaded by the SLA manager to provide the inputs to provisioning requests. In year 2, provisioning requests consist of PaaS services. Therefore, the use case does not need to implement the software SLA Manager. Software services are of course possible by choosing a virtual machine template that contains pre-loaded applications, but the quality of service of these is not part of the agreement terms in year 2.

The SLA Manager passes service-provisioning requests to the Infrastructure SLA Manager whose role is to carry out the creation of the new virtual machines that constitute the service along with monitoring and reporting for that service.

In year 2, the use case has implemented the basic functionality required for the provisioning scenario and some of the features needed to evaluate the runtime scenario. The architecture will be extended in year 3 to realise the remaining features, including the creation of plug-ins to support additional heterogeneous hypervisors, such as VMWare.

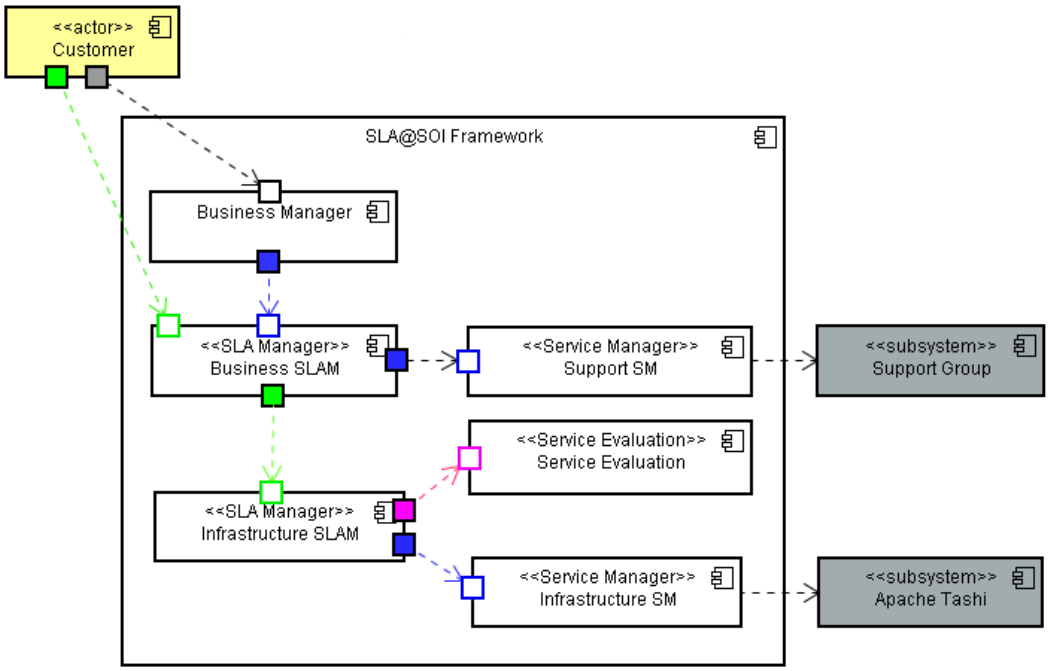


Figure 43: Adopted framework architecture for Enterprise IT use case.

Further details on this industrial use case are provided in [11].

8.3 E-Government

Background

The eGovernment use case applies the SLA@SOI architecture to the management of a hybrid service, which involves both automatic and human based activities, as it is typical in the Government domain.

The considered service allows the citizens to receive medical treatments and transport means to move from home to the treatment place, and to book them by phone.

Such a Mobility & Health Service is provided by the so called "Citizen Service Center", placed in the Italian region of Trentino, and is regulated by a SLA with the local Government ("Provincia Autonoma di Trento"). The service is composed by a Contact Service provided by an internal department of the Citizen Service Center that allow the citizen to book the treatment and trips, from Medical Treatment Services provided by several external Health Care Structures and from Mobility Services provided by specific providers. The implementation of the Contact Service is based on internal phone call operators (human operators or automatic answering machines) and, when needed, also on external Call Centers.

The Government chooses the possible Health Care Structures and Mobility Providers, and their services are regulated by SLAs with the Government. The specific provider used at each execution of the Mobility & Health Service is selected by the Citizen Service Center on the base of the Citizen's choices and preferences.

The business relationship with the external Call Centers is under the complete control of the Citizen Service Center, which directly establishes the SLA with the selected Call Center.

Therefore, the Citizen Service Center is responsible for selecting, coordinating, and monitoring the component services needed to provide the Mobility & Health Service, under the constraints established by the SLAs extablished by the Government. The monitoring activity includes the assessment of the Citizen satisfaction that is also performed using human operators provided by the internal department or by the external Call Centers. The involved SLAs includes the payment of penalties in case of violations of the constraints.

The Citizen Service Center wants to adopt an SLA@SOI based software application in order to automate activities that currently are performed manually, such as the coordination between the treatment and mobility service, the aggregation of monitoring data to evaluate the quality of the composed service, the allocation of the phone call operators, the selection of the mobility providers based on the matching between the Citizen request and the SLA established with the providers, the management of the requests of renegotiations coming from the Government, by planning the resources needed by the internal department and re-negotiating SLA with the external Call Center.

The application offers specific dashboards that report the satisfaction of their SLAs and the penalties to pay by each part to both the Government and to the Internal Department of the Citizen Service Center. Moreover an interface is offered to the Government to start a renegotiation, and another interface is offered to the internal department of the Citizen Service Center to see the resource planning and allocation established by the system.

The Health Care Structures and Mobility Providers provide sensors for monitoring their services. Any external Call Center offers a negotiation interface that is compliant with the SLA@SOI standard, but no one of the external providers is required to adopt the SLA@SOI framework.

Further details on this industrial use case are provided in [13] .

Architecture

The generic framework architecture can be applied for the eGovernement use case as shown in Figure 44.

The coordination of Health & Mobility Service is performed through a BPEL process that is managed by a corresponding Software Manager; in particular it controls the dynamic binding of external service providers. A corresponding Human Service Manager manages the human resources.

While the Health & Mobility Service involves both human and software components, the quality constrained in the SLAs does not depend in a significant way from the quality of the software or infrastructure hardware components. Therefore, one single SLA Manager at business level is sufficient to manage the negotiation of the Government and with the external Call Center and to manage the SLAs with the other external providers.

Moreover the SLA Manager uses a Service Evaluation Component to predict at negotiation time the quality of the service obtainable with the available internal resources in order to determine the SLA to negotiate with the external providers.

The runtime prediction feature of the monitoring system is used during the execution of the service to warn about possible violation of the guarantee terms in the SLA of the Contact Service and to trigger the automatic adjustment of internal human operators.

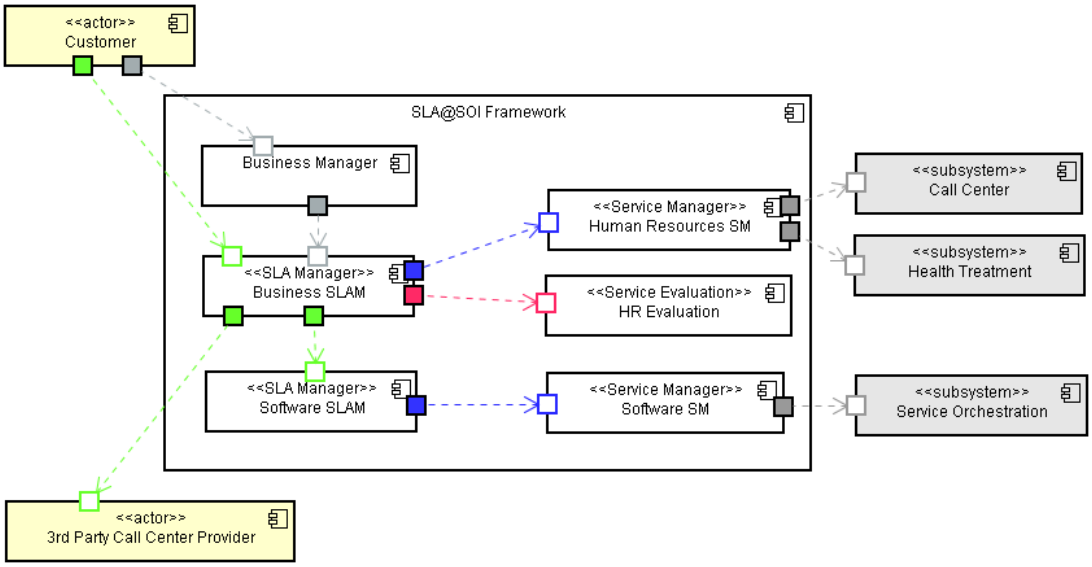


Figure 44: Adopted framework architecture for eGovernment use case.

9 Conclusions

9.1 Summary

This document presented the reference architecture for the SLA management framework that has been created within the second year of the European research project SLA@SOI.

The reference architecture has been developed based on the requirements of four industrial use cases as well as other (project-external) sources (see also [15]).

The architecture constitutes a clear innovation in its ability to mediate across layers, to support arbitrary domains, and to address the complete SLA and service lifecycle.

This document provided a complete overview of the framework architecture including a high-level overview, a detailed discussion of relevant foundational concepts, a description of the modelling foundation i.e. the most important metamodels that are shared between different components of the architecture, and the actual architecture overview with its building blocks, components and interactions.

The overall architecture is well synchronized and consistent with the more detailed architectures of the various main framework modules. Details on these module-specific aspects can be found in deliverables D.A2a-D.A6a.

The technical architecture has been successfully implemented into an SLA management software framework. Details about the technical/scientific evaluation of the SLA framework can be found in deliverable D.B1a.

Finally, the software framework has been successfully applied in four industrial use cases.

9.2 Further reading

This document serves as formal deliverable D.A1a for reporting the work progress of work package A1 at project month 26. It is complemented by the deliverable D.A1b which provides the actual implementation of the SLA framework. Further insight into specific modules of the overall framework is given within the context of the other scientific work packages A2-A6 via deliverables D.A2a-D.A6a. The technical/scientific evaluation of the SLA framework is subject to deliverable D.B1a, and to the deliverables of the various use cases D.B2a-D.B7a.

9.3 Outlook on Future Work

Following the broadness of the specified architecture and its applicability to four industrial use cases we expect that the developed architecture will remain largely stable for the next project phase.

Of course adjustment and optimization will be incorporated into the architecture, as these become clear in the detailed evaluation conducted by use cases but also based on ongoing external feedback that is collected.

Significant work is definitely needed to support framework adopters. Therefore, we plan to invest quite some effort into the proper documentation of the architecture as well as into guidelines for its application.

10 References

[1] SLA@SOI Source Forge project

[2] SLA@SOI project: *Deliverable D.A1a Framework architecture (full lifecycle).* July 2010. URL http://www.sla-at-soi.eu

[3] SLA@SOI project: *SLA@SOI Glossary*. Annex to deliverable D.A1a Framework architecture (full lifecycle). July 2010. URL http://www.sla-at-soi.eu

[4] SLA@SOI project: *Deliverable D.A1a Framework architecture – adhoc architecture*. September 2009. URL http://www.sla-at-soi.eu

[5] SLA@SOI project: *Deliverable D.A1b Integrated Framework – full lifecycle*. September 2010. URL http://www.sla-at-soi.eu

[6] W. Theilmann, R. Yahyapour, J. Butler, "Multi-level SLA Management for Service-Oriented Infrastructures", In: Proc. of ServiceWave 2008 conference, 10.12.-13.12.2008, Madrid, URL: http://www.servicewave.eu, P. Mähönen, K. Pohl, and T. Priol (Eds.): ServiceWave 2008, LNCS 5377, pp. 324–335, 2008, Springer-Verlag Berlin Heidelberg 2008

[7] Marco Comuzzi, Wolfgang Theilmann, Gabriele Zacco, Christoph Rathfelder, Constantinos Kotsokalis, and Ulrich Winkler: *A Framework for Multi-level SLA Management*. Proc. of 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing, NFPSLAM-SOC'09, November 23, Stockholm, Sweden, co-located with 7th International Conference on Service Oriented Computing (ICSOC 2009), "Quality of Service" workshop track ICSOC paper

[8] Wolfgang Theilmann, Jens Happe, Costas Kotsokalis, Andy Edmonds and Keven Kearney: A Reference Architecture for Multi-Level SLA Management. Journal of Internet Engineering (JIE), to appear

[9] ITIL (IT Infrastructure Library): *ITIL Version 3 - Service Improvement*. URL: http://www.itil-officialsite.com

[10] SLA@SOI project: *Deliverable D.B3b Lab Demonstrator ERP Hosting*. September 2010. URL http://www.sla-at-soi.eu

[11] SLA@SOI project: *Deliverable D.B4b Lab Demonstrator Enterprise IT*. September 2010. URL http://www.sla-at-soi.eu

[12] SLA@SOI project: *Deliverable D.B5b Lab Demonstrator Service Aggregator*. September 2010. URL http://www.sla-at-soi.eu

[13] SLA@SOI project: *Deliverable D.B6b Lab Demonstrator eGovernment*. September 2010. URL http://www.sla-at-soi.eu

[14] SLA@SOI project: *State of the Art Analysis.* Annex of: *Deliverable D.B1a Scientific Evaluation Report.* September 2010. URL http://www.sla-at-soi.eu

[15] SLA@SOI project: *Deliverable D.B1a Scientific Evaluation Report*. September 2010. URL http://www.sla-at-soi.eu. URL http://www.sla-at-soi.eu

[16] SLA@SOI B4 M17 Use Case Specification Deliverable. October 2009. URL http://www.sla-at-soi.eu

[17] SLA@SOI project: *Deliverable D.A2a Business SLA management*. September 2010. URL http://www.sla-at-soi.eu

[18] SLA@SOI project: *Deliverable D.A3a SLA-aware service management*. September 2010. URL http://www.sla-at-soi.eu

[19] SLA@SOI project: *Deliverable D.A4a SLA-aware infrastructure management*. September 2010. URL http://www.sla-at-soi.eu

[20] SLA@SOI project: *Deliverable D.A5a Foundations for SLA-management*. September 2010. URL http://www.sla-at-soi.eu

[21] SLA@SOI project: *Deliverable D.A6a Predictable/Manageable Service Engineering Methodology and Prediction Services*. September 2010. URL http://www.sla-at-soi.eu

[22] Wikipedia: *Builder Pattern*. http://en.wikipedia.org/wiki/Builder_pattern

[23] Wolfgang Theilmann, Gabriele Zacco, Marco Comuzzi, Christoph Rathfelder, Constantinos Kotsokalis, and Ulrich Winkler: *A Framework for Multi-level SLA Management*. Accepted chapter in: Handbook of Research on Non-Functional Properties for Service-Oriented Systems: Future Directions. Planned publication: 2011

Appendix A: Glossary

The following list shows the most important entries of the SLA@SOI glossary. Note that terms that are specific for the current document and not part of the overall project wide glossary are marked with an asterix *.

Agreement Initiator	An agreement initiator is a party to a *service level agreement*. The initiator creates and manages an agreement on the availability of a service on behalf of either the service customer or service provider, depending on the domain-specific signalling requirements.
Agreement Offer	An offer is the description of the agreement relationship that is sent from *agreement initiator* to *agreement responder* during agreement creation, indicating the relationship which the initiator would like to form.
Agreement Responder	The agreement responder is a party to a *service level agreement*. The responder implements and exposes an agreement on behalf of either the service provider or service customer, depending on the domain-specific signalling requirements.
Agreement Template	An agreement template is an XML document used by the *agreement responder* to advertise the types of offers it is willing to accept.
Agreement Term	Agreement terms define the content of a *service level agreement*.
Business Service	A business service is exposed/invoked via at least some non IT elements.
Business Manager	A specialization of *service provider*: person that defines the SLATs of products and joins available services in a product.
External Service	External services are exposed across the boundaries of an organization, i.e. across at least two administrative domains.
Framework Administrator	A specialization of *service provider*: person that configures/adapts the SLA@SOI framework for a specific application.
Guarantee Term	Guarantee terms define the assurance on service quality associated with the service described by the service definition terms. They refer to the service description that is the subject of the agreement and define service level objectives, qualifying conditions and business value expressing the importance of the service level objectives.
Hybrid Service	A hybrid service is a set or bundle of other services where all these services are exposed to the customer but have different service interface types (e.g. an IT service and a business service).
Infrastructure Manager	A specialization of *infrastructure provider*: person/system that is interested to measure and control infrastructure properties.
Infrastructure Provider	A specific kind of service provider that focuses on the provisioning of *infrastructure services*.

Infrastructure Service	An infrastructure service is a specific *IT service* which exposes resource/hardware-centric capabilities.
Internal Service	Internal services are exposed within the boundaries of an organization, i.e. within one administrative domain.
IT Service	An IT service is exposed/invoked by means of information technology. Specific classes of IT services may be software services, infrastructure services or media services.
Offered Service	An abstract service (more precisely: service type) which is offered by a specific *Service Provider* to its *Service Customers*.
Operation Level Agreements	A specification of the conditions under which an *internal service* or a component is to be used by its "customer".
Service	A means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks. See also *service interface type*, *service concreteness*, *service exposure*
Service Concreteness	The stage a service reaches over time from a fully abstract type to actually instantiated. See also *service type*, offered *service*, *service implementation*, *service instance*
Service Consumer	Person(s) who actually consume/use the provided services. Typically they belong to the *service customer*.
Service Customer	Someone (person or group) who orders/buys services and defines and agrees the service level targets.
Service Description Term	Service Description Terms describe the functionality that will be delivered under the *service level agreement*. The agreement description may include also other non-functional items referring to the service description terms.
Service Exposure	Services can be exposed either internally (within the same administrative domain) or externally. See also *internal service*, *external service*
Service Implementation	A service implementation is a possible concrete realization of a given *service type*.
Service Instance	A concrete realization of an *offered service* which is ready for consumption by service users. It relies on the instantiations of all the resources required for a given *service implementation*.
Service Interface Type	Describes the nature of an actually exposed service, i.e. about the nature of his invocation interface. See also *business service, IT service*, *hybrid service*
Service Level Consequence	An action that takes place in the event that a service level objective is not met.
Service Level Agreement	An agreement defines a dynamically-established and dynamically managed relationship between parties. The object of this relationship is the delivery of a service by one of the parties within the context of the agreement. The management of this delivery is achieved by agreeing on the respective roles, rights and obligations of the parties. The agreement may specify not only functional properties for identification or creation of the service, but also non-functional properties of the service such as performance or

	availability. Entities can dynamically establish and manage agreements via Web service interfaces.
Service Level Objective	Service Level Objective represents the quality of service aspect of the *agreement*. Syntactically, it is an assertion over the agreement *terms* of the agreement as well as such qualities as date and time.
Service Provider	An organization supplying services to one or more internal customers or external customers.
SLA Manager	A specialization of *service provider*: person/system that is responsible for managing SLATs and SLA relationships.
Software Designer	A specialization of *software provider*: person that designs/develops the architecture and components of a specific SLA based application.
Software Manager	A specialization of *service provider*: person that defines software-based services, takes care of their management and supports the SLA manager in creating appropriate SLA templates.
Software Provider	An organization producing *software components* which might be used by a *service provider* to assemble actual *services*.
Software Service	A software service is a specific *IT service* which is exposed/invoked by means of software entities such as Web services, user interfaces, or software-based business processes.
Software Component	Software components are the entities produced at design-time by a *software provider*.
Service Type	A service type (or abstract service) specifies the external interface of a service possibly including non-functional aspects. It does not specify any means (components, resources) which are needed for the actual provisioning of that service.

Appendix B: Abbreviations

AOP	Aspect Oriented Programming
BM	Business Manager
B-SLAM	Business SLA Manager
EMF	Eclipse Modelling Framework
ERP	Enterprise Resource Planning
IE	Interaction Event
FCR	Finite capacity regions
Infr-SLAM	Infrastructure SLA Manager
Infr-SM	Infrastructure Service Manager
IoC	Inversion of Control
KPI	Key Performance Indicator
LLMS	Low Level Monitoring System
LQN	Layered Queueing Networks
MA	Manageability Agent
MRE	Monitoring Result Event
MVC	Model View Controller
NFP	Non-functional property
ORC	Open Reference Case
OVF	Open Virtualization Format
QoS	Quality of Service
QPN	Queueing Petri Nets
PAC	Provisioning and Adjustment Component
POC	Planning and Optimization Component
POJO	Plain Old Java Objects
SaaS	Software as a Service
SE	Service Evaluation
SLA	Service Level Agreement
SLAM	SLA Manager
SLAT	Service Level Agreement Template
SM	Service Manager
SME	Small and Medium-sized Enterprise
SOA	Service Oriented Architecture
SW-SLAM	Software SLA Manager
SW-SM	Software Service Manager
TCO	Total Cost of Ownership
TOGAF	The Open Group Architecture Framework