

# A Reference Architecture for Multi-Level SLA Management

Jens Happe, Wolfgang Theilmann, Andrew Edmonds, and Keven T. Kearney

**Abstract** Service-orientation is the core paradigm for organising business interactions and modern IT architectures. At the business level, service industries are becoming the dominating sector in which solutions are flexibly composed out of networked services. At the IT level, the paradigms of Service-Oriented Architecture and cloud computing realise service-orientation for both software and infrastructure services. Service composition across different layers is a major advantage of this paradigm. Service Level Agreements (SLAs) are a common approach to specifying the exact conditions under which services are to be delivered, and thus are a prerequisite for supporting the flexible trading of services. However, typical SLAs are only specified at a single layer and do not provide insight into metrics or parameters at the various lower layers of the service stack. Thus they do not allow service providers to manage their service stack optimally.

In this chapter, we present a reference architecture for a multi-level SLA management framework. We discuss fundamental concepts of the framework and detail its main architectural components and interactions.

---

Jens Happe

SAP Research, Vincenz-Priessnitz-Str.1, 76131 Karlsruhe, Germany,  
e-mail: Jens.Happe@sap.com

Wolfgang Theilmann

AP Research, Vincenz-Priessnitz-Str.1, 76131 Karlsruhe, Germany,  
e-mail: Wolfgang.Theilmann@sap.com

Andrew Edmonds

Intel Ireland Limited, Collinstown Industrial Park, Leixlip, Ireland,  
e-mail: andrewx.edmonds@intel.com

Keven T. Kearney

Engineering Ingegneria Informatica spa, Via Riccardo Morandi, 32, 00148 Roma, Italy,  
e-mail: keven.kearney@eng.it

## 1 Introduction

*Service Level Agreements* (SLAs) are a common way to formally specify the exact conditions (both functional and non-functional) under which services are or should be delivered. However, in practice, SLAs are only specified at the top-level interface between a service provider and a service customer. Customers and providers can use top-level SLAs to monitor whether their actual service delivery complies with the agreed SLA terms. In the case of SLA violations, top-level SLAs allow for penalties or compensations to be directly derived.

In a service-oriented world, services offered are (usually) composed of or built on a complete set of other services. These services may reside in the domain of the provider itself, or be hosted by external providers. Such services include business services, software services, and infrastructure services. The quality of an offered service depends heavily on the quality of the services it uses. Service quality also depends on the elements used and the structure of the underlying IT system realising the service. Currently, service providers cannot plan their service landscapes using the SLAs of dependent services. They have no means by which to determine why a certain SLA violation might have occurred, or how to express an associated penalty. SLA guarantee terms are not explicitly related to measurable metrics, nor is their relation to lower-level services clearly defined. As a consequence, service providers cannot determine the necessary (lower-level) monitoring required to ensure top-level SLAs. This missing relationship between top-level SLAs and (lower-level) metrics is a major hurdle to efficient service planning and prediction or adjustment processes in service stacks.

In this chapter, we present a reference architecture for a multi-level SLA management framework. The framework was built based on previous discussion of a purely conceptual architecture [1] and experimental analysis of a specialised showcase [2]. We also present underlying concepts of the architecture and its main building blocks (components and interactions).

The remainder of this chapter is organised as follows: Section 3 describes foundational concepts, Section 4 introduces the developed reference architecture, and Section 7 concludes with a brief summary and outlook.

## 2 Scope and Goals

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 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 that business entity.

The primary goal of the 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 organisational domains;
2. supports arbitrary service types (business, software, infrastructure) and SLA terms;
3. covers the complete SLA and service life cycle 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.

To achieve these goals, the reference architecture is based on three main design principles: First, we put a strong emphasis on clearly separating service management from SLA management and supporting a well-layered and hierarchical management structure.

Second, we included a solid foundation common to meta-models for SLAs, as well as their relation to services and the construction of actual service instances; such a foundation is essential to supporting clear semantics across the different framework components. Third, since design for extensibility and adaptability is key to addressing multiple domains, we clearly distinguished between generic solution elements and places where domain-specific logic/models need to be provided. We aim to achieve an architecture where even generic parts can be replaced by domain-specific versions, perhaps dictated by preexisting (legacy) management functionality.

### 3 Foundation Concepts

Before diving into the actual reference architecture, we will detail some of its fundamental concepts surrounding the notion of SLAs and the relationships between them. This includes the definition of SLA management, service and SLA life cycles, the SLA (template) model, and the service construction model (SCM).

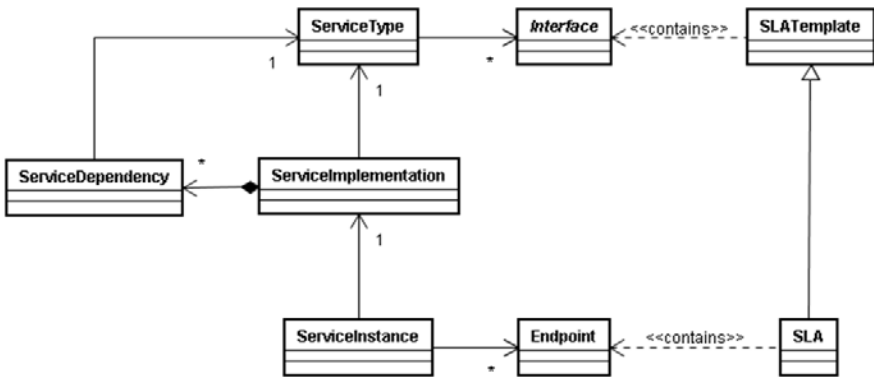
#### 3.1 Service Hierarchy

A first and fundamental concept for the architecture is the refinement of services into three specialisations, with respect to their concreteness:

- *Service Type*: Specifies the service as a fully abstract entity via its external interface.
- *Service Implementation*: Describes specific resources or artifacts (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*: Describes 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 and control). Service instances might have multiple service/management endpoints if their service type specifies a bundled service.

As an example of these concepts, we can take a database service: The abstract type of such a service is specified in that it 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 service instances might be created and these may differ in their concrete configuration. For example, one instance might be configured for optimised read access, another one for fast write access.



**Fig. 1** Service concepts and their relations to each other.

Figure 1 shows the main concepts coined for services and SLAs, as well as their relationships. SLA templates specify the types of SLA offerings a service provider 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 the service types that a given implementation relies on. To instantiate higher-level services, these dependencies must be resolved into concrete instances of the depending sub-service.

## 3.2 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.

### 3.2.1 SLA Management

The term management is interpreted here as "control" - in the classic control theory sense; synonymous with "applied constraint". We assume that the management actions enacted by the manager upon a system are goal-based; that is, that they serve to satisfy one or more management objectives, which are in some way dependent on the state of the system. To this end, the management relation is necessarily bi-directional: the application of management entails a continuous feedback loop in which the manager observes the dynamic state of the system, and acts upon it to constrain its state dynamics in some way. Both observation (sensing) and action are necessarily mediated by information exchange. Finally, management systems can be:

- *hierarchically organised* in that 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 to meet the quality of service (QoS) objectives (goals) specified in SLAs. SLA management covers all stages in the SLA lifecycle:

- *SLA template design* ensures that offered QoS guarantees are realistic;
- *SLA negotiation* ensures that agreed QoS guarantees are realisable;
- *SLA runtime* ensures that QoS guarantees are satisfied; and
- *SLA (template) archiving* ensures that previous experience is available to future cycles.

### 3.2.2 Service Life cycle

The management of SLAs happens in the context of the overall service life cycle (Figure 2), which consists of the following stages:

- *Design and development*: development of artifacts needed for service implementation
- *Service offering (including SLA template design)*: offering a service (type) to customers; results include specification of SLA templates
- *Service negotiation (including parts of SLA negotiation)*: actual negotiation between customer and provider; results in an agreed SLA

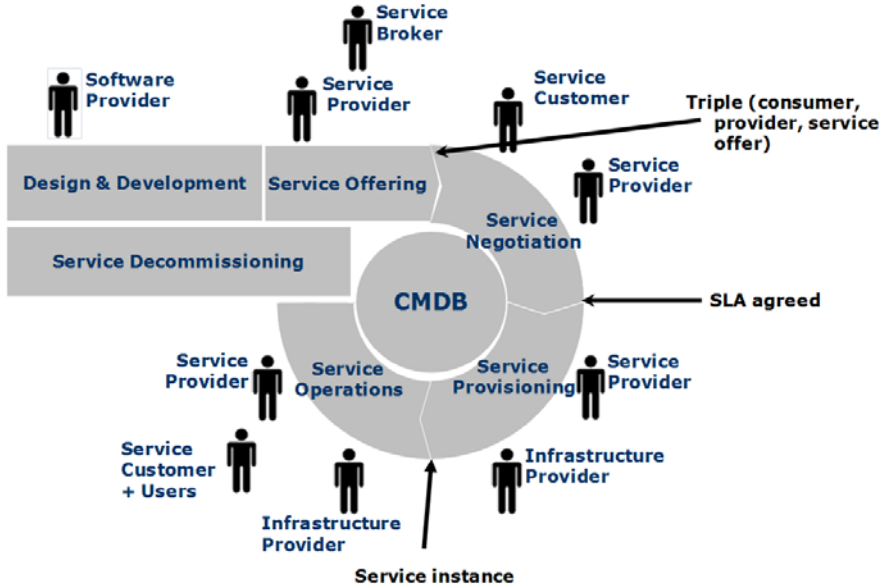


Fig. 2 Service life cycle.

- *Service provisioning (including 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 (including SLA runtime):* an actual service instance is up and running; it might be adjusted to enforce an SLA
- *Service decommissioning:* the service instance is stopped and can no more be accessed by the service customer

### 3.2.3 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 organisational coherence: for example, an independent organisation or a department that operates largely as a profit centre. Within an administrative domain, two main views can be considered:
  - The business view basically representing a sales department: that is, the activity of selling services via SLAs.

- The management view oversees all the offered or active SLAs within a certain domain and is responsible for the eventual SLA operation.
2. *Technical domains* are areas where certain kinds of resources or artifacts can be coherently managed: for example, domains for infrastructure artifacts, software artifacts, business artifacts or even subdivisions of these.

Technical domains can be understood as horizontal layers within a business/IT stack, while administrative domains relate more to vertical, cross-cutting pillars within an organisation (though they can form a hierarchy as well). Section 4 gives insight into how the notion of domains impacts the architecture.

### 3.3 Data Models

To communicate, the components of the SLA@SOI architecture make heavy use of two models that reflect the essential data structures in the system: The *SLA(T) model* (Chapter ‘The SLA Model’) describes SLAs for communication within and among SLA managers, as well with external providers. The *Service Construction Model* (SCM) (Chapter ‘The Service Construction Meta-Model’) provides and collects information necessary to create a new instance of a service (for a particular SLA). In the following, we provide a brief summary of both models.

#### 3.3.1 SLA(T) model

The SLA and SLA template model (SLA(T) model) extends pure functional service descriptions to allow for the expression of non-functional service properties and QoS guarantees. The main body of the present specification defines the SLA(T) model, which allows the expression of non-functional service properties and QoS guarantees. The SLA(T) model leaves the specification of particular QoS terms open, supporting extension through standard vocabularies. A set of default QoS terms are provided as a standard vocabulary. An SLA is a set of agreements between two (or more) parties. These agreements are expressed using terms that each denote guarantees made by, or obligations on, the various parties.

#### 3.3.2 Service Construction Model (SCM)

The SCM is inspired by the management of services and the SLA concepts sketched above. We introduced the SCM to ease communication between the different components responsible for core SLA management, service management, and quality evaluation of possible service offerings. Its core structure is the service hierarchy introduced in Section 3.1.

Service managers can use the SCM to manage multiple implementations of the same service. Further, the SCM allows different components to access and add information about a potential service instance.

In the following, we describe the architecture of the SLA management framework and how it makes use of the concepts presented in this section.

## 4 Architecture

### 4.1 Building Blocks

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

#### Overview

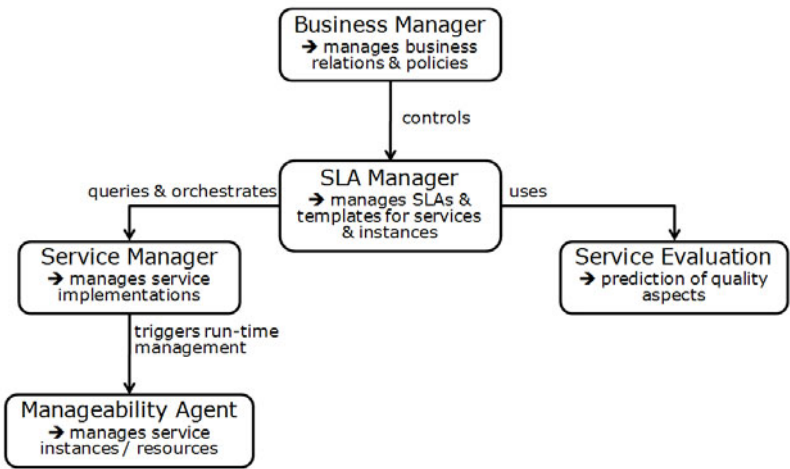


Fig. 3 Generic building blocks and their relations.

Figure 3 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 imple-



mentation. It uses manageability agents to trigger runtime management activities. The SLA manager also relies on 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 to meet business objectives and maintain customer and provider relations. To that extent, it captures knowledge about pricing schemes (including rewards, promotions and discounts), customer profile information, third-party service provider profiles and business rules for taking cost/profit-aware decisions. Business managers may contain sensitive data that must not be shared among components. The actual functionality of a business manager includes:

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

The *SLA manager* component is responsible for managing a set of SLA templates and SLAs in its domain. It also captures knowledge about negotiation and planning goals (such as utility functions or policies). Depending on the specific context/requirements of a particular use case, a separate SLA manager may be set up for a complete organisation, a 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 third parties, including conversion between different SLA formats
- SLA planning and optimisation
- 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, including 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 collection 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 proactive management decisions at all levels, the framework also provides a *service evaluation* component. It relies on background information (from design-time, runtime or historical information) about the quality characteristics of services. It provides functionality for *a priori* quality evaluations of services, depending on influencing factors such as customer behavior 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 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 third parties must be commonly understood by business managers and SLA managers.

## Component Setups

As stated in the design goals for the framework architecture (Section 2), a key goal is support for flexible configurations and setups where different domain cuts can be realised.

Our architecture supports cuts along the two main criteria mentioned in Section 3.2: that is, via administrative domains and technical domains. Administrative domains are characterised by having a dedicated SLA manager in charge of all the SLAs within that domain. Technical domains are characterised by having a dedicated service manager in charge of all the artifacts needed for a (set of) service implementation(s).

Figure 4 shows an example of how such a domain cut can be realised for a single service provider organisation that interacts with customers and third parties. Basically, the service provider organisation is split into two main administrative domains: one might be for application services, the other for infrastructure services.

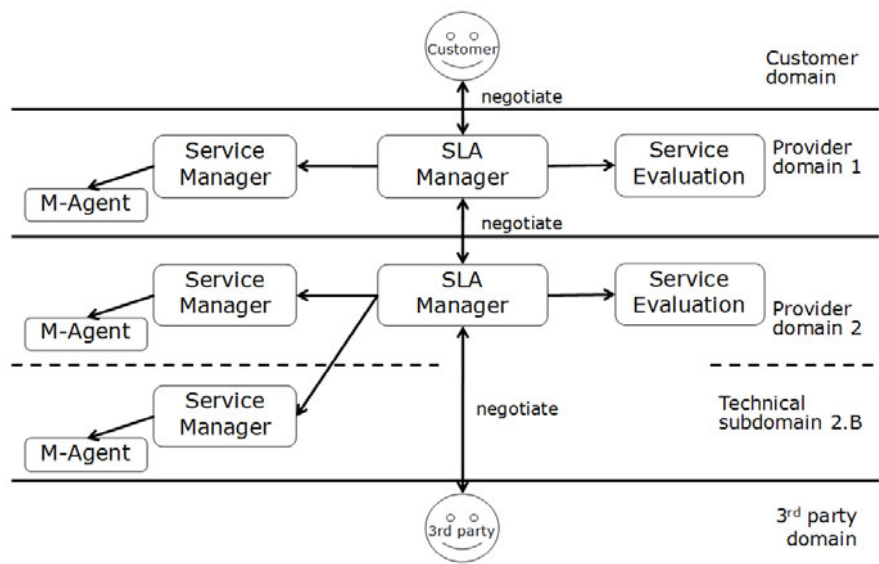


Fig. 4 A possible domain split and component setup.

Further, there is a split into three technical domains, each represented by a service manager. For example, one might be for application artifacts, one for middleware artifacts, and one for infrastructure artifacts.

4.2 Top-Level Architecture

In the following, we present more detail of the top-level view of the SLA@SOI framework architecture. 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 2 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 managers 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 status of the service instance. To achieve a good generalisation of the framework architecture, several components are realised as specialisations of

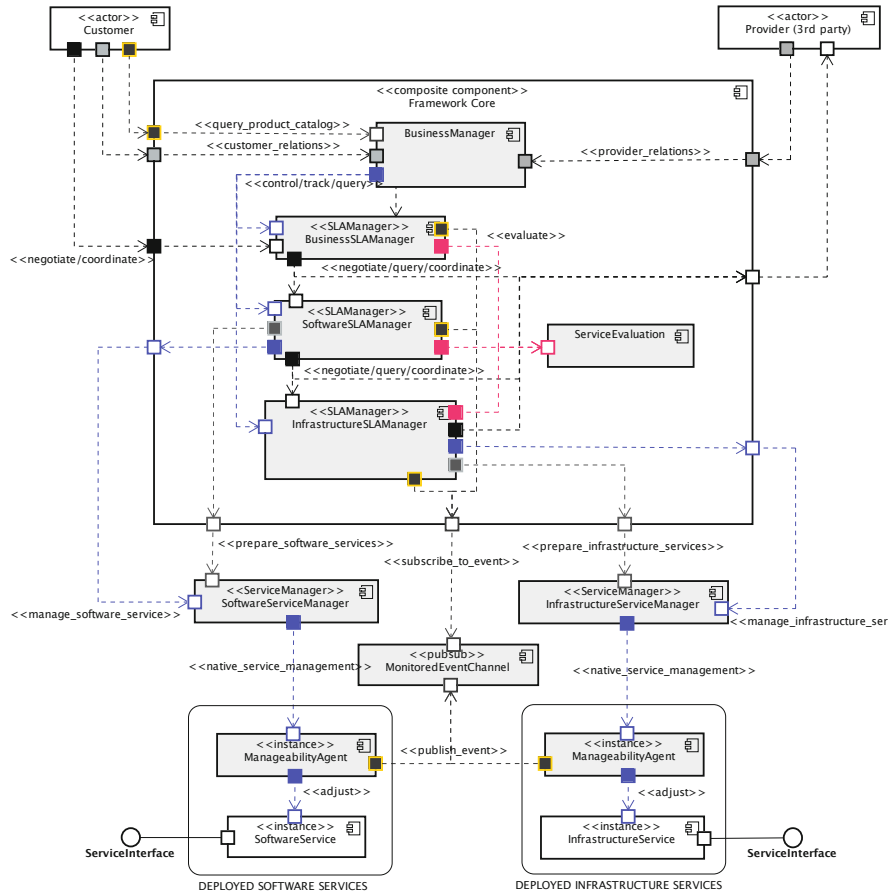


Fig. 5 Top-level view of the SLA@SOI framework.

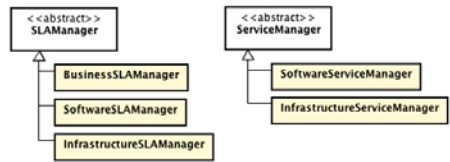


Fig. 6 Component specialisation hierarchies.

abstract components, namely SLA manager components and service manager components. The component hierarchy assumed for the top-level view is depicted in Figure 6.

### 4.3 Main Components

The top-level architecture comprises the building blocks introduced in Section 4.1. In the following, we describe the specific setup as well as the interactions shown in Figure 2.

The *business manager* is responsible for controlling all business information and communication with customers (`<<customer_relations>>`) and providers (`<<provider_relations>>`). For example, it realises the customer relationship management (CRM) necessary to efficiently sell the offered services. Further, the business manager governs the (business-, software-, and infrastructure-) SLA managers (`<<control/track/query>>`). For this purpose, SLA managers must adhere to business rules defined by the business manager (*control*) and must inform the business manager about their current status and activities (*track*).

The (*business-, software- and 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 specialisations of an abstract generic SLA manager (Figure 6). SLA managers are responsible for the negotiation of SLAs, and for the management of services that are subject to the SLAs. All SLA managers can act as "service customers", negotiating SLAs with other SLA managers inside the same framework, or with external (third 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 negotiations are equally used at business-level for 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.

The *monitoring and adjustment management system* provides the underlying fabric across the different layers (i.e. across software and infrastructure layers), 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 realised via the `<<publish_event>>` and `<<subscribe_to_event>>` interaction stereotypes. Manageability agents support the actual configuration and

management of service instances. Access to manageability agents for SLA managers is always mediated via a specific service manager. Due to their domain-specific nature, the interactions between service managers and manageability agents are 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 that 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.

## 5 Conclusions

In this chapter, we presented a reference architecture for multi-level SLA management that supports the comprehensive management of possibly complex service stacks. SLAs are used for managing the nonfunctional aspects of the complete service life cycle. Further, SLA translations across different layers allow for consistent interlinking of complete service networks and hierarchies. The presented architecture is based on experiences gained from an SLA framework built around a specific reference application. The main achievement when compared to that work is the generalisation of concepts such that the architecture can serve a large variety of domains.

## References

- [1] W. Theilmann, R. Yahyapour, and J. Butler, *Multi-level SLA Management for Service-Oriented Infrastructures*. Towards a Service-Based Internet (2008) 324–335
- [2] M. Comuzzi, C. Kotsokalis, C. Rathfelder, W. Theilmann, U. Winkler, and G. Zacco, *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

<http://www.springer.com/978-1-4614-1613-5>

Service Level Agreements for Cloud Computing

Wieder, P.; Butler, J.M.; Theilmann, W.; Yahyapour, R.  
(Eds.)

2011, XXVI, 358 p., Hardcover

ISBN: 978-1-4614-1613-5