

A Reference Architecture for Multi-Level SLA Management

W. Theilmann, J. Happe, C. Kotsokalis, A. Edmonds, K. Kearney, J. Lambea

Abstract—There is a global trend towards service-orientation, both for organizing business interactions but also in 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 realize service-orientation for both software and infrastructure services. Again, flexible composition across different layers is a major advantage of this paradigm. Service Level Agreements (SLA) are a common approach for 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 just specified at a single layer and do not allow service providers to manage their service stack accordingly. They have no insight on how SLAs at one layer translate to metrics or parameters at the various lower layers of the service stack. In this paper, we present a reference architecture for a multi-level SLA management framework. We discuss the fundamental concepts and detail the main architectural components and interfaces. Furthermore, we show how the framework can be flexibly used for different industrial scenarios.

Index Terms—Service Level Agreement (SLA), Service-Oriented Infrastructure (SOI), e-Contracting, Adaptive Infrastructures, Manageability, Non-Functional Properties

I. INTRODUCTION

THE paradigm of *Service-Oriented* has gained enormous momentum over the past few years. Business interactions are all the more based around services. Service industries are becoming the dominating sector and solutions are flexibly composed out of networked services [1], [2]. The paradigm of *Service Oriented Architectures* (SOA) has changed the way of building software systems [3]. Initially SOA was mainly applied to restructure the IT stack within an organisation. More recently it also evolves to a common paradigm for integration of cross-organisational service landscapes. Similarly, *Cloud Computing* has changed the way of providing complete IT systems (infrastructure, platforms or complete software solutions) to a service-oriented fashion (decoupled ownership, on demand provisioning, pay as you go) [4]. Overall, both business and IT services operate under a strong business context where customers can expect services

to be provided under well-defined and dependable conditions and with clearly associated costs.

Service Level Agreements (SLAs) are a common way to formally specify the exact conditions (both functional and non-functional) under which services are or are to be delivered. However, SLAs in practice are specified at the top-level interface between a service provider and a service customer only. Customers and providers can use top-level SLAs to monitor whether the actual service delivery complies with the agreed SLA terms. In case of SLA violations, penalties or compensations can 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 hosted by external providers. Such services include business, software, and infrastructure services. The quality of the service offered heavily depends on the quality of the services it uses. Furthermore, it also depends on the used elements and structure of the underlying IT system that realizes the respective service. Currently, service providers cannot plan their service landscapes according to SLAs of dependent services. They have no means to understand why a certain SLA violation might have occurred and how to express a penalization associated if the guarantee is breached. SLA guarantee terms are neither 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 to ensure top-level SLAs. Overall, the missing relation between top-level SLAs and (lower-level) metrics is a major hurdle for managing service stacks in terms of service planning, prediction or adjustment processes.

As part of the European Research project SLA@SOI [5], we developed the vision to use SLAs for managing a complete service stack (business and IT) in correlation with top-level business SLAs. 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 organisational trend in IT companies to organise 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/virtual infrastructure, middleware, application and process-level. Such internal SLAs describe the contract between the lower-level entities and higher-level entities consuming the lower ones. More precisely, the SLAs specify the required or agreed quality metrics but also the

W. Theilmann and J. Happe are with SAP Research, Karlsruhe, Germany, mail: wolfgang.theilmann|jens.happe@sap.com

C. Kotsokalis is with Dortmund University of Technology, Germany, mail: constantinos.kotsokalis@udo.edu

A. Edmonds is with Intel, Leixlip, Ireland, mail: andrewx.edmonds@intel.com

K. Kearney is with Engineering, Rome, Italy, mail: Keven.Kearney@eng.it
J. Lambea is with Telefonica Investigacion y Desarrollo, Madrid, Spain, mail: juanlr@tid.es

related configuration parameters.

In this paper, we present the detailed conception of a reference architecture for a multi-level SLA management framework. It is built on a previous discussion of a purely conceptual architecture [6] and experimental analysis of a specialized showcase [7]. We present the underlying concepts of the architecture and its main building blocks (components and interactions). Last, we show how the architecture can be flexibly used and adopted in a variety of different settings, including domains from Enterprise Resource Planning, Enterprise IT management, eGovernment and Telco Service Aggregation.

The remainder of this paper is organised as follows. Section II discusses the state of the art. Section III describes foundational concepts. Section IV introduces the developed reference architecture and Section V shows different adoption patterns of four exemplary use cases. Section VI concludes with a brief summary and outlook.

II. RELATED WORK

THE ambition to create a multi-level SLA management framework requires the integration of concepts from a large variety of disciplines and areas. We summarise the most important related work along the areas of modelling, negotiation, planning, provisioning, monitoring, and adjustment of SLAs. Other related aspects such as eContracting, service composition or autonomic management are omitted due to space restrictions.

SLA modelling is about the formal description of SLAs and other related artefacts (e.g. software and infrastructure) that matter within the overall SLA management process. WS-Agreement [8] is the prevalent standard for expressing SLAs. It defines a representation of SLA templates with terms free to modify, SLA offers based on these templates, and agreements themselves. A large body of work exists in the areas of software modelling (e.g. Unified Modeling Language (UML) [9] or Palladio Component Model (PCM) [10]) and infrastructure modelling (e.g. ITIL's CMDB [11]). Our framework goes beyond these approaches as it realises an SLA foundation model with much higher expressiveness which is at the same time also integrated with other modelling artefacts from software or system level.

SLA negotiation is about the actual interaction protocol used between service customers and providers. WS-Agreement provides only a "single-shot" protocol, which unfortunately does not cover requirements for multi-round negotiation with counteroffers and re-negotiation, although relevant extensions can be found in literature (e.g. [12]).

SLA planning is about the reasoning to determine actual SLAs a provider can agree to. It strongly relates to (performance) prediction and SLA translation techniques. Prominent approaches in this area rely on (layered) queueing networks (LQNs) [13], [14] or stochastic Petri nets (SPN) [15]. Chen et al. [13] uses LQNs for translating service level objectives (SLOs) into low-level system thresholds. Menasce et al. [14] use queueing networks to allow service brokers to plan the capacities of services traded by them. They ensure that SLAs of all customers are adhered to. Wombacher et al. [15]

use SPNs for analysing cost and resource usage of service compositions. A detailed analysis of this area can be found in [16]. Distinguishing aspects of our framework are the flexible support of different scenarios (service composition styles) and the clear separation of concerns between service and SLA planning but also between involved parties (preventing from universal knowledge assembled in one place).

SLA provisioning is about the orchestration of provisioning resources (human, software, infrastructure) according to agreed SLAs. This is essentially a scheduling problem, where service start times must be properly synchronised for dependent services and their SLAs. Scheduling dependent tasks has been extensively researched in the past (e.g. [17]). Most important aspect of our approach is the systematic integration into the overall SLA and service lifecycle.

SLA monitoring and adjustment is about the monitoring of an SLA-governed system and the interpretation of monitoring events against agreed SLAs. Research on service-based systems monitoring has so far focused only on mechanisms for monitoring properties expressed in first-order temporal logic languages, e.g. Event Calculus, either event-based [18], [19] or based on the instrumentation of the service execution environment [20], e.g. the BPEL Process [21]. Distinguishing aspects of our framework are the automated derivation of monitoring rules from SLAs and the integration between monitoring and related (autonomic) control actions in reaction to detected violations.

Though a large body of research exists on different isolated aspects of SLA management, there is no comprehensive approach yet available, that covers the complete SLA lifecycle, supports multi-level SLA management, supports different service types (business and IT), and can be flexibly applied and adapted to arbitrary domains. At least, a conceptual architecture with implementation of toolbox components for Grid services has been proposed in [26].

III. FOUNDATION CONCEPTS

BEFORE diving into the actual reference architecture, we detail some of its fundamental concepts around the notion of service level agreements and the relationships between them. This includes the definition of SLA management (including service and SLA lifecycles), the SLA (Template) model, and the service construction model.

A. SLA Management

The overall goal of the reference architecture is multi-level SLA management. We understand *SLA Management* as the management of service delivery systems in order to meet the QoS objectives (goals) specified in SLAs. As SLAs are the core artefact for describing offered, requested or agreed service characteristics, the SLA lifecycle focuses on the steps of interaction between a service provider and a service customer. It includes the following stages:

- *SLA Template design* ensures that offered QoS guarantees are realistic;
- *SLA negotiation* ensures that agreed QoS guarantees are realizable;

- *SLA runtime* ensures that QoS guarantees are satisfied; and
- *SLA(Template) archiving* ensures that previous experience is available to future cycles.

The management of SLAs happens in the context of the overall service lifecycle, which consists of the following stages:

- *Design and Development* of artefacts needed for service implementation;
- *Service Offering* prepares service artefacts for their instantiation and offering a service to customers;
- *Service Negotiation* between customer and provider ideally results in an agreed SLA;
- *Service Provisioning* creates an actual service instance which may include booking, deployment, and configuration activities;
- *Service Operations* addresses an actual service instance that is up and running. It might be adjusted in order to enforce an SLA; and
- *Service Decommissioning* subsumes activities to stop a service instance so that it cannot be accessed by the service customer anymore.

B. Data Models

In order 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* describes SLAs for the communication within and among SLA Managers, as well with external providers. The *Service Construction Model* provides and collects information necessary to create a new instance of a service (for a particular SLA). In the following, we provide an overview of both models.

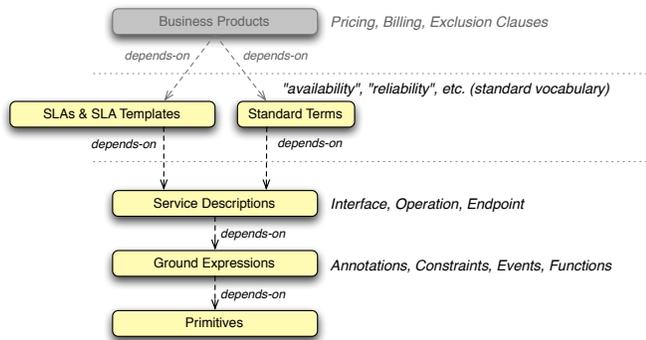


Fig. 1. Overview of the model hierarchy including the SLA(T) Model.

a) 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 quality of service (QoS) guarantees. The SLA(T) Model is part of a larger model hierarchy, depicted in Fig. 1. At the lowest level of the hierarchy are the Primitive (Ground) Terms on which everything else is built. Building on the Primitives are a handful of Ground Expressions, supporting annotations, and the generic expression of constraints, events, and functions. These expressions support extension/customisation through

standard vocabularies, in which specific terms, and semantics can be defined. Above this is a generic Service Description model, providing a means to describe the functional properties (interfaces, operations, and endpoints) of services. The main body of the present specification defines the SLA(T) Model, which builds on the Service Description level to allow for the expression of non-functional service properties and quality of service (QoS) guarantees. The SLA(T) Model provides the basic structure of an SLA(T), but 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 of Common Metrics. At the highest level is a Business SLA(T) Model, which builds on the SLA(T) Model and Common Metrics to model business-specific information, such as business terms (i.e. support), offer, constraints, pricing, penalties, billing details, termination and exclusion clauses.

Basically, an SLA is a set of agreements between two (or more) parties. These agreements are expressed by terms each of which denotes guarantees made by, or obligations on, the various parties. Each agreement term comprises an optional constraint expression specifying the conditions under which the agreement term holds (i.e. a precondition on the term). If no preconditions are specified then it is assumed the term holds for the entire effective duration of the SLA. Guarantees defined in the agreement are either guaranteed states or guaranteed actions.

A *guaranteed state* is a guarantee made by one of the parties to the agreement, that a certain state of affairs will hold, e.g. service level objectives (SLOs) or targets for key-performance indicators (KPIs). The state of affairs is defined by a constraint expression. A *guaranteed action* is an action that one of the parties to the SLA is obligated to perform (or may perform, or is forbidden from performing) under certain, specified circumstances.

For example, the expression $\text{mean}(\text{completion_time}(S)) < 500.0 \text{ ms}$ states that for operation S of the target service the customer requires a mean completion time below 500 ms. A precondition like $\text{mean}(\text{arrival_rate}(S)) < 200.0 \text{ rpm}$ ensures an upper bound for its usage by giving a mean arrival rate of 200 requests per minute.

b) Service Construction Model (SCM): The SCM is inspired by the management of services (cf. Section IV-B) and SLA concepts sketched above. We introduced the SCM to ease the communication between different components, responsible for the core SLA management, the management of services, and the quality evaluation of possible service offerings. Service Managers can use the SCM to manage multiple implementations of the same service. Furthermore, the SCM allows different components to access and add information about a potential service instance.

Figure 2 illustrates the basic concepts of the SCM. The core classes of the SCM (Service Type, Service Implementation, Service Builder, and Service Instance) represent the different stages in a service's life cycle.

A *Service Type* specifies the functional interfaces a service provides. Service Types can be realized by multiple *Service*

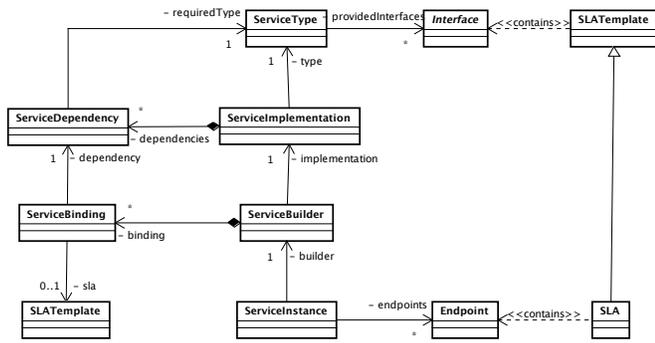


Fig. 2. Overview of the Service Construction Model.

Implementations that specify i) the artifacts of the implementation (such as software components, or executables) and ii) the dependencies of the artifacts on other services. For example, a software service “InventoryService” may depend on infrastructure services that host its virtual machines. Such dependencies are expressed in terms of *Service Dependencies*.

Service Builders hold information about resolved dependencies of a Service Implementation. For this purpose, a *Service Binding* relates a Service Dependency to an SLA Template or one of its specializations (SLA and Business Product). The SLA Template contains all information necessary to evaluate and access a service outside the domain of an SLA Manager. The SLA Template includes quality constraints and, after the SLA has been agreed, endpoints of the service.

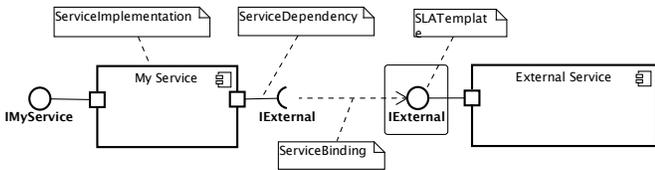


Fig. 3. Binding a Service Dependency to an SLA Template.

Figure 3 illustrates the concepts of ServiceBuilders and ServiceImplementations. On the lefthand side, a Service Implementation called “My Service” is depicted. It is represented by a simple component providing a single interface “IMyService”. Furthermore, the implementation depends on external functionality captured by interface “IExternal”. The link between implementation and interface reflects the Service Dependency (also known as (required) role in software component models [10]). The dependency is resolved by a Service Binding that links the Service Dependency to an SLA Template of an external provider. In Figure 3, the SLATemplate also contains the specification of interface “IExternal”.

Finally, a *Service Instance* models the actual runtime entity representing a service. The Endpoints of a Service Instance either refer to a running and accessible service or point to the location where the service will be available according to the time constraints defined in its SLA.

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

IV. ARCHITECTURE

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 (de-)composition 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.

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 meta models 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 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.

A. Overview

Figure 4 illustrates the main components of the SLA@SOI framework and their relationships. The framework communicates to external parties, namely customers who (want to) consume services and 3rd party providers which the actual service provider might rely upon. Relationships are defined by stereotyped dependencies that translate to specific sets of provided and required interfaces. Figure 4 shows a *possible* setup of the framework. Other configurations are given in Section V.

In order to achieve a good generalization of the framework architecture, the SLA Manager and the Service Manager are defined as general components that can be specialised for different domains (cf. Section IV-C). In Figure 4, we chose an example where SLA Managers are realized for business, software and infrastructure level and Service Managers are realized for software and infrastructure level.

On the highest level, we distinguish the Framework Core, Service Managers (infrastructure and software), deployed Service Instances with their Manageability Agents and MonitoringEventChannels. The Framework Core 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 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 status.

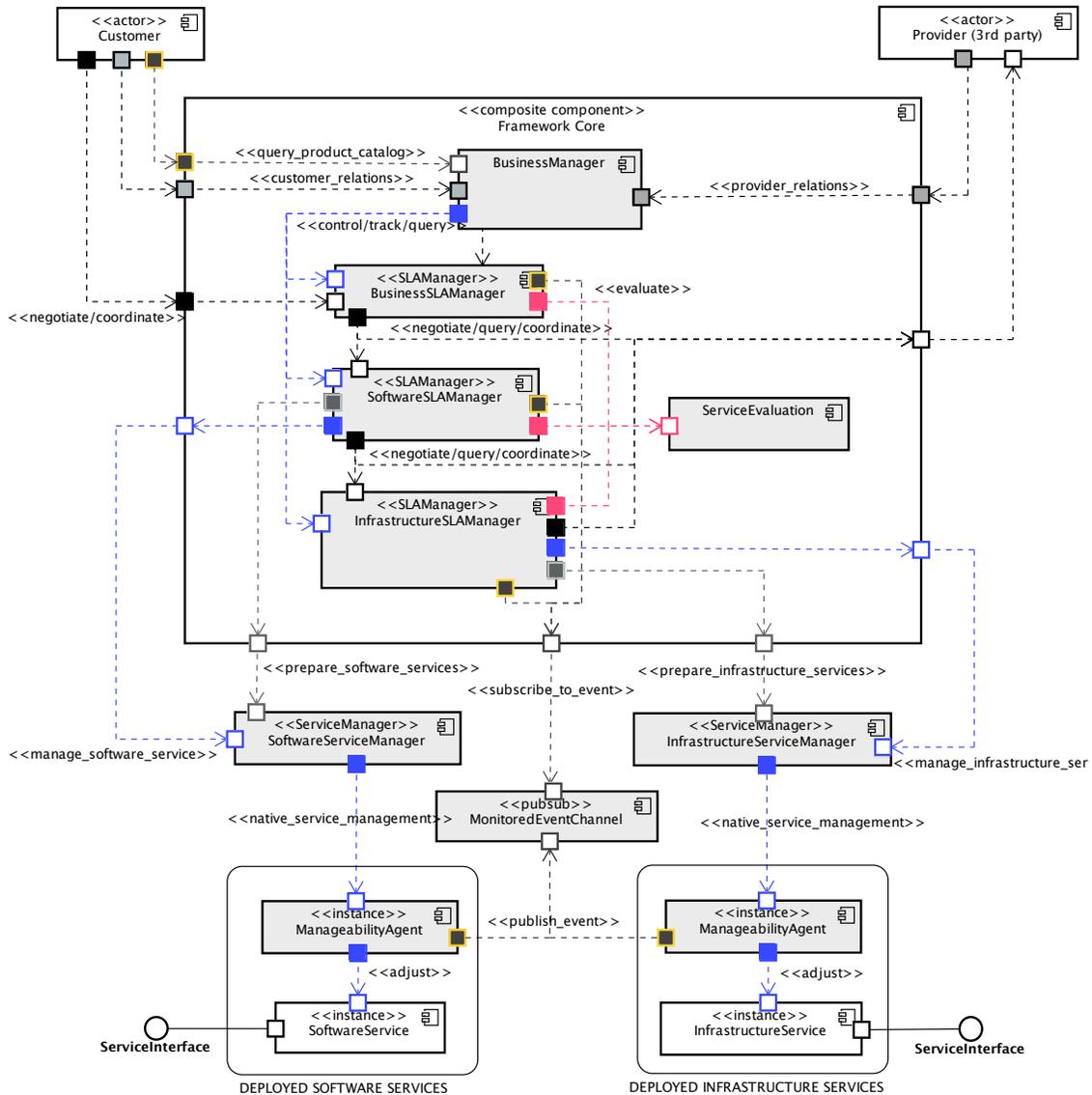


Fig. 4. Architecture Overview

In the following, we briefly describe the main components of our framework and their interactions. The *Business-Manager* controls 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 services. Furthermore, the BusinessManager governs the (Business-, Software-, and Infrastructure-) SLAManagers (<<control/track/query>>). For this purpose, an SLAManager has to adhere to business rules defined by the BusinessManager ('control') and have to inform the BusinessManager about their current status and activities ('track').

The (*Business-, Software- & Infrastructure-*) *SLAManagers* are responsible for the management of all SLA-related concerns. They are specialisations of a generic SLA Manager (cf. Figure 5). 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. However, only the BusinessSLAManager 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 catalog (accessible via <<query_product_catalog>>) maintained by the BusinessManager. 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 and to provide a unique access point to query historical consolidated information about his products consumption linked with the specific SLAs. All negotiation and reporting interactions are captured by the <<negotiate/query/coordinate>> relation.

These interactions are equally used at business-level for the customer interaction (<<negotiate/coordinate>>) where the BusinessSLAManager adds business-level considerations (e.g. billing) to the negotiation protocol. Finally, all SLA Managers can consult ServiceEvaluation to evaluate *a-priori* 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 of appropriate quality.

Infrastructure- and SoftwareServiceManager encapsulate all service-specific details. Both are specialisations of the abstract ServiceManager concept. Service Managers provide information about the service implementations supported, such as the service realisation 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 Managers can manipulate service instances via generic management functions provided by the ServiceManager.

The *MonitoringAdjustment Management System* provides the underlying fabric across different layers (i.e. across software and infrastructure layer) supporting the monitoring and management of service instances. The MonitoringEventChannel 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. ManageabilityAgents 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 service-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.

The *negotiation/planning sequence* starts with a customer querying for available products and SLA Templates (<<query_product_catalog>>) and eventually registering himself with the provider (<<customer_relations>>). The actual negotiation is then started by the customer via <<negotiate/coordinate>>. The framework manages the negotiation request in a hierarchical manner across different SLA Managers, again using the same <<negotiate/coordinate>> interaction. Each SLA Manager checks with its Service Managers for possible service implementations (<<prepare_[T]_services>>). For each possible implementation it resolves required dependencies to lower-level services with the next level SLA Manager (<<negotiate>>), evaluates possible combinations of service implementations and service bindings via the ServiceEvaluation (<<evaluate>>) and, finally, returns a set of feasible SLA offers to the customer (or the higher-level SLA Manager). Once an agreement has been established (again via <<negotiate>>) relevant resources are booked on all layers (<<prepare_[T]_services>>).

In general, the *provisioning sequence* is automatically triggered by the respective SLA Managers according to the service start times specified in the agreed SLAs. It is executed via the respective Service Managers (<<manage_[T]_service>>) who take care of the creation of service instances (if necessary), their configuration as well as the set up of a responsible ManageabilityAgent. Alternatively, provisioning might be also explicitly triggered by a customer (<<coordinate>>). Successful provisioning is eventually reported back to customers via the reporting mechanisms of <<customer_relations>>.

The *monitoring/adjustment sequence* typically starts with the detection of an SLA violation by the monitoring system. SLA Managers evaluate these violations and initiate adjustment actions via the Service Managers (<<manage_[T]_service>>) who use the respective ManageabilityAgent to execute the corresponding adjustment actions. Furthermore, SLA Managers report violations to the BusinessManager (<<control/track/query>>) who decides whether the customer needs to be informed (<<customer_relations>>) and/or a renegotiation may be initiated (<<negotiate>>).

The examples above illustrate the interplay of the framework components. However, the flexibility of the framework allows many different variants, such as provider initiated negotiation processes or different order of planning steps.

B. Generic elements

In order to apply the architectures to multiple service domains, the SLA management framework includes a set of generic components that can be specialised for the domains of interest. In the following, we explain the structure of the generic (i.e., domain-independent) framework components, discuss their functionality, and describe the hooks to plug in domain-specific extensions.

1) *Generic SLA Manager*: As discussed earlier in Section IV, the framework includes a generic SLA management component that is expected to be customized depending on the domain and use case at hand. The *Generic SLA Manager* (GSLAM) builds around domain-independent concepts for SLA negotiation and service monitoring, and enhances them with placeholders for domain-specific models and algorithms. Figure 5 illustrates the internal architecture of the GSLAM.

Its most important generic components are as follows:

a) *Protocol Engine*: The *Protocol Engine* (PE) implements different negotiation protocols, by means of a configurable state machine. We have made a design choice to separate between the protocol (i.e. message exchange sequencing and control), and the negotiation strategies which concern decision-making as regards what constitutes a “good” SLA. The PE controls the interactions between different SLAM implementations, and ensures sanity and security. For example, an incoming message contains the information about an agreement to a previous offer, but the referenced offer is unknown to the specific SLAM. Such a message should be rejected without further processing.

b) *Template Registry*: SLA Templates are used to start negotiations. In real-world situations, people know by experience what kind of quality levels they can expect from a service.

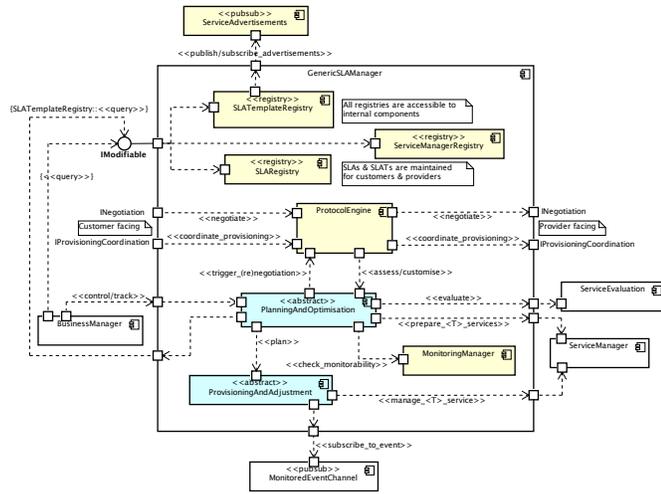


Fig. 5. Generic SLA Manager

However, computing systems do not carry this knowledge; instead, we implement templates as a way to instruct them with regards to the negotiable variables in relation to a service. For instance, a template would include information to indicate that, it is possible to negotiate property *CompletionTime* for a particular operation of a service. The domain-specific decision-making algorithms know by design how to use this property, as long as the service provider offers it for negotiation. The template registry is a query-able data store, where we implement complex reasoning for matching queries (also implemented as SLA templates) to stored information. Templates are normative and described in full by the SLA(T) model (cf. Section III-B).

c) *SLA Registry*: SLAs are stored in this registry after they are established. Queries return not only SLA information, but also the dependencies of one SLA onto others. This way, it is possible to find which SLAs would be affected if one fails.

d) *Service Manager Registry*: This data store is used by SLA Managers to manage the known Service Managers. It contains only static information about Service Managers; dynamic information (e.g. resource availability) is retrieved directly from the Service Manager using respective query methods.

e) *Monitoring Manager*: The *Monitoring Manager* (MM) determines the monitoring infrastructure that must be deployed in order to monitor a service, for which we establish one or more SLAs. In addition, the MM can perform *monitorability checks*, based on which it is possible to know at negotiation time, whether a SLA *can* be monitored at all. The rational is that a SLAM should never establish an agreement that it cannot monitor.

f) *Service Advertisements*: One of the design goals was to achieve service discovery and SLA template discovery concurrently. Additionally, we need to achieve a high degree of decentralization, and SLAM autonomy. For this purpose, additionally to query capabilities of the SLA template registry, we have implemented a Publish/Subscribe system which is used to advertise new templates as they become available. Prospective customers can receive the advertisements, filter

them, and cache them as initial indications of candidate providers for the services of interest. Follow-up queries to the respective providers' registries retrieve additional information about particular offers. We have not tied the system by design to a specific broadcasting technology; this is left open to the implementation.

In addition to these generic components, the GSLAM also includes placeholders for domain- or use-case-specific components, in the form of the *Planning and Optimization Component* (POC) and the *Provisioning and Adjustment Component* (PAC). The former implements algorithms to decide, during a negotiation, if an incoming SLA offer is acceptable and how to construct a subsequent outgoing SLA offer. Additionally, the POC carries the knowledge to do resource planning, to negotiate subsequent SLAs when required, and to decide whether a monitoring infrastructure is reasonably economic to deploy for monitoring a service and the respective SLAs. In general, the POC concerns with *utility* concepts.

Finally, the PAC has the responsibility to provision the services for which a SLA was negotiated, and to safeguard the SLAs in collaboration with monitoring infrastructure. It relies on plans that the POC feeds to it, but it also carries knowledge of tasks to execute when SLAs are violated (or, when its forecasting models indicate that a SLA will soon be violated unless appropriate actions are taken). The PAC will behave differently in different circumstances: It may take an action on its own given sufficient certainty about the problem and its solution, or it may request the POC to create a new plan, or even to re-negotiate an existing SLA.

2) *Service Manager*: Service Managers encapsulate all management activities specific to services, including service lifecycle management, booking of resources required, and maintenance of service specific information (service landscape). All this is part of the top-level interactions `<<prepare_[T]_services>>` and `<<manage_[T]_services>>`.

Management functionality for services throughout their lifecycle is primarily facilitated by the `<<manage_[T]_services>>` interaction. It includes the deployment, configuration, and startup and shutdown related functionality of services. The configuration is predominantly domain and service specific and, therefore, customized for each service.

Furthermore, the Service Manager manages and dispatches communication with domain-specific Manageability Agents. The Service Manager keeps track of the available Manageability Agents and their status.

In addition, the Service Manager is responsible for maintaining and managing the software resource booking and reservation related information. For example, it can maintain information about software licenses required for software service provisioning.

Finally, the Service Manager keeps track of available service types, service implementations, and of provisioned service instances. SLA Managers can query available service types as well as service implementations for a particular service type and provide metadata about all service implementations. The metadata includes (but is not limited to): dependencies

of the implementation to other services, a description of the service realisation, information related to provisioning, and available monitoring / manageability configuration options. However, SLA Managers can only read information about the available service implementations. The implementations and their metadata are maintained by the Service Provider. In addition to information about service implementations, the service landscape administrates metadata about service instances of the implementations under its control. The metadata of these instances may contain the service endpoint, a reference to the corresponding Manageability Agent, and external services used.

3) *Service Evaluation*: SLA Managers at each level (business, software, infrastructure) use Service Evaluation to determine a-priori evaluation of service quality parameters. The results of evaluation are used during negotiation to agree upon feasible terms and conditions regarding service quality. However, the concrete realization of Service Evaluation is domain-specific and varies in scope and solution strategy. Each type of service (business, software, infrastructure) comes with its own specific characteristics and evaluation goals. Regarding solution strategy, very different approaches may be possible, such as (i) interpretation and aggregation of historical data of service quality, (ii) application of rules and constraints to calculate expected quality parameters, and (iii) analysis and / or simulation of architectural models created during system design.

4) *Business Management*: Also business-related management aspects are largely realized in a generic manner. First, the generic business product model is build upon the SLA(T) model (see Section III-B) and supports business-specific terms such as pricing, warranties etc. and it has been derived from the SID standard [22]. Based on these modelling standards framework users can configure their product offerings, and the complete functionality for their automated management is then realized by the framework in a generic manner. Second, a rule-based approach (based on Drools [23]) allows to formulate business intelligence rules that are to be applied during the overall product/service lifecycle, e.g. for negotiation, penalty evaluation and priority analysis. Based on concrete rules specified by framework users, the framework will automatically implement them in the overall management lifecycle without the need for any further domain-specific adjustments. All this business-related functionality is realized within a generic BusinessManager implementation.

C. Domain-specific extensions

Within the framework there are two specialisations of the GenericSLAManager. The first is the SoftwareSLAManager used to manage software services (e.g. SaaS) under their agreed software SLAs. The second is the InfrastructureSLAManager that is used to manage infrastructure services (e.g. IaaS) also under their agreed infrastructure SLAs.

a) *Software*: In case of software services, the SoftwareSLAManager is responsible for the negotiation, the configuration of SLA-related monitoring, the coordination of provisioning actions, and the adjustment of running Service

Instances. For negotiation (done by the software-specific PlanningAndOptimisation), the SoftwareSLAManager makes use of the predictive SoftwareServiceEvaluation that is based on a model-driven performance and reliability prediction methodologies [10]. The predictions allow the SoftwareSLAManager to identify a set of feasible offers for its customers. The SoftwareSLAManager stepwise constructs a service using the Service Builder. Moreover, the Builder provides all information needed by the predictive SoftwareServiceEvaluation to assess the potential quality of a service instance. Finally, the SoftwareServiceManager provides the functionality to provision the required service instances and acts as a gateway to the service instance's Manageability Agent (used by the software-specific ProvisioningAndAdjustment Component).

b) *Infrastructure*: For infrastructure services, the InfrastructureSLAManager manages all aspects of SLAs concerning IT infrastructural resources. It is a specialisation of the GenericSLAManager and as a result supplies infrastructure-specific implementations of the POC and PAC.

The InfrastructurePOC is responsible for the planning and optimisation of infrastructure SLAs. It receives requests for infrastructure, queries the InfrastructureServiceManager for potential provisioning solutions, selects and reserves the optimal one and requests the InfrastructurePAC to provision the selected plan as appropriate. If local resources cannot satisfy the request (e.g. due to lack of availability or specification discrepancies), the InfrastructurePOC can attempt to outsource to third party providers to satisfy the request.

The InfrastructurePAC is responsible for the provisioning and adjustment of Infrastructure SLAs. It directs the InfrastructureServiceManager to provision as per the plan supplied by the POC. It also decides on any adjustments required, e.g. to avoid potential SLA violations.

Of note is the InfrastructurePAC's ability to communicate with standard-compliant infrastructure services. It accomplishes this by implementing the Open Cloud Computing Interface (OCCI) [24], an OGFbib:ogf standard partly driven by SLA@SOI. The work within the InfrastructureSLAManager presents a SLA Manager that is applicable to all infrastructure providers that will offer this standardised interface.

V. ADOPTION EXAMPLES

BROAD adoption of our SLA management framework in different domains and settings is a key design goal. This section will sketch a few use case examples that demonstrate the broad applicability of the framework architecture but will also show the differences of the resulting framework instantiations. The examples span the areas of Enterprise IT, ERP Hosting, Service Aggregators and eGovernment and are visualized in figure 6.

The *Enterprise IT use case* is about the application of the framework to three core areas of Enterprise IT, namely, IT-enabling the enterprise, IT efficiency and IT investment and technology adoption. At the core of the use case is an SLA-enabled Platform-as-a-Service (PaaS) Service Manager and its complementing SLA Manager. The use case utilizes the generic framework by reusing the Business Manager

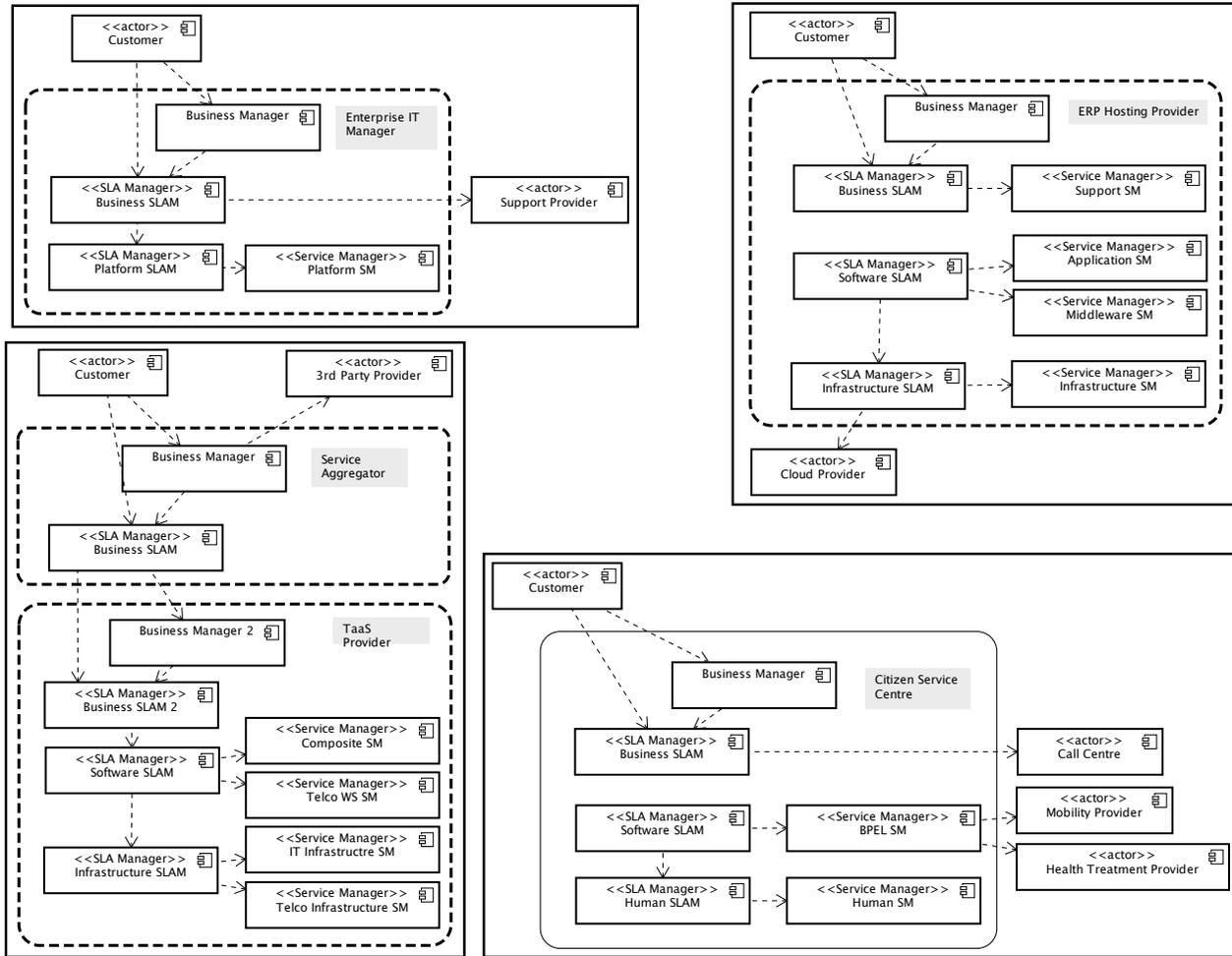


Fig. 6. Framework adoption by four different use cases: (1) Enterprise IT, (2) ERP Hosting, (3) Service Aggregator, (4) eGovernment

component and providing specialisations of the Generic SLA Manager, namely the Business SLA Manager and the Platform SLA Manager. In this use case there is only one Service Manager, the Platform Service Manager. Supporting the Business SLA Manager are the Enterprise's support provider. With the combination of these cooperating managers, four themes are being investigated. These themes are 1) Business Alignment, 2) Server Efficiency, 3) Sustainability and 4) Data Security. All these themes and the resulting policies and SLAs are governed by the Enterprise Capability Framework.

The *ERP Hosting use case* is about the dynamic provisioning of hosted Enterprise Resource Planning solutions. It applies the generic framework architecture in a straightforward manner by realizing 3 distinct SLA Managers and 4 distinct Service Managers. A Business SLA Manager is responsible for the overall offering but also for the planning of human support services, a Software SLA Manager is responsible for the SLAs of the complete software stack and an Infrastructure SLA Manager is responsible for the in-house IT resources but also able to contact external cloud providers in order to acquire resources for burst scenarios. Furthermore, a Support Service Manager is responsible for planning/managing the human support services. Two separate Software Service Managers

are introduced, one for managing application logic, one for managing the underlying middleware. Last, an Infrastructure Service Manager deals with all the internal IT resources.

The *Service Aggregator use case* is about the Web-based offering of traditional telco services (e.g. voice, SMS) and for the flexible composition and aggregation of these into higher-level added value services, including multiple offers from external service providers (such as Internet players). The use case is realized by instantiating two different variants of the SLA framework. The first one supports the foundation of telecommunication as a service (TaaS) and comes with a layered SLA/Service Manager architecture for Web-Service wrapped telco infrastructure services, software compositions on top of these and a business layer for managing the sales of the eventual services/products. The second one addresses the role of a pure service aggregator who does not own any resources but simply aggregates external services. This one adopts a framework with only a business SLA Manager and a business manager.

The *eGovernment use case* is about supporting a citizen service center providing combined health treatment and mobility services (elderly citizens that need a medical treatment can request a mobility service for transportation to the place

of treatment). The use case is realized with 3 distinct SLA Managers and 2 Service Managers. A Business SLA Manager has the overall responsibility of the offering and automatically negotiates with 3rd party call centers, when the expected demand exceeds the capacity of an in-house booking service. A Software SLA Manager (together with a Software Service Manager) is realized which is specialized on BPEL processes (in the essence coordinate the combined service offering, including the dynamic selection and binding of mobility providers based on already negotiated SLAs). Last a pair of SLA and Service Manager is introduced for managing the human based in-house service.

VI. CONCLUSION

WITH this paper we presented a reference architecture for multi-level SLA management, that supports the comprehensive management of possibly complex service stacks. Service Level Agreements are used for managing the non-functional aspects of the complete service lifecycle. Furthermore, SLA translations across different layers allow for consistent interlinking of complete service networks and hierarchies. The presented architecture is based on the experiences gained from an SLA framework built around a specific reference application. The main achievement against that work is the generalization of the concepts so that the architecture can serve a large variety of domains. Four example use cases demonstrate the applicability and flexibility of our architecture.

Future work is planned on assessing the business benefit of multi-level SLA management in the presented use cases. Technically, the SLA management framework will be extended in various dimensions, such as support for managing specific non-functional properties (e.g. reliability), a library of SLA planning algorithms and finally advanced interfaces for harmonized cloud infrastructures.

ACKNOWLEDGEMENTS

The research leading to these results is partially supported by the European Community's Seventh Framework Programme (FP7/2001-2013) under grant agreement no.216556.

REFERENCES

- [1] IfM and IBM, *Succeeding through service innovation: A service perspective for education, research, business and government* White Paper, Univ. of Cambridge, 2008, URL: http://www.ifm.eng.cam.ac.uk/ssme/documents/080428cambridge_ssme_whitepaper.pdf
- [2] SAP Research, *Service Delivery Framework: Supporting Service Delivery for On-Demand, Business Network, Cloud Environments on an Internet-scale*. White Paper, SAP Research, November 2009, URL: http://www.internet-of-services.com/uploads/media/SDF-Value-Proposition_01.pdf
- [3] M. Papazoglou and W.J. van den Heuvel, *Service oriented architectures: approaches, technologies and research issues*. The VLDB Journal **16**(3) (07 2007) 389–415
- [4] M. Armbrust, et al., *Above the Clouds: A Berkeley View of Cloud Computing*. Report, UC Berkeley Reliable Adaptive Distributed Systems Laboratory, February 10, 2009, URL: <http://radlab.cs.berkeley.edu/>
- [5] SLA@SOI project: IST- 216556; Empowering the Service Economy with SLA-aware Infrastructures, <http://www.sla-at-soi.eu/>
- [6] W. Theilmann, R. Yahyapour, J. Butler, *Multi-level sla management for service-oriented infrastructures*. Towards a Service-Based Internet (2008) 324–335

- [7] 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
- [8] Andrieux, A., Czajkowski, K., IBM, A., Keahey, K., IBM, H., NEC, T., HP, J., IBM, J., Tuecke, S., Xu, M., et al., *Web Services Agreement Specification (WS-Agreement)*. Open Grid Forum (2007)
- [9] Object Management Group (OMG), *Unified Modeling Language: Superstructure Specification: Version 2.1.2, Revised Final Adopted Specification (formal/2007-11-02)* (2007)
- [10] Becker, S., Koziolok, H., Reussner, R., *The Palladio component model for model-driven performance prediction*. JSS **82** (2009) 3–22
- [11] Office of Government Commerce, *The official introduction to the ITIL service lifecycle*. Stationery Office Books (TSO) (2007)
- [12] Frankova, G., Malfatti, D., Aiello, M., *Semantics and Extensions of WS-Agreement*. Journal of Software **1**(1) (2006)
- [13] Chen, Y., Iyer, S., Liu, X., Milojevic, D., A., S., *Translating service level objectives to lower level policies for multi-tier services*. Cluster Computing **11** (2008) 299–311
- [14] Menascé, D.A. and Ruan, H. and Gomaa, H., *QoS management in service-oriented architectures* Performance Evaluation **64** (2007) 646–663
- [15] Bodenstaff, L., Wombacher, A., Reichert, M., Jaeger, M.C., *Monitoring dependencies for slas: The mode4sla approach*. Proceedings of IEEE International Conference on Services Computing, 2008. SCC '08., 8-11 July 2008, Honolulu, USA. (2008) 21–29
- [16] Li, H., Theilmann, W., Happe, J., *Sla translation in multi-layered service oriented architectures: Status and challenges*. Technical Report 2009-8, Universität Karlsruhe (TH) (April 2009)
- [17] Chetto, H., Silly, M., Bouchentouf, T., *Dynamic scheduling of real-time tasks under precedence constraints*. Real-Time Systems **2**(3) (09 1990) 181–194
- [18] Spanoudakis, G., Mahbub, K., *Non intrusive monitoring of service based systems*. Int. Journal of Cooperative Information Systems **15**(6) (2006) 325–358
- [19] Barbon, F., Traverso, P., Pistore, M., Trainotti, M., *Run-time monitoring of instances and classes of Web service compositions*. In: Proc. IEEE ICWS 2006. (2006)
- [20] Lazovik, A., Aiello, M., Papazoglou, M., *Planning and monitoring the execution of Web service requests*. Int. Journal of Digital Libraries (2006)
- [21] Baresi, L., Guinea, S., *Towards dynamic monitoring of WS-BPEL processes*. In: Proc. ICSOC 2005. (2005)
- [22] TM Forum, *Information Framework (SID)*, URL: <http://www.tmforum.org/InformationFramework/1684/home.html>
- [23] JBoss, *Drools 5 - The Business Logic integration Platform*, URL: <http://labs.jboss.com/drools>
- [24] OCCI, *Open Cloud Compute Interface*, URL: <http://www.occi-wg.org>
- [25] OGF, *Open Grid Forum*, URL: <http://www.ogf.org>
- [26] Brein project, *Final Brein Architecture*. Deliverable D4.1.3 v2, 13 July 2009, URL: http://http://www.eu-brein.com/index.php?option=com_docman&task=doc_download&gid=63&Itemid=31