

SLA@SOI Adoption Guide

**Whitepaper and Appendix D of
Deliverable D.A1a Framework architecture**

Version 1.0; July 20, 2011

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

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](https://creativecommons.org/licenses/by/3.0/).

Document History			
Version	Date	Author	Changes
0.1	18 January 2010	SLA@SOI consortium	Initial structure draft.
0.2	4 February	SLA@SOI consortium	Input on SCM & SW SM
0.3	9 February 2011	Franz Brosch	Update of FZI-related contents
0.3	21 February 2011	Juan Lambea	Sections 4.2, 5.3, 5.4 updated
0.4	23 February 2011	Miguel Rojas	Section 4.1 SLA Manager
0.5	28 February	Peter A. Chronz	Section on SLA templates.
0.6	01 March	Kuan Lu	Section on specific SLAM
0.7	10 May 2011	Wolfgang Theilmann	Revision on models and specialized service managers;
0.8	18 May 2011	Miha Stopar	Tashi related sections, Infrastructure Monitoring sections
0.8	14 June 2011	SLA@SOI consortium	Consolidation of feedback
0.8	15 June 2011	Howard Foster	Monitoring Manager section
0.9	15 June 2011	Miguel Rojas	Edition of SLA Manager
0.9.1	15 June 2011	Khaled Mahbub	Monitoring Rule Section
0.9.2	15 June 2011	Davide Lorenzoli	Split Monitoring Approach to SLA Monitoring and Infrastructure Monitoring
0.9.3	16 June 2011	Juan Lambea	Review of TID contribution
0.9.4	24 June 2011	Peter A. Chronz	Updated negotiation protocols and SLAMs
1.0	20 July 2011	SLA@SOI consortium	Final cross-check

Executive Summary

The mission of the SLA@SOI project [1] is to deliver and showcase an innovative open SLA Management Framework that provides holistic support for service level agreements (SLAs) - 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.

The project created a reference framework architecture [2] and an open source implementation [3] to support service providers in managing SLAs for their service offerings in a systematic way.

This adoption guide addresses decision makers and chief architects of service providers and shall help them to understand on how to best apply the overall framework and how to embed/enhance it into their target environment. The guide attempts to clarify step by step the questions/issues use case adopters have to resolve when realizing an SLA management based on the SLA@SOI framework in their domain.

The guide is complemented by a tutorial [4] that exemplifies the adoption for one particular use case, the so-called open reference case.

Table of Contents

1	Introduction	7
2	Service analysis	8
2.1	Role clarification	8
2.2	Service clarification	8
2.3	Negotiation protocols.....	8
3	System analysis.....	10
3.1	Service implementations.....	10
3.2	Quality characteristics	10
3.3	Predictability	10
3.4	Monitorability	11
3.5	Adjustment	11
3.6	Template construction	12
4	Architecture specification	13
4.1	SLA Managers	13
4.2	Business Manager	13
4.3	Service Managers.....	14
4.3.1	Software Service Manager.....	15
4.3.2	Infrastructure Service Manager.....	15
4.4	Service Evaluation	16
4.5	Manageability Approach	17
4.6	SLA Monitoring	17
4.6.1	Monitoring Manager.....	17
4.6.2	Sensors and Reasoners.....	18
4.7	Infrastructure Monitoring	19
5	Data Model Preparation.....	20
5.1	Service Types.....	21
5.2	SLA Templates	21
5.3	Service offers	22
5.4	Business Rules.....	23
5.5	Service Construction Models.....	24
5.6	Evaluation models.....	24
5.7	Manageability models	25
5.8	Monitoring rules.....	26
5.8.1	Infrastructure Monitoring	27
6	Custom Development	28
6.1	Specific SLA Managers.....	28
6.2	Specific Service Managers	28
6.3	Manageability System.....	29
6.3.1	Manageability Agents	29
6.3.2	Reasoner	30
7	Framework configuration & setup	31
7.1	Requirements.....	31
7.2	Download.....	31
7.3	Configuration	31
7.4	Setup	34
8	References	36
	Appendix A: Glossary	37
	Appendix B: Abbreviations	40

1 Introduction

The mission of the SLA@SOI project [1] is to deliver and showcase an innovative open SLA Management Framework that provides holistic support for service level agreements (SLAs) - 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.

The project created a reference framework architecture [2] and an open source implementation [3] to support service providers in managing SLAs for their service offerings in a systematic way.

This adoption guide addresses decision makers and chief architects of service providers and shall help them to understand on how to best apply the overall framework and how to embed/enhance it into their target environment. The guide attempts to clarify step by step the questions/issues use case adopters have to resolve when realizing an SLA management based on the SLA@SOI framework in their domain.



The guide is complemented by a tutorial [4] that exemplifies the adoption for one particular use case, the so-called open reference case.

The adoption model covers the following phases

1. *Service Analysis*, where the general service offering is to be clarified.
2. *System Analysis*, where the capabilities and constraints of the underlying system are to be clarified.
3. *Architecture Specification*, where the general SLA management architecture is to be defined.
4. *Data Model Preparation*, where the required meta models and actual model data is to be specified.
5. *Custom Development*, where the development of custom components is to be analysed.
6. *Framework Configuration & Setup*, where the overall configuration and setup of the framework shall be defined.

These phases are further detailed in the following, each phase via a dedicated section which details the individual steps.

For each phase, we describe

- the required activities, annotated with the symbol 
- the expected results, annotated with the symbol 

2 *Service analysis*

The goal of this phase is to understand the top-level context and requirements for the envisaged service offering.

2.1 *Role clarification*

The first step is to analyse the basic players relevant for your service offering scenario.

» **Activity**

Analyse your scenario, who is assuming the role of a primary service provider, who are his customers and which 3rd party service providers exist on which the primary service provider possibly relies upon.

- A default assumption could be that the SLA management framework is basically operating to support the primary service provider.

■ **Result**

- A list of the relevant roles in your scenario.

2.2 *Service clarification*

The next step is to understand the scope and characteristics of the envisaged service offerings.

» **Activity**

Specify the (top-level) services that can be offered by the primary service provider.

- Create a list of relevant services, their functional scope and possible variability aspects which could be subject to customer negotiation.

Clarify which service level objectives are relevant for your top-level services.

- Create a list of relevant service level objectives per offered service.

■ **Result**

- A list of relevant service offerings, incl. their scope, variability, and SLOs.

2.3 *Negotiation protocols*

Finally the usage of negotiation protocols for establishing SLAs needs to be clarified.

» **Activity**

Based on the prepared taxonomy of services and their hierarchy, devise the protocols for negotiating SLAs on the basis of SLA templates.

Guiding questions here are:

- Who will consume the service offered by the SLA-template?

- How many sub-SLAs will be required?
- How many complementary services are required?

Based on these considerations, appropriate negotiation protocols need to be devised for each service.



Result

- A list of negotiation protocols per service, including specific parameters such as the number of involved negotiation parties, negotiation time-outs and negotiation customizations, which depend on specific protocols.

3 *System analysis*

The goal of this phase is to understand the capabilities and constraints of the underlying system that shall eventually deliver the services defined before.

3.1 *Service implementations*

The first step is to understand, how a service can be implemented (aka realized) at all.

» **Activity**

Analyse how the identified top-level services can be implemented, via internal means/resources and/or external sub-services.

- Determine internal means that can implement (parts of) a service including their configuration options and variants.
- Determine dependencies of those service implementations on other subsequent services.
- Include possible sub-services into the list of services to be further analyzed, regarding their respective SLOs, SLA templates, implementations etc.

■ **Result**

- A list of possible service implementations, their respective internal means and their dependencies on sub-services.

3.2 *Quality characteristics*

The next step is to understand the quality characteristics of the possible service implementations.

» **Activity**

Analyse possible quality (aka KPI) ranges that your service implementations could achieve.

- This analysis should be aligned with the top-level service level objectives listed in the previous phase.
- You may apply some design time prediction techniques for this analysis.
- You also have to analyse KPI dependencies of possible service implementations on the KPIs / SLAs provided by lower level services.

■ **Result**

- For each service implementation list the possible KPI ranges (depending on assumed sub-service quality KPIs).

3.3 *Predictability*

The next step is to understand if and how quality characteristics can be dynamically predicted, based on customer requirements, internal service configuration parameters, and lower-level service quality characteristics.

» Activity

Analyse how you can predict the actual quality KPIs of a service implementation.

- You might have dedicated prediction models, rely on historical data or just have some educated guesses.
- Take into account that service quality depends on many parameters such as service usage, service configuration, and quality of sub-services.
- For each possible prediction approach, determine its constraints in terms of predictive power (which influencing parameters can be considered) and prediction effort (how fast and with how many resources can a prediction run be executed).
- Compare the power of the respective approaches with the needs of the required service level objectives and in particular the needs of having dynamic negotiation on certain parameters.

■ Result

- A ranked list of possible prediction approaches including a description of their power.

3.4 *Monitorability*

The next step is to understand if and how service level objectives can be monitored.

» Activity

Analyse how the identified KPIs / service level objectives can be monitored.

- Monitoring might happen either directly or indirectly by some related indicators.
- Analyse whether additional instrumentation of your system is required.

■ Result

- A list how each KPI can be monitored.

3.5 *Adjustment*

The next step is to understand if and how services can be adjusted during runtime (for safeguarding of SLAs or also for re-negotiation purposes).

» Activity

Analyse the means you have for adjusting a running service instance.

- Analyse the means for adjusting internal resources while the service is up and running, e.g. by resizing or reconfiguring resources. Consider the cases where re-negotiation will be required for external services, and how your planning would change in such a case (in relation to negotiation-time planning). Normally this would imply some additional constraints, or extra knowledge acquired which did not exist at the time of negotiation.
- Analyse the impact of these adjustment activities on your quality KPIs.

■ Result

- A list of possible/available adjustment operations and their respective impact and constraints.

3.6 *Template construction*

The final step of this phase is to construct the actual SLA templates which seem feasible from a provider perspective. This is based on the provider capabilities regarding quality, prediction, monitorability, and adjustment which must be well understood at this point.

» Activity

Construct possible SLA templates for all layers of services analyzed so far.

- Take into account the lessons learned from the top-level requirements, the provider-internal service (implementation) hierarchy, and the dependencies on 3rd party services.

■ Result

- SLA Templates (in an arbitrary notation) for top-level and lower-level services which are under control of the service provider.

4 *Architecture specification*

This phase shall specify the top-level architecture of your specific framework instance.

4.1 *SLA Managers*

The first step is to decide how many SLA Managers are needed and what their respective responsibilities are about.

» **Activity**

General considerations

- Familiarize yourself with the SLA model and the SLA Manager architecture/interactions
- Prepare your SLA Templates
- Decide whether you will implement an SLA manager for each service instance, for each service, for groups of services, or for your whole domain. The grouping of services is a question of IT-architecture as well as one of business-issues. As such SLA managers should correspond to your technical as well as to your institutional domains.
- If you want to have a SLAM per service instance, that means that after each successful negotiation and if there is a new service instance required, you need to create on-the-fly a new SLAM configuration file and spawn the new SLAM process.
- If you want to have a SLAM per service, service group or domain, it is easier. You only need to make sure the respective service managers are registered onto the SLAM.
- The implementation of new (domain specific) SLAMs is supported by the Skeleton-SLAM plugin. Familiarize yourself using this maven-based plugin for creating the basic structure of the domain-specific components. Modify configuration files and related java classes
- You will have to implement the use-case specific POC, and also the use-case-specific parts of the PAC. The business logic of the two components will have to take into account both the knowledge of what is a "good" SLA for your provider/customer, but also what can be done if a SLA is violated or about to be violated.

■ **Result**

- This activity results in an architectural hierarchy of SLA managers that correspond to your various technical and business domains.
- Each of your SLA managers contains customized implementations for the domain-specific parts of the POCs and the PACs. As such, your SLAMs and the hierarchy of your SLA-managers is tailored to your custom use-case.

4.2 *Business Manager*

The next step is to plan for the possible usage of a Business Manager component.

» Activity

General considerations

- Business Manager is needed whenever some internal business controls are needed for products and service offerings

Analyse

- Market and business restrictions needed for the product offering to customers.
 - ⇒ Geographical areas
 - ⇒ Currencies
 - ⇒ Languages
 - ⇒ Countries
- Business models supported for products and services to be sold.
 - ⇒ Price types
 - ⇒ Price variations to be used to increase or decrease standard prices
 - ⇒ Billing frequency
- Service specification characteristics in terms of categories that can be linked with the products and services. Later providers can publish products and services linked to these categories and customers can find them.
- The kind and nature of promotions that can be applied to products and services to offer discounts and charges in order to be able to generate rules.

■ Result

- Description of the business framework and specification of the environment in which products and services can be sold to customers.

4.3 Service Managers

The next step is to decide how many Service Managers are needed and what their respective responsibilities are about.

» Activity

General considerations

- Service managers are needed whenever some internal resources/components/artefacts are needed for implementing a service.
- In general, the relationship of service manager to SLA manager is n:1, while the most typical setup is probably 1:1.
- Separate service managers are useful if different kinds of internal resources/components/artefacts can/shall/must be managed separately, either for having a clear separation of concerns or for legacy reasons of required management software.

Analyse

- What kind of artefacts is needed for implementing your respective services?

- How are those artefacts currently being managed?
- To what extent would it be beneficial to separate the management of different artefact types.

Result

- Description of the planned service manager instances, their respective responsibility on artefacts, and their relationship to existing management functionality.

4.3.1 Software Service Manager

Activity

Analyse

- Which software services are to be managed by the Software Service Manager?
- What kind of software artefacts are necessary to instantiate the service?
- What are important configuration options of the software service?
- How can the service provisioning achieved best?

Steps

- Create a description of all service types and implementations that are to be supported by the Software Service Manager (SSM). The SSM uses this information for provisioning and management.
- Define a concept for the provisioning of services. The concept must be implementable as an extension of the SSM (IProvisioning).
- Analyse the management capabilities needed for the services and provide a concept for a corresponding manageability agent that can be used by the SSM. The access to the manageability agent is to be encapsulated by the interface IManageabilityAgentFacade.

Result

- Description of the planned software service manager instances, including its responsibility on artefacts and relationship to existing management functionality.

4.3.2 Infrastructure Service Manager

Activity

Analyse

- What type(s) of infrastructure needs to be provisioned?
- Which provisioning system(s) will be used to manage the infrastructure?
- Are third party infrastructure providers required?
- How are images prepared and managed for the provisioning system?
- What resource attributes can be manipulated by the provisioning system?
- Does Ganglia support all terms that need to be monitored?

- Does the provisioning system have an OCCI interface?
- What infrastructure policies need to be comprehended when planning and optimising deployments?

Steps

- Decide on provisioning system and monitoring system to deploy, taking infrastructure requirements of the use case and available implementations of Infrastructure SLA Managers and Infrastructure Service Managers into account.
- Comprehensively describe all functional and non-functional properties that can be manipulated and offered as part of the SLA.
- Define all infrastructure policies that should be comprehended when deciding on where to provision, or re-provision, infrastructure.
- Implement or adapt or configure an Infrastructure SLA Manager by implementing / adapting / configuring an appropriate Infrastructure POC and Infrastructure PAC.
- Implement or adapt or configure an Infrastructure Service Manager on top of the selected Provisioning System.
- Implement any infrastructure monitors not already available.
- Configure the Provisioning System, e.g. with appropriate images.



Result

- An SLA-enabled Infrastructure Layer that can provision, monitor, re-provision and optimise infrastructure automatically based on infrastructure policies and SLA commitments.

4.4 Service Evaluation

The next step is to decide whether to use a Service Evaluation component.



Activity

General considerations

- The Service Evaluation component offers functionality to predict the expected quality (e.g., performance, reliability) of services before they are actually used.
- Typically, Service Evaluation is invoked during automated SLA negotiation by the Planning And Optimization sub component of SLA Managers, i.e. it supports the planning and optimization of SLAs.
- The typical relationship of SLA Manager to Service Evaluation is 1 : 1. However, this is not the only possibility. Principally, one Service Evaluation instance might serve multiple SLA Managers. On the other hand, one SLA Manager might use multiple Service Evaluation instances for different kinds of quality evaluations.
- For performance and reliability prediction of software services, a prediction model of the service-based system under study has to be created first (see Section 5.6).

Analyse

- Which quality properties are critical for the software services that you offer, and should be negotiated in an SLA?

- Which external software and infrastructure services are required for providing the target software services? Which offers with which quality guarantees exist for those external services?
- How are the target services structured, and how do they make use of external services and infrastructure resources?

■ Result

- Prediction of the quality (e.g. completion time, throughput, probability of failure) that the target services will exhibit, once they are deployed and operating. Can be used to determine feasible quality parameters for SLA negotiation.

4.5 *Manageability Approach*

The next step is to clarify the architecture of the manageability system, in particular the usage of manageability agents to provide management access to service instances. The cardinality of the manageability agent with respect to its managed service instances is 1 to n , meaning that a single manageability agent can take care of many specific service instances at a time. Moreover, the service instances can be implemented using different domain-specific technologies. This is achieved through the notion of domain-specific manageability agent facades. Each specific service instance has a domain-specific façade through which the platform can get information about the service, manage sensor configurations, and execute specific corrective actions on the service instance.

» Activity

General considerations

- Analyse the services that are used throughout the system and determine which specific implementation technologies they use.
- Implement a specific Manageability Agent Façade for each of the service technologies being used. The project already provides façades for BPEL processes, and for Axis 1.4.2 and Axis 2 atomic services.
- Extend the source code of the manageability-agent subproject to include the newly implemented manageability agent facades.
- Notice that in order to use domain-specific effectors the framework will need a domain-specific planning and adjustment component (PAC).

■ Result

- The platform is now capable of configuring sensors, and of triggering effectors on domain-specific service instances.

4.6 *SLA Monitoring*

4.6.1 *Monitoring Manager*

The Monitoring Manager (MM) coordinates the generation of a monitoring configuration of the system. It decides, for an SLA specification instance it receives, which is the most appropriate monitoring configuration according to configurable selection criteria. A monitoring configuration describes which components to configure and how their configurations can be used to obtain results of monitoring Guaranteed States and Agreement Terms. There are three

types of Monitoring Features in the monitoring system. First, Sensors collect information from a service instance. A sensor can be injected into the service instance (e.g., service instrumentation), or it can be outside the service instance intercepting service operation invocations. Second, Effectors are components for configuring service instance behaviour. An effector can also be injected into a service instance or can interface with a service configuration. The third type of monitoring feature is a Reasoner (or also known as a Reasoning Engine) which performs a computation based upon a series of inputs provided by events or messages sent from sensors or effectors.

» Activity

General considerations

- Monitoring configurations may represent partial terms of an SLA, determined by assessment of each Agreement Term in an SLA.
- Selection of monitoring components is currently made on a “first match” basis. There is an opportunity to replace this selection with optimisation and preference selection routines.
- If features are not located for part of a given term, then the complete Agreement Term under consideration is deemed not monitorable.

Analyse

- The features required for all terms specified in an SLA, including reasoners for Guaranteed State expressions and Agreement Term expressions.

■ Result

- A complete monitoring system configuration for a given SLA. The configuration is then ready to be used by the Low Level Monitoring System to instantiate the monitoring components required and configure the events required or published for service monitoring.

4.6.2 *Sensors and Reasoners*

Sensors and Reasoners are monitoring system essential components. The former gather data from service providers, which is afterwards made available to reasoners. Sensors collect software information such as interactions that take place between service providers and consumers. This information is made available in form of events delivered, via a communication infrastructure, to reasoners. Reasoners compute received information to detect SLA violations.

» Activity

General considerations

- To provide generic means to gather and make available monitoring data, and to process for monitoring purpose gathered data
- The abstraction provided by sensors allows domain-specific implementations, e.g., sensors injected in BPEL processes or sensors gathering data by analysing traffic between services and customers.
- The abstraction provided by reasoners allows heterogeneous reasoning components, i.e., monitoring engines to detect SLA violations, to be deployed with the SLA@SOI framework.

■ Result

- A highly decoupled architecture for collecting and consuming information. Sensors gather data and send it, via appropriate communication channels. Reasoners receive sent data and perform activities such as SLA violation detection.

4.7 *Infrastructure Monitoring*

Low Level Monitoring System (LLMS) is a component that is responsible for the alerting the higher-level components (Infrastructure PAC as a part of Infrastructure SLAM) about the Infrastructure SLA violations.

» **Activity**

General considerations:

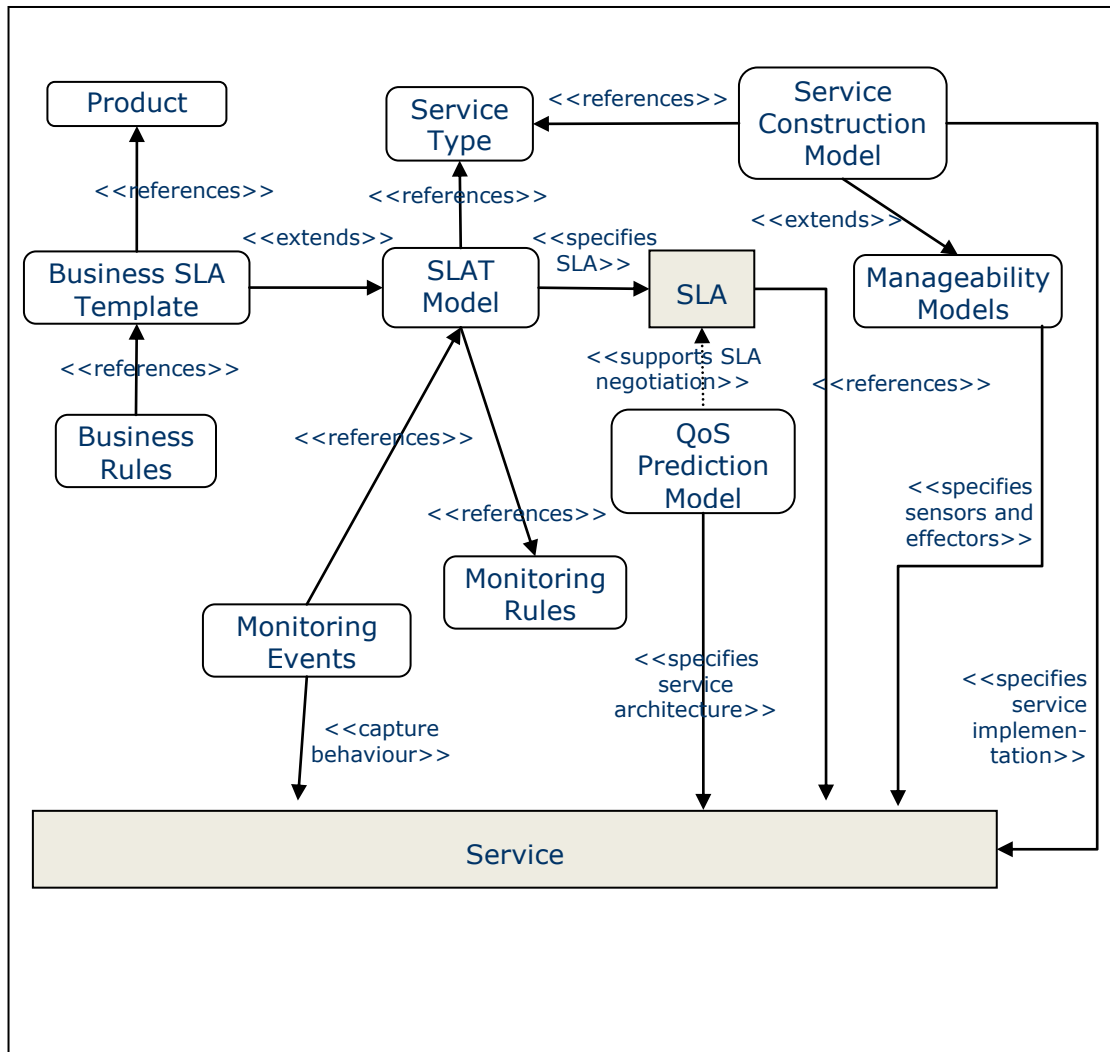
- There are no special interfaces to be implemented in order to adapt the LLMS component for a specific use case. The installation guide and interaction instructions can be found on the [LLMS Source Forge page](#).
- The only input that is required for an LLMS is a monitoring request. LLMS is listening on the XMPP channel in order to get it. In the SLA@SOI framework monitoring request is provided by a Monitoring Manager, but LLMS can be also used as a standalone application, the user just needs to take care for a monitoring request, which contains information about which virtual machines need to be monitored and what are the SLA values – means what are the SLA violation thresholds (memory amount for a virtual machine, number of CPU cores...). The sample monitoring request can be found on the [Source Forge](#).
- When SLA warning or violation is detected, the LLMS sends a message alert on the XMPP channel. See [Source Forge](#) for a sample of this message and information how to connect to the XMPP channel and how to parse the messages.

■ **Result**

- System for a virtual machines monitoring, which can be used as a standalone application and can be easily integrated with any component. The system configuration is dynamic and is required at the start-up of the virtual machines. SLA warning messages can be configured as well. The system can be easily extended with new metrics.

5 Data Model Preparation

This phase is basically about creating all the required data models that are later used at framework runtime. The following diagram provides an overview on the relationships of the various models and also on the aspect the respective model describes for the eventual target service.



Explanation of inter-model stereotypes

- <<references>> denotes that a source model contains a reference/link to a target model.
- <<extends>> denotes that a source model extends the target model with additional details that represent a cross-cutting concern.

Explanation of modelled service aspects

- <<specifies service implementation>> denotes the core purpose of the service construction model which is the description of how a service can be implemented.
- <<specifies service architecture>> The QoS prediction model specifies service components, interfaces, composition, behaviour, usage and

allocation, together with quality annotations that allow for conducting quality predictions.

- <<specifies sensors and effectors> The Manageability Model specifies the sensors needed to capture and correlate runtime data regarding the service, and the effectors needed to adjust the service's behaviour.

Explanation of relations to SLA

- <<specifies SLA>> The SLA(T) model is the means to specify SLAs (as well as the more general SLATs).
- <<supports SLA negotiation>> Based on an architectural service specification, prediction methods can be applied to predict service quality, in order to determine feasible SLA parameters. SLA parameters for service quality are expressed through standard QoS terms (from the SLAT model).

5.1 Service Types

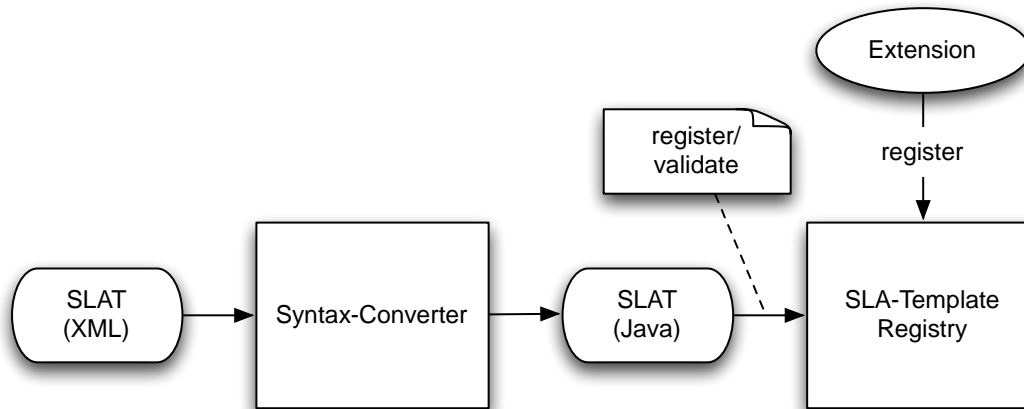
As general preparation step, specify the ServiceTypes for each service that is to be provided or required by the framework – this definition is done as a paper based definition. It will be later used by the SLA template model and the service construction model.

5.2 SLA Templates

SLA templates are a core requirement for adopting and using the SLA@SOI framework, since they provide the basis for negotiation, monitoring and provisioning of services. SLA templates are published via the publish-subscribe mechanism to potential customers as product-descriptions. An important aspect of SLA templates is that they contain variables fields for the provided non-functional properties. During negotiation concrete values of those fields are being chosen by all parties involved in the negotiation process. Upon agreement a service level agreement is created, which contains and preserves all agreed upon parameters of the SLA template. As such the values of an SLA template can be changed, which is contrasted by SLA, which are fixed.

SLA templates can in theory be created using various concrete syntaxes of the SLA model. For example the Java-based representation can be used to create an SLA template during run-time. A similar approach is chosen during negotiation to create an SLA from an SLA template. Usually however the most convenient way is to prepare an SLA template in form of an XML-document. This XML-based SLA template has to comply with the provided XML-schema.

The provided XML schema describes the syntax, which is expected from an SLA template, so that it can be parsed by the syntax-converter into the SLA model's Java representation. However it should be noted that even a SLA-template which is compliant with the XML-schema, might not pass validation through the template-registry, which is a obligatory step, when introducing a template to the framework. The reason for this is the use of external vocabulary to express semantics of non-functional properties and operators, which apply to those parameters. External vocabulary is denoted by the type STND, which are represented by URI-strings within the XML document. These URI-strings are checked for validity in the template-registry against the registered vocabulary.



Conclusively writing a SLA template, which relies on non-functional properties beyond the published core-terms, requires to also register new terms with the template registry. An extension needs to implement the interface **org.slasoi.slamodel.vocab.ext.Extensions** and provide information regarding the validation of this extension. This new term is then registered with the instance of the template registry as follows:

```

SLATemplateRegistry registry = context.getSLATemplateRegistry();
registry.addExtensions(new B4Terms());
  
```

Once an extension is registered with the template registry, the corresponding terms can be used in SLA templates by including their URIs in the appropriate STND-elements.

Decide on the eventual top-level services that shall be offered and their interfaces.

- You may add the service interface specification to a service registry (not part of the SLA@SOI framework).
- For each service (and its interface) you must specify a unique interface reference (UUID) which can be later used by SLATs and Service Managers to identify the service.

Specify the SLATs that shall be offered.

- Specify the eventually offered SLATs including their unique interface reference and their guarantee terms (states and actions).
- Make them available in the template registry.

Product models

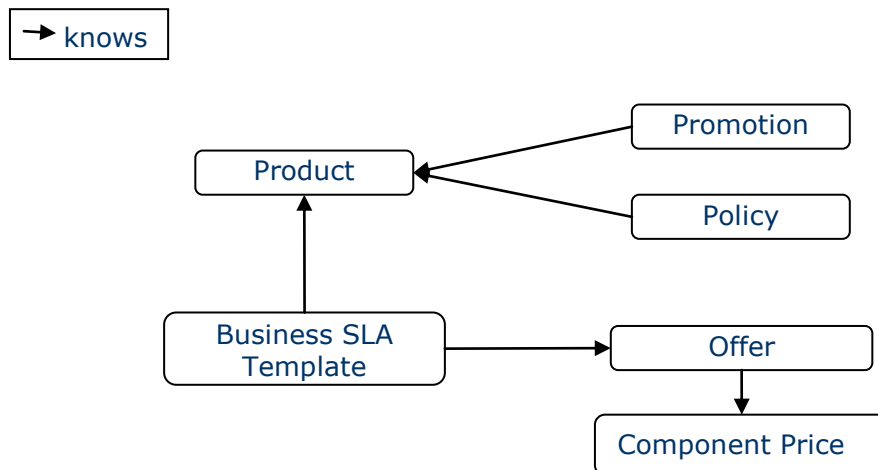
- Specify the product models on top of the specified SLATs.

5.3 Service offers

The product can have several offers, and each one has his different component prices that specify the way to be charged (all this information is stored in database). For instance, subscription and monthly fee are different component

prices. The promotion and/or policy are linked with the product and they are stored in the database.

A business SLA template has knowledge about the product and the concrete offer and his price definition. Business SLA templates and SLA templates are stored in SLAT registry.



The steps needed to create a product are:

- The first step is to create the service SLA template.
- Next one is to publish a service with the previously created SLA template.
- In the product creation, is necessary to select the services (SLA templates) to be included in the product. Second part of this step is offers creation. It means to define business model behind the product. It is to define the component prices specification (for instance subscription fee of 1 euro and monthly fee of 0,5 euros). And finally is necessary to create the different Business SLA templates with the business terms associated to each offer.
- Next one is to create policies associated to the product.
- Next one is to create promotions associated to the product.
- Finally, it is necessary to activate the product.

5.4 Business Rules

Business rules are used to manage some aspects of the framework from business point of view. We can distinguish two main groups of aspects that can be modelled:

- SLA Negotiation. In each negotiation of a product, there are several aspects that will be considered.
 - Product. There are some parameters associated to each product like maximum price discount, or penalties associated to it.
 - Policies are associated to guarantee terms and expresses how a guarantee term can be modified for a customer and this measure of the change can be translated to a price modification of the product.

There are different kinds of policies because they are based on business rule templates. Price details are stored in the BSLAT with product and offer information.

- Promotions. Business rule templates can be defined in a different way to increase or decrease price of the product using other kind of information. The parameters used with promotions are products features and/or customers profile information. Price details are stored in the BSLAT with product and offer information.
- Violations & Adjustment
 - Penalty policies. In each BSLA template can be defined how can be generated the penalties and under what kind of conditions are triggered. These information is linked with product because they are parameters of the product (like above).
 - Adjustment policies. Business rules are also applicable to the adjustment and they can drive the way of react when a penalty is produced. These kinds of rules are associated with automatic termination clauses and with renegotiation.

5.5 Service Construction Models

The specification of a Service Construction Model subsumes the following activities:

- Collect the ServiceType definitions that are relevant for your model.
- For each implementation possible, create a model instance that captures the artefacts required for the implementation, their dependencies, their configuration points, and their monitoring capabilities. Furthermore, the implementation model must specify the ServiceType that it realizes. Please note, that a ServiceType can be realized by multiple ServiceImplementations.
- For software services, this information is stored in a software landscape.

5.6 Evaluation models

Specify evaluation models.

- A specification of the prediction models and the description of development of the ORC models as one example can be found under <https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/doc/QoS-Model-Creation.doc>. When creating the prediction models, you need to specify model annotations that indicate quality-relevant properties of the services under study, such as
 - physical resource demands during service execution that influence the service completion times and the utilization of resources,
 - failure-on-demand probabilities of individual computation steps during service execution that influence the overall service success probabilities.
- The service evaluation (see Section 4.4) uses the prediction models at the service negotiation stage for software performance and reliability prediction. Through the <<evaluate>> interaction, a caller of service evaluation (typically a software SLA manager, see Section 4.1) can use the prediction results to assess the feasibility of standard QoS terms

(which are defined through the SLA model) as negotiated in SLA templates (see Section 5.2):

- `qos:completion_time`: The time between the arrival of a service invocation request at the service provider's system boundary and the completion of service execution. Prediction provides a full probability distribution over the completion time, rather than only a mean value.
- `qos:throughput`: The maximum arrival rate of service invocation requests that the system will serve without dropping requests. Prediction yields the throughput of the target service as a result.
- `qos:reliability`: The probability that upon a service invocation request, the service completes its execution without failures (where we assume that failures may happen due to software faults in service components, as well as unavailable hardware and network resources).
- Beyond the prediction models, service evaluation takes a set of service builder objects from the service construction model (see Section 5.5) as an input, which represent different possible service realizations. For a detailed specification of the <<evaluate>> interaction and the corresponding data models, see <http://sourceforge.net/apps/trac/sla-at-soi/wiki/SoftwareServiceEvaluation>.

5.7 Manageability models

The manageability models are for reasoning on the software system's architecture, and for designing it to be manageable at run-time. In order for a service to be manageable we require that it support the installation of sensors, for capturing run-time data that can be used for monitoring, and of effectors, for issuing run-time adjustments onto the service.

The model definitions can be found in the A6 deliverables for years two and three. The models define the data sources that should be added to the system, as well as where they should be attached. The models also allow for atomic data, collected through the data sources, to be correlated, aggregated, and in general manipulated to produce complex data such as key performance indicators. Example KPIs are the service's average response time, its reliability, and its request arrival rate. A model specifies that the deployed system will deploy sensors that allow the collection of such KPIs, meaning that those specific KPI values can be negotiated with the client and formalized in SLAs. Finally the models allow the definition of a number of adjustment capabilities (effectors) that need to be present in the system, from simple violation notification services, to more complex activities such as dynamic binding, and BPEL process restructuring.

Starting from these models it is possible to automatically synthesize sensor configurations for the sensor instrumentation approaches proposed within the project for BPEL processes and Axis 1.4.2 and Axis 2 atomic services. It is also possible to automatically configure the dynamic binding and process restructuring capabilities available within the Dynamic Orchestration Engine.

The modelling is to be achieved before we negotiate specific SLAs with customers. The resulting models should guide the negotiation. They determine what can be monitored at run time, since monitoring depends on the data that can be collected through the sensors, and what can be done to attempt to keep the system aligned with a negotiated SLA.

5.8 Monitoring rules

The monitoring engine that has been used in the SLA@SOI framework for monitoring SLAs at the software service layer is called EVEREST. EVEREST is a general-purpose engine for monitoring behavioural and quality properties of distributed systems based on events captured from them during the operation of these systems at runtime. The properties that can be monitored by EVEREST are expressed in an Event Calculus [5] based language called EC-Assertion.

Event Calculus (shortly referred to as "EC" in the following) is first order temporal logic language that expresses properties of dynamic systems (i.e., systems that can consume and generate events in ways that depend on and can alter their internal state) in terms of two basic modelling constructs, namely events and fluents. An event in EC is something that occurs at a specific instance of time, has instantaneous duration, and may cause some change in the state of the reality (system) that is being modelled. This state is represented in by fluents.

The occurrence of an event in EC is represented by the predicate *Happens*($e, t, \mathcal{R}(t_1, t_2)$). This predicate represents the occurrence of an event e at some time point t that is within the time range $\mathcal{R}(t_1, t_2)$ and is of instantaneous duration. The boundaries of $\mathcal{R}(t_1, t_2)$ can be specified by using either time constants or arithmetic expressions over time variables of other predicates in the EC formula that includes the *Happens* predicate. Events in EC can affect the overall state of a system into two possible ways: they can initiate or terminate a specific state within it. To represent these effects, EC uses two specific predicates, namely the predicate *Initiates*(e, f, t) and the predicate *Terminates*(e, f, t). The predicate *Initiates*(e, f, t) signifies that a fluent f starts to hold after the event e occurs at time t . The predicate *Terminates*(e, f, t) signifies that a fluent f ceases to hold after the event e occurs at time t . EC uses also two additional predicates, namely *Initially*(f) and *HoldsAt*(f, t). The first of these predicates signifies that a fluent f holds at the start of the operation of a system. The second predicate signifies that a fluent f holds at time t .

EC defines a set of axioms that can be used to determine when a fluent holds based on initiation and termination events that regard it.

Events in EC-Assertion can be invocations of system operations, responses from such operations, or exchanges of messages between different system components. Furthermore, fluents are defined as relations between objects and represented as terms of the form *rel*($O1, \dots, On$), where *rel* is the name of a relation which associates the objects $O1, \dots, On$.

The properties to be monitored at runtime are specified in EC-Assertion in terms of monitoring specifications that consist of *monitoring rules* and *assumptions*. Monitoring rules and assumptions are expressed in terms of the predicates listed above and have the general form *body* \Rightarrow *head*. The meaning of a *monitoring rule* is that if its body evaluates to *True*, its head must also evaluate to *True*, whilst the meaning of the *assumption* is that when its body evaluates to *True*, its head can be deduced from it. A detailed description of EC-Assertion can be found in [6].

For each QoS Term that EVEREST reasoning engine supports, there is a parametric monitoring template which includes a set of assumptions, used for the QoS Term computation and monitoring. The use of such templates is necessary since the SLA model does not provide formal definitions of standard QoS terms in a form that would enable their processing and generation of the corresponding EC-Assertion formulas from basic EC predicates and fluents. To automate the process of template selection and instantiation, the monitoring templates are described using the Formal Template Language (FTL) [7]. FTL is a generic formal language for expressing templates of any target language. FTL is generative, i.e., it

describes sentences of some target language (in this case EC-Assertion) and can generate sentences when provided with an instantiation. A brief introduction of FTL, as well as, a detailed description of the EVEREST monitoring specifications generation process using EC aware FTL templates is provided in [8].

The Reasoning Component Gateway (EVEREST RCG) for EVEREST is responsible to generate the operational monitoring specifications for EVEREST. EVEREST RCG generates the operational monitoring specifications in three steps. First it transforms SLA expressions to an intermediate notational form known as Abstract Syntax Trees (AST). Then it parses the ASTs and selects the appropriate FTL templates for monitoring the SLA guarantee terms. Finally, it generates the operational EVEREST monitoring specifications from the selected FTL templates by analysing the monitorable SLA expressions that are assigned to EVEREST. A detail description of parametric monitoring templates and the transformation of SLA guarantee terms into EC-Assertions can be found in [6].

5.8.1 Infrastructure Monitoring

There is set of SLA terms implemented in the LLMS module (e.g. auditability, data classification, hardware redundancy level, isolation, MTTF, MTTR, MTTV, reporting interval, SAS70 compliance, CCR compliance, CPU cores, CPU speed, data encryption, disk throughput, net throughput, location, memory size, persistence...).

To enable a new SLA term in the LLMS please check the documentation on [Source Forge](#).

6 Custom Development

This section describes all the needed custom (probably Java) development activities that might be needed for instantiating a complete SLA management framework.

6.1 Specific SLA Managers

The next step is to describe what is needed for realizing a complete SLA Manager

» Activity

General considerations

- Understanding about SLA model and SLA manager architecture.
- Domain specific SLA templates (Software / infrastructure) have to be developed here. Therefore, Java-based implementation will be instantiated here for parsing the SLA templates.
- Domain specific (Software / infrastructure) SLA managers are corresponding to domain specific (Software / infrastructure) service managers. Therefore, Java-based implementation will be instantiated here for the interaction with service manager.
- Based on the Skeleton-SLA manager plug-in in section 4.1, domain specific POC and PAC are implemented here. In POC, there are 4 interfaces that have to be implemented in Java: << IAssessmentAndCustomize >>, << INotification >>, << IPlanStatus >> and << IReplan >>.
- The Java implementation for interaction between domain specific SLA managers and monitoring components (monitoring manager and Low Level Monitoring System) should also be considered.

■ Result

- During the negotiation phase, the POC has to consider not only the customers' functional requirements but also their non-functional ones. Finally an optimal solution can be generated.
- During the provisioning phase, the PAC can handle the provisioning of corresponding types of services.
- In case of a violation, the PAC can adjust the provisioned service efficiently and avoid the penalty. The POC can also start re-negotiation with customer if it is necessary.

6.2 Specific Service Managers

Specific Service Managers can be easily derived from the concept of a Software Service manager.

» Activity

General considerations

- Implement the IProvisioning interface of the Software Service Manager for your service domain.

- Implement the `IManageabilityAgentFacade` interface of the Software Service Manager for your service domain.
- If needed the `BookingManager` can also be overwritten with domain specific functionality.
- Describe your service implementations (artifacts, dependencies, configuration parameters, monitoring features, ...) within the `SoftwareLandscape`.

Result

- A fully implemented service manager implementation for your specific domain which is automatically linked to the general SLA management process and the manageability/provisioning capabilities of your environment.

6.3 *Manageability System*

6.3.1 *Manageability Agents*

Manageability Agents are essential components used to bridge the SLA@SOI framework with the intricacies of domain-specific services. They allow the framework to manage services through appropriate sensors and effectors.

Activity

General considerations

- Analyze the domain-specific services that are part of your system. Understand the intricacies and differences of each. Each domain-specific service will provide different sensors and effectors, and will require a different domain-specific façade to be adopted within the manageability agent.
- If your system comprises BPEL processes running within the DOE (Dynamic Orchestration Engine) and/or atomic AXIS-based services you can adopt the corresponding already provided facades.
- If your system comprises other kinds of services, implemented using other kinds of technologies, you must to implement an `IManageabilityFacade` for each kind of service. In particular you will have to define how the sensor configuration methods and the effector invocation methods in the façade's interface map onto the specific sensors and effectors that exist for that specific service. Moreover, you will also have to extend the implementation of the `ManageabilityAgent` so that it can be aware of the new kind of façade and instantiate it when needed.
- You must also take into consideration that the cardinality relationship between Manageability Agents and services is not required to be one-to-one. A single Manageability Agent can "take care" of more than one service at a time through different facades. It is up to the designer to define the amount of manageability agents the need to be deployed in the system. This may depend on the sheer number of services that collaborate in the system, or on the amount of different kinds of service technologies being used, or on the different layers in the system (e.g., we can have one agent for software and one for infrastructure), or on a combination of all these reasonings.

Result

The definition of the required manageability agents allows the framework to effectively configure sensors on domain-specific services so that their behaviour can be monitored with respect to a negotiated SLA. It also allows the framework to issue adjustments onto the domain-specific services to keep their behaviour aligned with the negotiated SLA.

6.3.2 Reasoner

Reasoners are monitoring system essential components. Reasoners compute information provided by Sensors to detect SLA violations.

» Activity

General considerations

- The abstraction provided by reasoners allows heterogeneous reasoning components, i.e., monitoring engines to detect SLA violations, to be deployed with the SLA@SOI framework.

■ Result

A highly decoupled architecture for collecting and consuming information. Reasoners receive sent data and perform activities such as SLA violation detection. Sensors can be provided by any vendor and can be integrated into the SLA@SOI framework as long as they implement the ReasonerComponentGateway interface.

7 Framework configuration & setup

In this section we explain the major activities that are required for the technical configuration and setup a framework instance.

7.1 Requirements

The requirements for executing the following activities and thus setting up a framework instance are the following:

- JDK 1.6
- Maven 3.0.2
- Pax Runner 1.5.0
- MySQL Community Server 5.1.43
- Open Fire event bus
- SVN

The above mentioned tools should be installed as per the instructions provided on associated web pages.

7.2 Download

The source code of the SLA@SOI platform is available on the following SVN repository:

<https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk>

7.3 Configuration

In this section, we will describe the activities required to configure the framework and the environment in which it is to run.

» Environment variables configuration

The next activity is to configure the framework environment with required environment variables.

An environment variable named SLASOI_HOME must be created with the purpose of tracking all the configuration files required by different framework components in a common place. This allows for a complete portability of the platform code, in particular with respect to its integration within the SLA@SOI framework.

Create a new environment variable SLASOI_HOME and set its value to:

trunk/common/osgi-config

In order to avoid heap space issues during the framework build it is advisable to create the MAVEN_OPTS environment variable and set it to the value:

Xms512m -Xmx1024m -XX:MaxPermSize=512M



Result

The environment variables that are needed by the framework are defined.

» DB configuration

The next activity consist of setting up a database instance and populate it with all the required schemas needed by the framework. Persistency management inside the framework is handled using a common instance of a MySQL database that serves all the needs of the framework modules.

Once you have installed MySQL Community Server 5.1.43 instance, populate it with generic schemas that are required to run the framework. Following are the databases that must be populated on the MySQL db instance.

- SLA Registry
- SLA Template Registry
- Business SLA Model

The SLA Registry is a persistent store for SLAs, and historical SLA state information. It is essential to create this schema before framework launch. In order to create this schema, run the following script file.

<https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/generic-slamanager/sla-registry/db/model.sql>

In order to populate data in to the SLA Registry database, use the following script

<https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/generic-slamanager/sla-registry/db/testbed.sql>

In order to use the newly created SLA registry database you need to update the relevant properties inside the following file:

`$(SLASOI_HOME)/generic-slamanager/sla-registry/db/gslam-slaregistry.cfg.xml`

The SLA Template Registry is a persistent queriable store for SLA Templates. The schemas for the template registry of the SLA managers are created at runtime during the framework launch by each SLA manager (software, business, infrastructure).

For populating such registries with relevant SLA template, see the adoption guidelines of the corresponding SLA manager.

In order to use the SLA Template Registry database you also need to update the relevant properties inside the following file:

`$(SLASOI_HOME)/generic-slamanager/template-registry/db.properties`

The Business SLA Model is a persistent queriable store that allows the framework to define products and services, and other business oriented information that are commonly included in the SLA. The corresponding schema is named SLA@SOI and it is important to create it before the framework launch.

In order to create this schema, run the following script on the newly created database instance.

<https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/business-manager/DB/dbModel.sql>

In order to populate this schema, possibly modify and then run the following script.

<https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/business-manager/DB/dataModel.sql>

■ Result

Framework persistency is now configured.

» Event Bus configuration

Several framework components communicate with each other by placing messages on an event bus instance. It is therefore essential to install an instance of the Open Fire event bus, as specified in the requirement section, and configure the framework so that the bus is used by the different components of framework.

Let us take for instance the Provisioning and Adjustment Component (PAC) of one of the Software SLA Manager (SSM). The configuration file for defining the bus properties is located inside the configuration folder of PAC component inside `$SLASOI_HOME`. E.g., the `eventbus.properties` file of the SSM PAC can be found on following location:

```
$SLASOI_HOME/software-slamanager/provisioning-  
adjustment/eventbus.properties
```

Likewise the `eventbus.properties` file of the PAC for the Infrastructure SLA Manager can be found on following location:

```
$SLASOI_HOME//infrastructure-slamanager/provisioning-  
adjustment/eventbus.properties
```

Other than PAC components, several other components of framework, e.g. `FBKRcg`, `CityRcg`, rely on event bus communication. For more details about configuration of event bus channel for these components, it is recommended to see the corresponding adoption guidelines.

■ Result

The framework components are now enabled to communicate using the event bus.

» Provisioning layer configuration

Apache Tashi is used for the virtual machines management. It has been extended with some new SLA terms for the SLA@SOI needs. Isolation, location, CPU speed, auditability, data classification, net throughput, disk throughput have been added to the existing CPU cores, memory, number of virtual machines and other terms. In order to enable SLA management of these terms the following configuration of the Tashi is needed:

- Hosts have to be configured appropriately. For example it must be specified if the host has auditability enabled. This can be done via `tashi-client.py`, described on [Source Forge](#).
- XMPP communication has to be configured (all configuration resides in the Tashi configuration file). This is needed for a communication between OCCI server and Tashi, as well as for Tashi messages that are sent to the Infrastructure Monitoring module (LLMS) and which enables the detection of the Infrastructure SLA violations. For a configuration of the LLMS check the documentation on [Source Forge](#).

■ Result

Infrastructure Service Manager (OCCI server) is now enabled for starting and managing the virtual machines. Low Level Monitoring System is enabled for monitoring the Infrastructure SLAs and alerting Infrastructure PAC about SLA violations.

»» Log configuration

This activity is optional and serves for configuring the logging levels of different framework component as per the requirements of the user. If one has strong requirement for changing the default logging levels for framework component, then you can use this activity to define logging levels as per your need.

The framework uses a centralized logging system for controlling logging levels of different framework modules.

The logging properties for different framework modules can be configured using the `org.ops4j.pax.logging.properties` file located on following path.

```
$SLASOI_HOME/configuration/services
```

During the execution of the framework, the logging levels can be modified and applied dynamically by modifying the mentioned property file and by refreshing the logging bundle with following command:

```
> refresh [org.ops4j.pax.configmanager_0.2.2_bundle_id]
```

■ Result

The logging of the framework will appear on the console as per what has been defined in the configuration.

7.4 Setup

In this section we describe how to setup and launch a running instance of the framework.

»» Build

The first setup activity is the installation of the framework. After the checkout of the source code, the platform can be built and installed by running the following maven command from the trunk directory:

```
mvn clean install -fae -Dmaven.test.skip=true
```

■ Result

All the required artifacts are now installed within the local maven repository and the platform is ready to be executed.

»» Syntax Converter launch

The Syntax Converter is part of the generic SLA manager which provides an interface for negotiating SLAs to external components. It is developed as an external broker application and is therefore started prior to framework using a script. Change to directory:

```
/trunk/generic-slamanager/syntax-converter
```

and start the Syntax Converter broker application using either the Linux script (`SycBroker`) or the Windows one (`SycBroker.bat`).

By default, the syntax converter broker application starts on local host port 7077. If you have requirement to run this application on another location or port then modify the following property file and restart the application:

```
$SLASOI_HOME/syntax-converter/broker.properties.
```

■ Result

The Syntax Converter broker is up and running, ready to receive requests.

» Framework launch

At this point one can launch an instance of the core framework. Some more configuration of its main components can be achieved.

By default, the framework Web services runs on port 8080. If you have need to run the framework Web services on a different port, then modify the following option inside the main launcher file (trunk/pax-runner/runner.args):

```
org.osgi.service.http.port=8080
```

and restart the framework. Please note that changing the port of framework Web services has direct impact on the SLAM configuration. It is therefore necessary to make sure that the property *slam.epr.negotiation* is referencing to correct URLs for negotiation Web services inside the corresponding configuration files:

```
$SLASOI_HOME/slams/<$slam_id>.instance1.cfg
```

To start the framework, step into the pax-runner folder of the trunk and execute the pax-run tool:

```
pax-run.
```

The framework consist of more than one hundred and eighty bundles that will get loaded after the launch command. The SLA@SOI framework will take some time to start. In order to verify the completion of the launch of framework, after sometime type on the OSGi console the command 'ss'. You should be able to see all the bundles in ACTIVE state except for fragment bundles (which are not meant for ACTIVE state).

■ Result

The framework instance is now up and running, thus waiting for request to be fulfilled.

8 *References*

- [1] SLA@SOI: *Project Web Site*. July 2011. URL <http://www.sla-at-soi.eu>
- [2] SLA@SOI: *Reference Architecture for an SLA Management Framework*. Whitepaper, version 2.0, July 2011.
URL: http://sourceforge.net/apps/trac/sla-at-soi/browser/platform/trunk/doc/SLA%40SOI-Reference_Architecture.pdf
- [3] SLA@SOI: Open Source project. <http://sourceforge.net/projects/sla-at-soi/>
- [4] SLA@SOI: SLA@SOI Tutorial based on Open Reference Case. Whitepaper. July 2011
URL: <http://sourceforge.net/apps/trac/sla-at-soi/browser/platform/trunk/doc/SLA%40SOI-tutorial.pdf>
- [5] M. P. Shanahan, "The Event Calculus Explained", In: *Artificial Intelligence Today*, LNAI no. 1600:409-430, Springer, 1999.
- [6] SLA@SOI Work Package A5, "Deliverable D.A5a: Foundations for SLA Management", 09/2010
- [7] Amálio N., Stepney S., and Polack F. A formal template language enabling meta-proof. In *FM 2006*. 2006: LNCS, Springer
- [8] Amalio N., Di Giacomo V., Kloukinas C., Spanoudakis G. A4.D4.1_Mechanisms for detecting potential S&D Threats. SERENITY Project 2008.

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 <i>service level agreement</i> . 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 <i>agreement initiator</i> to <i>agreement responder</i> during agreement creation, indicating the relationship which the initiator would like to form.
Agreement Responder	The agreement responder is a party to a <i>service level agreement</i> . 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 <i>agreement responder</i> to advertise the types of offers it is willing to accept.
Agreement Term	Agreement terms define the content of a <i>service level agreement</i> .
Business Service	A business service is exposed/invoked via at least some non IT elements.
Business Manager	A specialization of <i>service provider</i> : 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 <i>service provider</i> : 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 <i>infrastructure provider</i> : 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 <i>infrastructure services</i> .

Infrastructure Service	An infrastructure service is a specific <i>IT service</i> 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 <i>Service Provider</i> to its <i>Service Customers</i> .
Operation Level Agreements	A specification of the conditions under which an <i>internal service</i> 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 <i>service interface type, service concreteness, service exposure</i>
Service Concreteness	The stage a service reaches over time from a fully abstract type to actually instantiated. See also <i>service type, offered service, service implementation, service instance</i>
Service Consumer	Person(s) who actually consume/use the provided services. Typically they belong to the <i>service customer</i> .
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 <i>service level agreement</i> . 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 <i>internal service, external service</i>
Service Implementation	A service implementation is a possible concrete realization of a given <i>service type</i> .
Service Instance	A concrete realization of an <i>offered service</i> which is ready for consumption by service users. It relies on the instantiations of all the resources required for a given <i>service implementation</i> .
Service Interface Type	Describes the nature of an actually exposed service, i.e. about the nature of his invocation interface. See also <i>business service, IT service, hybrid service</i>
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 <i>agreement</i> . Syntactically, it is an assertion over the agreement <i>terms</i> 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 <i>service provider</i> : person/system that is responsible for managing SLATs and SLA relationships.
Software Designer	A specialization of <i>software provider</i> : person that designs/develops the architecture and components of a specific SLA based application.
Software Manager	A specialization of <i>service provider</i> : 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 <i>software components</i> which might be used by a <i>service provider</i> to assemble actual <i>services</i> .
Software Service	A software service is a specific <i>IT service</i> 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 <i>software provider</i> .
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