



*Empowering the Service Economy with  
SLA-aware Infrastructures*



**Project no.** FP7- 216556  
**Instrument:** Integrated Project (IP)  
**Objective ICT-2007.1.2:** Service and Software Architectures, Infrastructures and Engineering

# Deliverable D.B2b

## Reference Demonstrator

**Keywords:**

Web Service, Open Reference Case, CoCoME, Reference Demonstrator, Service Level Agreement, Service-Oriented Infrastructure

**Due date of deliverable:** 31<sup>st</sup> July 2011  
**Actual submission to EC date:** 29<sup>th</sup> July 2011

**Start date of project:** 1<sup>st</sup> June 2008  
**Duration:** 38 months

**Lead contractor for this deliverable:** XLAB

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination level		
PU	Public	Yes



Document Status	
Deliverable Lead	Miha Stopar, XLAB
Reviewer 1	Jaka Močnik
Reviewer 2	Peter Chronz
PMT Reviewer	Wolfgang Theilmann
Complete version submitted to reviewers	5 July 2011
Comments of reviewer 1 received	8 July 2011
Comments of reviewer 2 received	11 July 2011
Deliverable submitted to PMT	18 July 2011
PMT Approval	20 July 2011

Contributors	
Partner	Contributors
XLAB	Miha Stopar, Primož Hadalin
FZI	Christoph Rathfelder, Benjamin Klatt

Notices
<p>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.</p>
<p>* Other names and brands may be claimed as the property of others.</p>



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
V0.1	23/5/2011	Miha Stopar (XLAB)	Initial structure of the document.
V0.2	31/5/2011	Christoph Rathfelder (FZI)	Updated ORC section
V0.3	9/6/2011	Christoph Rathfelder (FZI)	First Draft of Evaluation
V0.4	19/7/2011	Miha Stopar (XLAB)	Final corrections

# Executive Summary

This document presents the Open Reference Demonstrator, the demonstrator used by the Integrated European Research Project SLA@SOI to showcase the results of the scientific work performed in the project. The demonstrator is both public and open-source and thus available to the widest possible range of stakeholders, interested in the implementation of a Service Level Agreement aware Service-Oriented Infrastructure. These include but are not limited to service providers, the scientific community and developers.

The Reference demonstrator is based on the Open Reference Case (ORC) - a reference application supporting the sales process in a retail-chain. The ORC is implemented on top of legacy components and it was adapted to the service-oriented application. ORC was extended to support various deployment options and levels of separations as well as with additional software hooks that facilitate run-time monitoring and SLA-awareness of the application. Detailed information about ORC installation and deployment is documented in [ORC Deployment Guide](#) document [1].

The Reference demonstrator is comprised of a graphical user interface that supports the interaction of the user with the SLA@SOI final framework version (version v2, as v0 was the framework version from the first year of the project), of an ORC application, and of a workload generator that simulates the consumption of the ORC services. The GUI provides a view on many aspects of the SLA@SOI framework, namely customer's perspectives (negotiation for the quality of ORC web services, re-negotiation, SLA warnings, violations and penalties, status and state of the ORC services), provider's perspective (services response times and arrival rates, SLA warning and violations, penalties) and developer's perspectives (framework internal messages, SLA hierarchy, interaction between framework components).

As the Open Reference Demonstrator is public and open source, the source code and detailed documentation providing installation and usage instructions are publicly available on Source Forge ([Open Reference Demonstrator guide](#) [2]). This document provides an overview of the Open Reference Demonstrator design and implementation, as well as a description of the ORC modifications - a detailed analysis of modifications necessary for an application to become SLA-aware. In the final chapters, the framework architecture is described and an evaluation of the framework implementation is given.

Documentation on how the SLA@SOI framework is applied in order to allow the SLA-driven management of a service-oriented application can be found in the [SLA@SOI tutorial](#) document [3], while documentation on how to use the SLA@SOI Studio to help use case developers configure and deploy the SLA@SOI framework, is provided in [SLA@SOI Studio](#) document [4].

# Table of Contents

1	Introduction .....	8
1.1	Context and Scope .....	8
1.2	Document Overview .....	9
2	Open Reference Case Specification .....	10
2.1	The Adapted ORC Scenario.....	10
2.1.1	ORC Stakeholders .....	10
2.1.2	Business Process.....	12
2.2	ORC Architecture .....	12
2.2.1	ORC Services .....	13
2.2.2	ORC Configuration Services.....	16
2.3	Specification of deployment options .....	18
2.4	Integration of ORC with the SLA framework .....	20
2.4.1	Manageability Configuration and Instrumentation of the ORC .....	20
2.4.2	SLA@SOI Models .....	20
3	Architecture of the Reference Demonstrator.....	20
4	Open Reference Demonstrator Implementation .....	24
4.1	Simulation Environment.....	24
4.1.1	Introduction .....	24
4.1.2	The sales process.....	26
4.1.3	Configuration options .....	27
4.2	Simulation Environment Implementation.....	30
4.2.1	WorkloadGenerator.java .....	30
4.2.2	AdhocJobManager.java .....	31
4.2.3	Agent.java .....	32
4.2.4	Generating Java code from WSDL .....	32
4.3	Graphical User Interface .....	33
4.3.1	Introduction .....	33
4.3.2	Graphical User Interface Implementation .....	34
5	Evaluation .....	36
5.1	The Goal/Question/Metric Approach .....	36
5.2	GQM-based Evaluation.....	37
6	Conclusions.....	49
6.1	Contributions and Achievements .....	49
6.2	Lessons learned .....	49
6.3	Outlook .....	50
7	References.....	51

# Table of Figures

Figure 1: ORC Stakeholders .....	11
Figure 2: ORC Scenario Overview .....	11
Figure 3: Sales Process. ....	12
Figure 4: ORC Architecture Overview .....	13
Figure 5: Inventory Service Interface .....	13
Figure 6: Store Information Service Interface. ....	14
Figure 7: Order Service Interface. ....	14
Figure 8: Payment Debit Service Interface.....	15
Figure 9: Card Validation Service Interface.....	15
Figure 10: Payment Service Composition. ....	15
Figure 11: PaymentService BPEL Process. ....	16
Figure 12: ORC Deployment Options.....	19
Figure 13: Negotiation interactions .....	21
Figure 14: Framework internal negotiation interactions .....	22
Figure 15: Interactions during creation of an agreement .....	22
Figure 16: Provisioning of services triggered by Service Manager .....	23
Figure 17: The architecture of the simulation environment. ....	25
Figure 18: The sales process workflow. Normal service calls are displayed in light blue. Dark blue rectangles are used for delays. Decision symbols are depicted in orange colour with the name of the probability used. ....	27
Figure 19: Relations between goals, questions, and metrics.....	36

## List of Tables

Table 1: Document changes against previous version .....	9
Table 2: An example of the JSON formatted workload configuration. ....	28
Table 3: Description of global configuration properties. ....	29
Table 4: Agent configuration properties.....	29
Table 5: Description of workflow probabilities controlling branches in the workflow. ....	29
Table 6: Description of workflow delay. Each delay is described in terms of the normal distribution, thus there are two arrays used (delay and randomStandardFactor). ....	30
Table 7: Main classes and packages constituting the Simulation Environment ...	30
Table 8: Example of manager.properties configuration. ....	31
Table 9: Example of a function, annotated with the XOSDCONSOLE annotation. ....	32
Table 10: JAX-WS pom.xml configuration. ....	32
Table 11: Axis pom.xml configuration. ....	33
Table 12: Reference Demonstrator Java Web Applicatoin packages.....	35
Table 13: Goals, Questions and Metrics.....	37

# 1 Introduction

## 1.1 Context and Scope

---

The purpose of the work package B2 of the European research project SLA@SOI is to provide a reference service-oriented application (Open Reference Case) and to integrate it with the SLA@SOI framework in order to have a public example, demonstrating the use of the framework. The results of this work package are provided as an open-source software and publicly available documentation on the Source Forge. The documentation on the Source Forge project website includes instructions on how to install the Open Reference Demonstrator components, instructions on how to use Reference demonstrator and [SLA@SOI tutorial](#) based on the Open Reference Case. Documentation on ORC Deployment can be found in the [ORC Deployment Guide](#) document, while instructions for Reference Demonstrator installation and usage can be found in [Open Reference Demonstrator guide](#) document.

Because Open Reference Demonstrator documentation is publicly available on the Source Forge this deliverable doesn't provide every detail of the Reference Demonstrator and its components, it provides information about the work process, information about legacy components, references to the Source Forge documentation and an evaluation of the SLA@SOI framework.

Open Reference Demonstrator is an extended demonstrator from the first year of the project (called Ad-hoc demonstrator). The main components remain the same (ORC service-oriented application, frontend application, workload generator – simulation environment), but with quite significant extensions and with a new binding packages enabling the integration with the framework version from Y3.

The service-oriented application (Open Reference Case) is based on Common Component Modelling Example (CoCoME)[6], a legacy application supporting the sales process in a retail chain. For the purposes of the SLA@SOI demonstrators, the application was first extended with additional Web Service layer on top of CoCoME components. A diverse set of deployment options was achieved by complete separation of the application logic from the data model and the service composition layer.

Finally, the application was transformed into a manageable and SLA-aware application ready to be deployed with the SLA@SOI framework. This step included an implementation of the instrumentation of services in order to monitor the behaviour of service invocations and a preparation of a design-time prediction model used during the negotiation and resource planning phase.

The purpose of the simulation environment is to simulate a workflow of a real-world sales process, thus generating load on the deployed application. It is used to showcase the behaviour of the SLA@SOI framework under different conditions. The workload is configured with a usage profile described in terms of number of concurrent cashboxes and patterns practiced by store's customers. The patterns are defined with an expected number of purchased items, delays simulating human actions, and probabilities of certain actions within the workflow.

The demonstrator's graphical user interface enables browsing through the product catalogues, gathering information about respective SLA templates, it enables customizing SLA templates and sending an SLA offer to the framework. When the

SLA is achieved, the GUI enables view on the user’s SLAs status and potential SLA violations. Furthermore, it enables the insight in the framework internal processes. Finally, it enables configuration of the simulation environment, its starting and run-time adaptations in order to monitor the SLA@SOI framework reactions on the different workload on the ORC services.

Evaluation of the Open Reference Demonstrator and, in particular of the final version of the SLA@SOI framework is based on a well-known Goal/Question/Metric approach. A series of goals has been developed and aligned with requirements from business, service provider and developer perspective.

## Main achievement

The main achievement of this work package is the **successful integration of all major SLA@SOI framework components with the Open Reference Demonstrator GUI and Simulation components**. This integration and the documentation available on Source Forge can serve as a starting point for those who are interested in the development of the SLA-aware applications using SLA@SOI framework.

## 1.2 Document Overview

The remainder of the document is structured as follows. Section 2 introduces service-oriented retail chain solution supporting the sales process. Part of the section also describes various deployment options. The modifications required for the integration with the SLA management framework and the installation of the ORC are described in separate documents available online.

Section 3 provides an overview of the Open Reference Demonstrator architecture and of the main interactions between various demonstrator components.

Section 4 is dedicated to the Open Reference Demonstrator implementation, in particular to the simulation environment and the graphical user interface allowing interested stakeholders to conduct experiments.

The evaluation plan and the actual evaluation with a detailed analysis of the SLA@SOI framework are presented in Section 5.

Conclusions are presented in Section 6.

**Table 1: Document changes against previous version**

Section	Change overview
Section 2	ORC adaptations updated to reflect Y3 work
Section 3	Revised, diagrams updated
Section 4	Revised and updated to reflect Y3 work, in particular new demonstrator GUI
Section 5	The same evaluation as in Y1 executed once again at the end of the project
Annex C	new
Annex D	new
Annex E	new

## 2 Open Reference Case Specification

The Open Reference Case (ORC) is used as an open source demonstrator highlighting features provided by the SLA@SOI framework. The ORC demonstrates the use of the SLA management framework in the context of a service-oriented retail solution supporting the sales process in supermarkets. More in detail, the ORC includes IT support for retail chains in general, covering enterprise headquarters (central management issues), stores (local management) and cash desks. Several shop providers, each with a certain number of stores, are connected to a single service provider, supporting sales of goods with an IT system. The ORC software can run on a virtualized infrastructure using various deployment options, to cover small as well as large shops with different requirements regarding the performance of the ORC system.

The underlying scenario of the ORC extends the Common Component Modelling Example (CoCoME)[6], which was introduced for comparing the facets of several well-known component models. In the following we give an overview of the adaptations made for highlighting the features of the SLA@SOI framework.

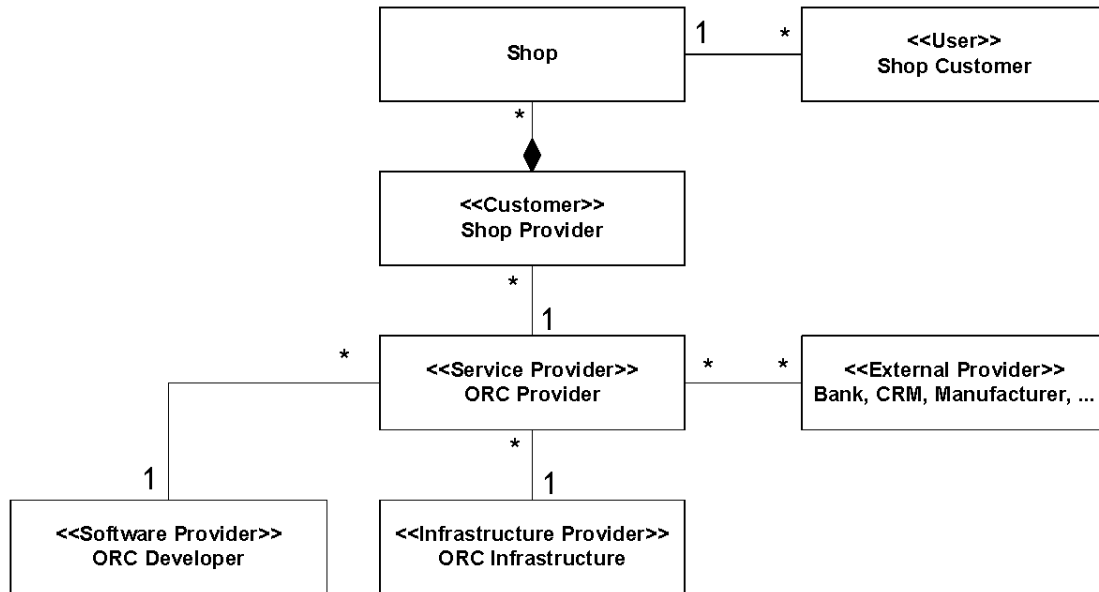
### 2.1 The Adapted ORC Scenario

---

The ORC is a service-oriented retail solution that can be used in a trading system of a supermarket to handle the sales and stock processes. The use of the ORC as Software as a Service (SaaS) promises several benefits. It should reduce the operational costs of the supermarket on one hand and reduce the waiting time of customers at a cash desk on the other. For this reasons the two most important quality characteristics that need to be considered in SLAs are the costs and the completion time of service invocations. The ORC supports the payment process at the cash desk, and thus reliability of the services plays an important role and should be reflected in the SLAs. From the service provider's point of view, it is required to specify and limit the load that is induced by a certain customer on his systems. This knowledge is required to determine the required infrastructure services. For this reason an additional important quality characteristic is the load on the system, which is measured in invocations per second. Using the ORC as a SaaS solution we use a slightly adapted scenario that is presented in the following more in detail.

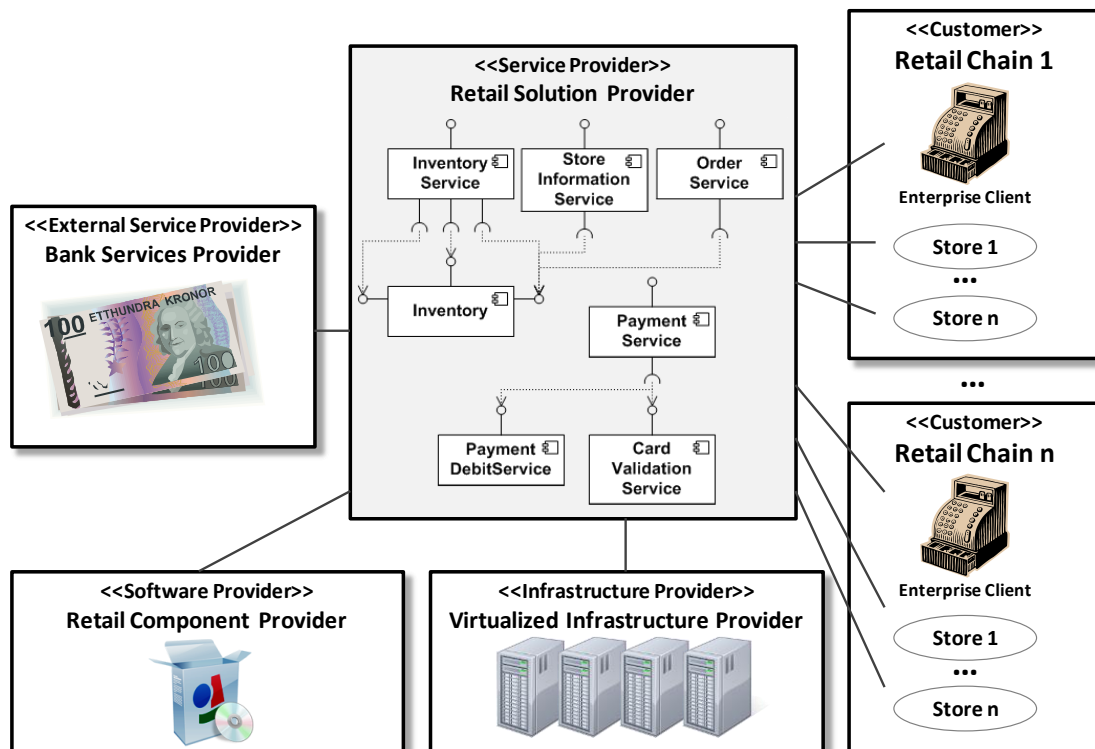
#### 2.1.1 ORC Stakeholders

For highlighting the features of the SLA@SOI framework the original CoCoME scenario is modified towards an SaaS scenario, where an overview of the involved stakeholders and their bindings during runtime are given in Figure 1.



**Figure 1: ORC Stakeholders**

A key element - the ORC Provider - is added to the scenario as a single Service Provider to which several Shop Providers are connected. The Service Provider enables the shop providers to access several additional services such as inventory management, credit card payments, preferred customer club cards, accounting etc. For providing services such as payment and credit card handling, wholesale centres, or CRM suppliers, the Service Provider can either use its realizations or make use of external services. The ORC service itself is running on top of a virtualized IT infrastructure offered by an additional provider - the ORC Infrastructure. To complete the scenario we assume the ORC to be designed and implemented by an independent Software Provider namely the ORC Developer.



**Figure 2: ORC Scenario Overview**

In Figure 2 we present the view on the structure of a concrete scenario, where we assume that the Service Provider only makes use of the Bank Service as an external service. To provide other “additional services” the Service Provider makes use of its own realizations.

### 2.1.2 Business Process

In the overall scenario the sales process is identified to be the key business process. In order to realize the sales process additional services need to be accessed. For example the functions ValidateCard and DebitAmount are provided by the external bank service, which are accessed via the ORC service. In Figure 3, the sales process at a cash desk in a supermarket and its sub-processes are sketched. The process starts with scanning the goods. For each good, the bar code is scanned and the GetProductDetails operation of the InventoryService is called. If the detected product code is correct, the stock is updated. If not, the product code has to be entered manually. After that, the Payment is handled automatically, if credit card payment is selected. First, the card is scanned and then validated using the CardValidationService. If it is valid, the payment is done using the PaymentDebitService. If not, the card is rejected and the payment has to be done manually using cash.

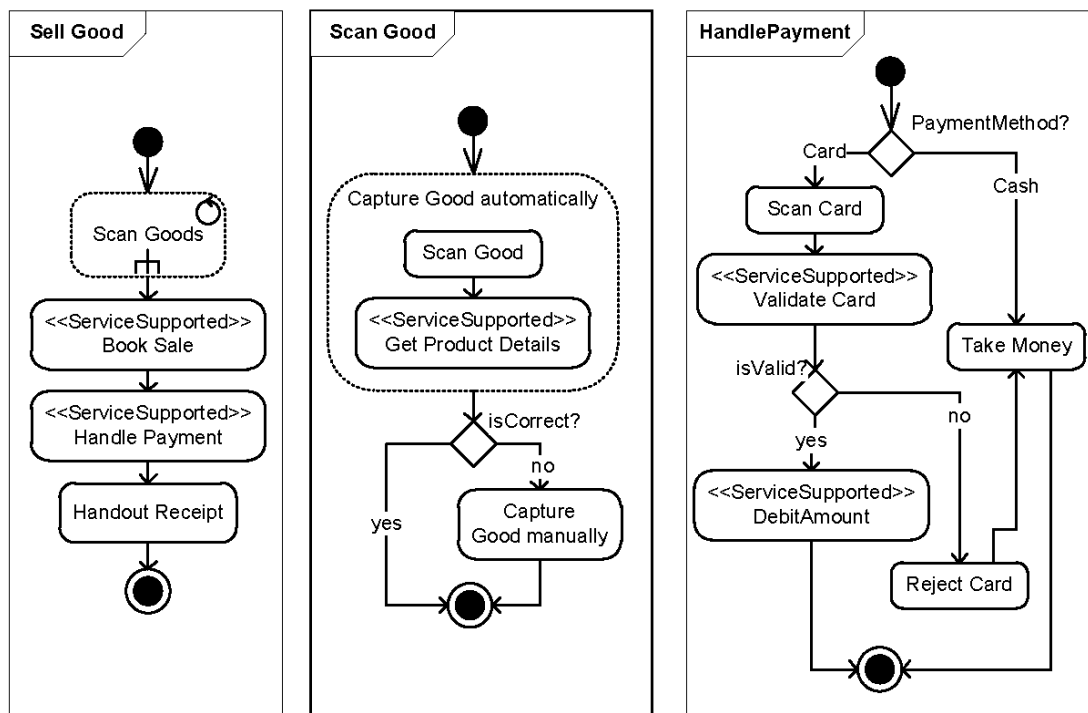
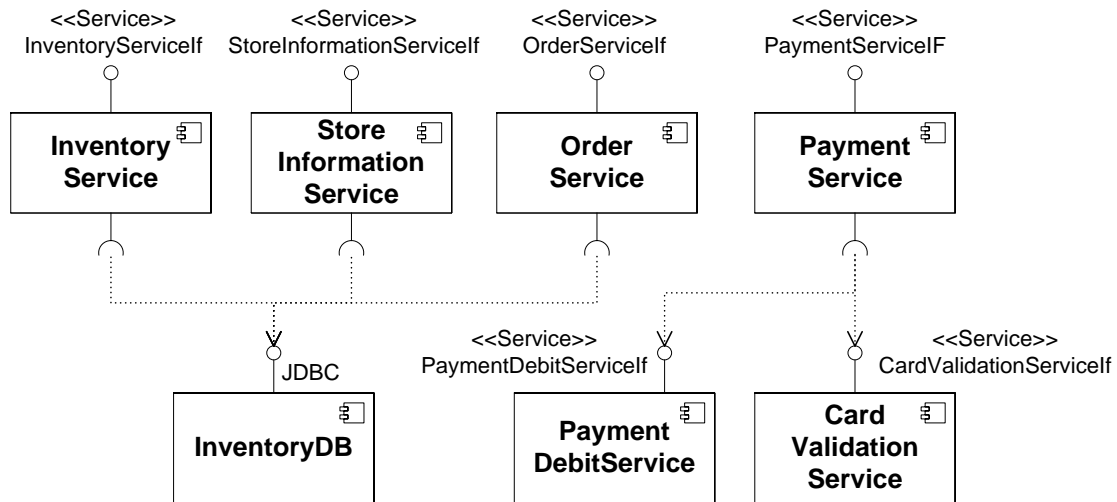


Figure 3: Sales Process.

The business process is thereby supported by different services and service compositions, defined in the preceding section.

## 2.2 ORC Architecture

In the following, the architecture of the ORC is described in more detail using the UML 2.0 syntax. Therefore a structural view on the system is given. Figure 4 presents an overview of the implemented system, containing the legacy CoCoME components and the additional web service components.



**Figure 4: ORC Architecture Overview**

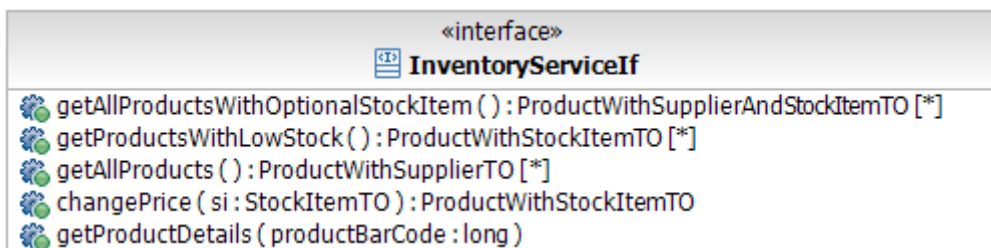
The five components `InventoryService`, `StoreInformationService`, `OrderService`, `PaymentDebitService` and `CardValidationService` implement web service interfaces. `InventoryService`, `StoreInformationService`, and `OrderService` require the functionality provided by the legacy CoCoME component `Inventory`. The `Inventory` component provides three interfaces. The interface `CashDeskConnectorIf` defines a method for getting product information like description and price based on the products' barcode. The interfaces `StoreIf` and `ReportingIf` deliver results of database queries. The `PaymentDebitService` and `CardValidationService` components provide functionality for handling credit card payments. Such services are often provided by a bank, however they can also be implemented internally.

### 2.2.1 ORC Services

In this section, the web services and their interfaces, which form the base of ORC scenario, are described in more detail.

#### Inventory Service

The `InventoryServiceInterface` (Figure 5) defines operations for getting product information and changing prices. It is used to get the current price of a product but also to check its availability in the stock and remove it from stock after it is sold. The interface consists of five operations:



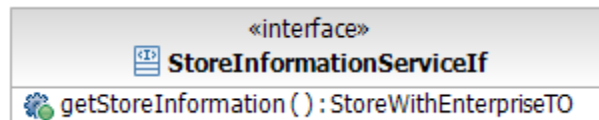
**Figure 5: Inventory Service Interface**

- `getAllProductsWithOptionalStockItem` determines all products of the portfolio of a given store and the supplier for each of them. Additionally the corresponding stock items are queried. It returns a list of products, their suppliers and the corresponding stock item if they have any.

- `getProductsWithLowStock` determines products and stock items which are nearly out of stock, meaning amount is lower than 10% of maximal stock. It returns a list of products and their stock item in the given store.
- `getAllProducts` determines all products of the portfolio of a given store and the supplier for each of them. I.e. it returns a list of products and their suppliers
- `changePrice` updates the sales price of a stock item. Needs the stock item with the new price as input and returns an instance of `ProductWithStockItemTO` which holds product information and updated price information for the stock item identified by the parameter `StockItemTO`.
- `getProductDetails` determines the product and the item in the stock of the store by the given barcode. It returns a `ProductWithStockItemTO` instance, which contains the identified product, which is linked to the stock item of the store.

### Store Information Service

The `StoreInformationServiceInterface` (Figure 6) provides a method for getting information about a store.



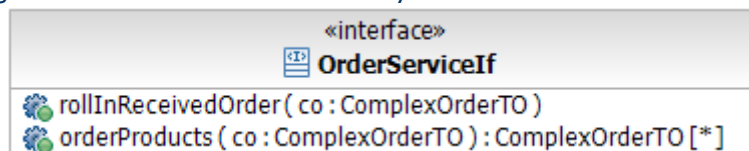
**Figure 6: Store Information Service Interface.**

The Interface consists of one method:

- `getStoreInformation` gets the transfer object with information of the store and its enterprise. This information is retrieved by the component during configuration and initialization.

### Order Service

The `OrderServiceInterface` (Figure 7) defines methods for ordering products and updating the stock after order delivery.



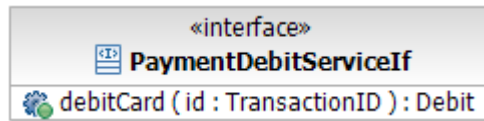
**Figure 7: Order Service Interface.**

The Interface consists of two methods:

- `rollInReceivedOrder` updates stocks after order delivery. Therefore it adds the amount of ordered items to the stock items of the store and sets the delivery date to the date of method execution. Requires an instance of `ComplexOrderTO`, which contains the order that is rolled in, as parameter.
- `orderProducts` creates a list of orders for different suppliers for an initial list of products to be ordered by a store. The product order is persisted and the ordering date is set to the date of method execution. The method requires an instance of `ComplexOrderTO`, which contains all products to be ordered and returns a list of orders, one for each supplier that is affected.

### Payment Debit Service

The `PaymentDebitServiceInterface` (Figure 8) provides a method for debiting of payments.



**Figure 8: Payment Debit Service Interface.**

The Interface consists of one method:

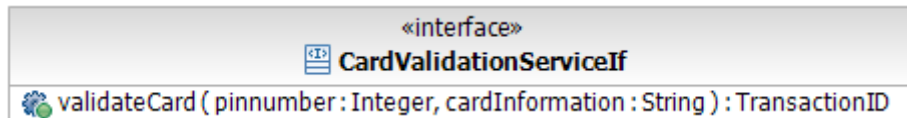
- `debitCard` is used to debit a bank account. Possible return values are OK, NOT\_ENOUGH\_MONEY and TRANSACTION\_ID\_NOT\_VALID. Requires a `TransactionID`, which is issued with a valid PIN.

**FastPaymentDebitService**

The `FastPaymentDebitService` also provides the `PaymentDebitServiceInterface` and thus similar functionality as the `PaymentDebitService`. The difference between these two services is in the fact, that the `PaymentDebitService` can be slowed down using the `BankConfigurationService`, which is explained in the next section. The `FastPaymentDebitServices` always provides a constant completion time.

**Card Validation Service**

The `CardValidationServiceInterface` (Figure 9) defines a method for validating a credit card.

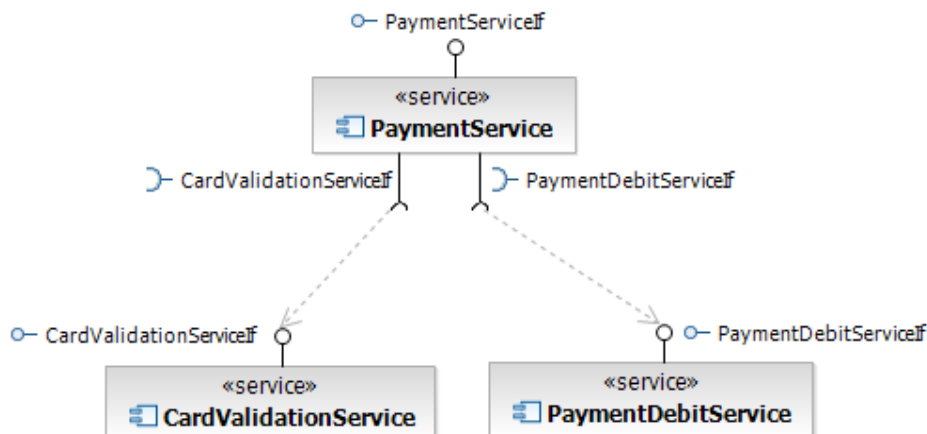


**Figure 9: Card Validation Service Interface.**

The interface consists of one method: `validateCard` is used to validate a credit card. It requires the PIN and some information about the card. The method returns a transaction id, which can be used for debiting the payment.

**Payment Service as Service Composition**

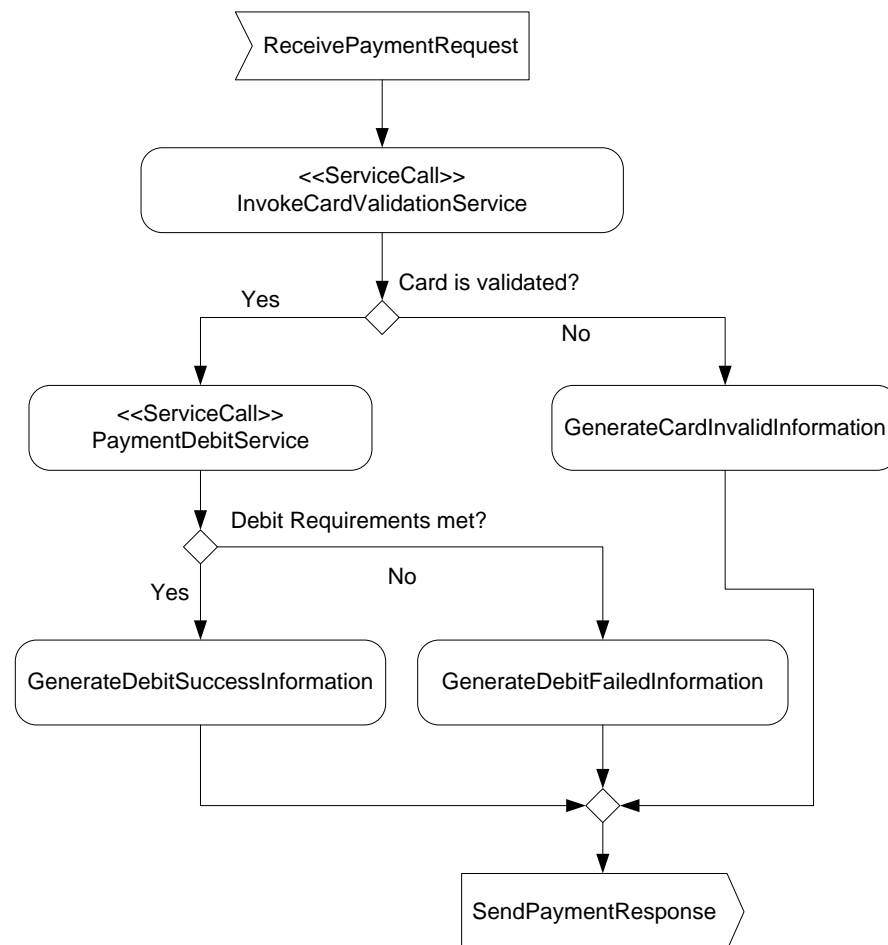
The `PaymentService` in Figure 10 orchestrates the web services `CardValidationService` and `PaymentDebitService` to handle the payments.



**Figure 10: Payment Service Composition.**

The `PaymentService` is implemented as WS-BPEL [9] process. It is deployed on the ActiveBPEL Engine [10] used as the run-time web service composition middleware. The subprocess `PaymentService` includes card validation and debit

payment and is required to support the whole Sale Process. Since the corresponding web services `CardValidationService` and `PaymentDebitService` are already available as basic web services the `PaymentService` can be built up by consuming these two services.



**Figure 11: PaymentService BPEL Process.**

As depicted in Figure 11, as soon as the payment request is received, the provided card information is validated. In case the card validation fails, the `CardInvalidInformation` is generated. If the card is validated, the second step is to validate if the debit requirement on this card is fulfilled. If the requirement is fulfilled, the `PaymentIsSuccessful` information is generated. Otherwise, the `DebitFailed` information is generated. Last but not least, all the generated information is sent back to the cashier as the `PaymentResponse`. In such a way, the card payment request can be automatically processed.

## 2.2.2 ORC Configuration Services

In addition to the services already described in the previous section, the ORC provides two configuration services, which support the simulation of different scenarios in the SLA framework.

- The **BankConfigurationService** can be used to simulate an SLA violation caused by the software layer. The service provides one operation with a boolean parameter, which allows to specify if the `PaymentDebitServices` should be slowed down or not. This functionality can be used to simulate

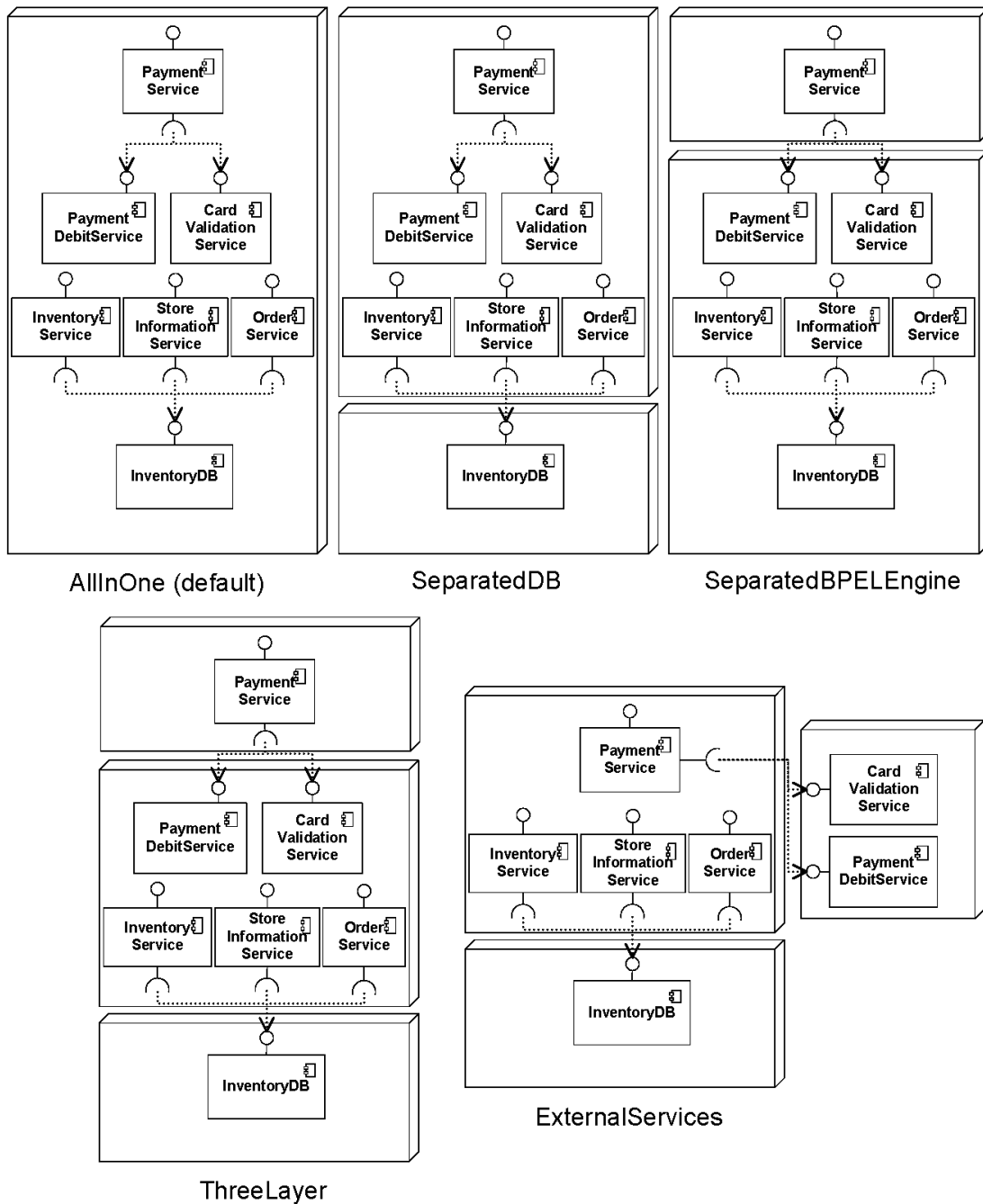
SLA violations, which can be detected by the framework and might lead to adjustment action.

- **ORCConfigurationService** is used to reconfigure the network connections. This reconfiguration is realized by manipulating the hosts file of the system. For this reason the operation provided by the service requires an array of (hostname, IP) tuples. The implementation of the ORC uses host names ORCBasicServices and ORCDatabase to communicate with the host running the basic services (InventoryService, StoreInformationService and OrderService) and with the host running the database.

## 2.3 *Specification of deployment options*

---

The ORC software solution can be deployed on one single machine as well as distributed over up to three machines. This allows selecting the most appropriate option depending on the extra-functional requirements like completion time or cost depending on the requirements and the expected amount of customers in the shop. It is also possible to use the ORC as a multi-tenant system that handles several different shops on one single ORC instance. Figure 12 illustrates some of these deployment options. Although the services are quite independent, there are some constraints limiting the deployment options. The three services `InventoryService`, `StoreInformationService`, and `OrderService` must be deployed on the same machine, as they are service wrappers for the legacy software components running in the back-end, which have some shared functionality and configuration. The `CardValidationService` and the `PaymentDebitService` are also bundled in one deployment unit, as separation of card validation and debiting makes no sense, being always provided by the same institution. In the SLA@SOI framework, we use virtualized machines, which allows us to deploy new instances by copying existing images and starting them. This dramatically reduces the effort compared to manual installation on physical hosts. However, it is possible to manually install the ORC without virtualization.



**Figure 12: ORC Deployment Options**

- **AllInOne:** This deployment option consists of only one single virtual machine image. This image contains the database as well as the basic services and the composite services which include the application logic of the ORC.
- **SeparatedDB:** The second bundle is a virtual appliance consisting of two virtual machine images. One provides an application server with all deployed services and the other hosts only the database.
- **SeparatedBPELEngine:** The DB consumes only a small part of the available CPU resources. Most CPU power is consumed in the web service container and the BPEL engine. Therefore this deployment option separates the BPEL engine on an individual virtual machine. Basic Services and the database are deployed on another virtual machine.

- **ThreeLayer:** This deployment option is a combination of the two previous ones. Database as well as BPEL engines are separated from the basic services. So this deployment option consists of three VMs.
- **ExternalServices:** ORC can also be used to demonstrate the involvement of an external service provider. It is possible to transfer the two services CardValidationService and PaymentDebitService to an external virtual machine. By this, they can be used as if they were provided by an external service provider including the negotiation of SLAs with this external service provider.

## 2.4 Integration of ORC with the SLA framework

To use the SLA@SOI framework in the ORC scenario, some ORC specific configurations, adaptations, and extensions are required. The general required steps are documented within the [adoption guide](#) [7]. Among others the activities include the definition of SLATs, the specification of prediction and service construction models. To enable the management of ORC a manageability interface had to be designed and developed. These ORC specific management extensions include instrumentation of the services that provides monitoring of the start and stop of service calls. The instantiation of the framework including the ORC specific extensions and adoptions is documented within the [SLA@SOI tutorial](#) [3].

### 2.4.1 Manageability Configuration and Instrumentation of the ORC

To create a manageable version of the ORC we followed the engineering methodology presented in deliverable D.A6a section 4.3. Accordingly, we first created a manageability model for the PaymentService and the referenced atomic services on basis of the existing ORC component model. For meeting requirements from SLA adjustment (see deliverable D.A5a, section 4.4) this model on one hand defines that management information of all provided and referenced operation calls should be available. On the other hand the model specifies that it should be possible to correlate the operation calls of the atomic services with the operation calls of the composite PaymentService.

For detailed specification of Manageability configuration and instrumentation of the ORC refer to [SLA@SOI tutorial document](#).

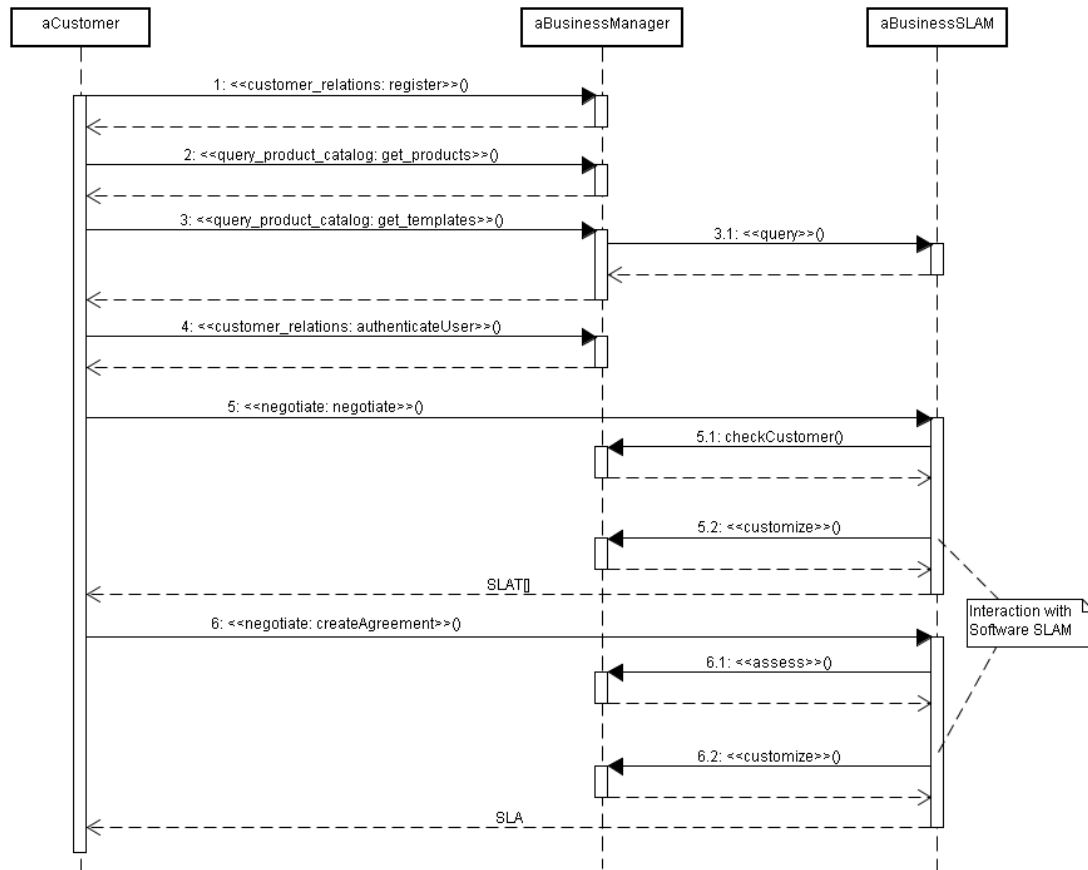
### 2.4.2 SLA@SOI Models

SLA@SOI models need to be adopted in order to enable the SLA management of an arbitrary application, namely the SLA model, the SCM (Service Construction Model) and the prediction model. Detailed documentation about models adoption for ORC can be found in the [SLA@SOI tutorial document](#).

## 3 Architecture of the Reference Demonstrator

Open Reference Demonstrator of the SLA@SOI project is based on a simplified service-oriented retail chain application. The latter was the main driving force for

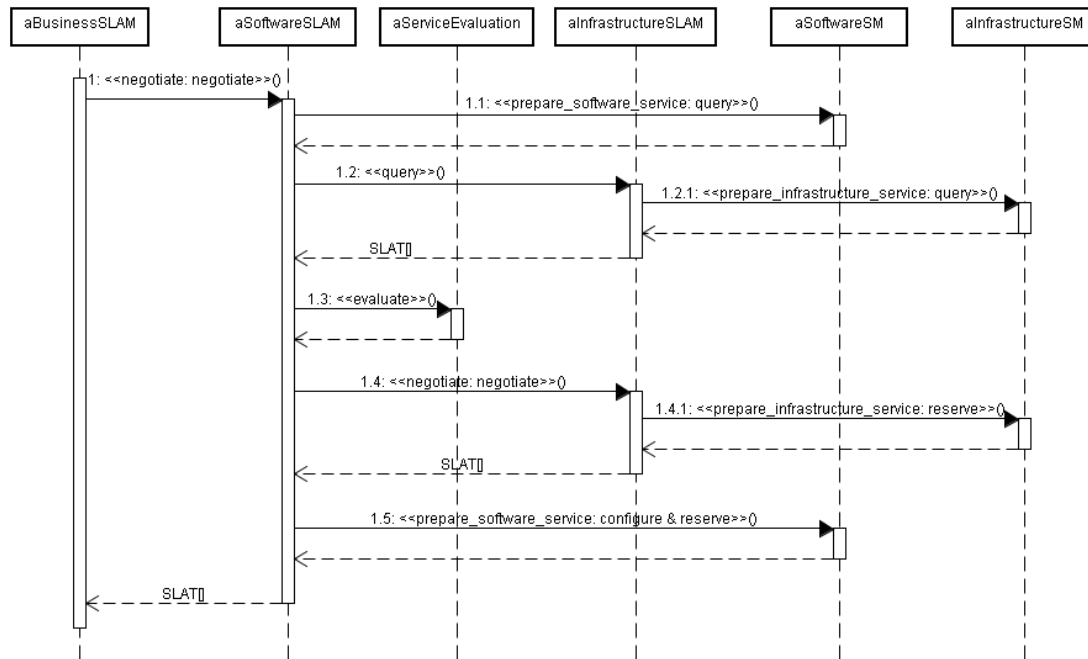
the design and implementation of the Open Reference Demonstrator architecture. The UI that was developed in order to demonstrate and visualise the SLA@SOI framework features is communicating with the framework via Business Manager. These interactions can be seen on the Figure 13. More details about framework interactions can be found in the [Reference Architecture guide](#) [5]. For descriptions of the UI component please see [Open Reference Demonstrator guide](#) [2].



**Figure 13: Negotiation interactions**

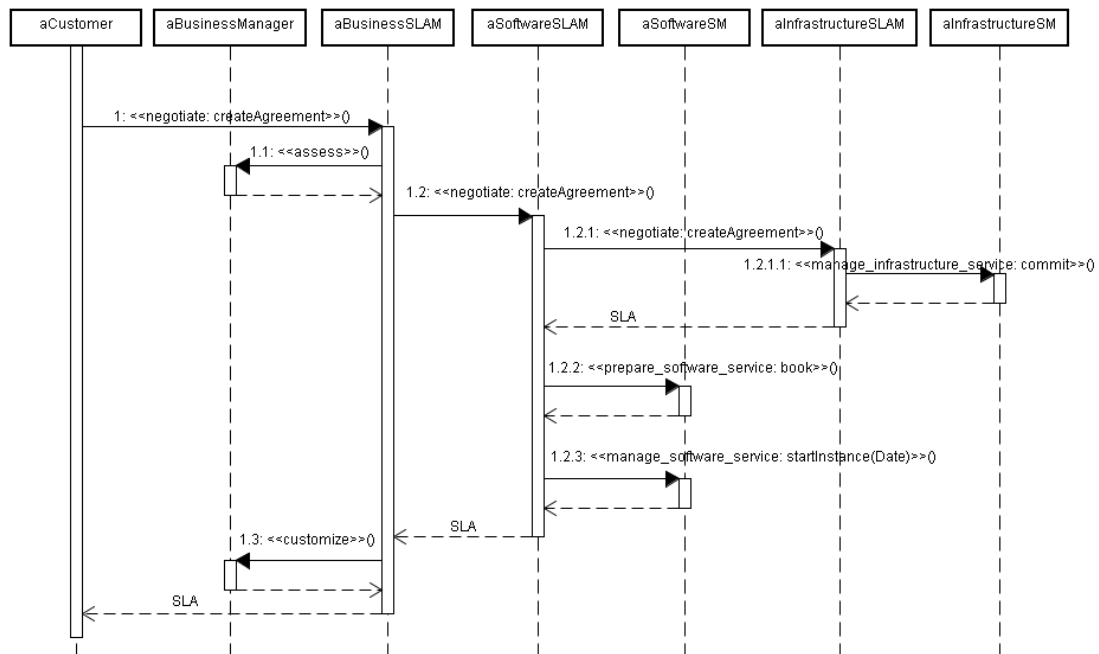
In addition to the communication with the Business Manager additional channels of communication were provided in order to retrieve internal actions of the framework and to visualize them. Components like Monitoring Manager, Software Monitoring, Infrastructure Monitoring, Service Evaluation, POCs (Planning and Optimization Components) and PACs (Provisioning and Adjustment Components) from different SLAMs (Service Level Agreement Managers), Business Manager and others have implemented XMPP messages with information about internal actions. These messages are shown in the UI Framework Internals view, where each of these components is visualized. A third way of communication is provided by a presentation of log4j messages [23] in the UI. A log4j XMPP appender has been developed, which enables visualizing of the framework log4j messages. The boundary between the SLA@SOI framework and the UI is mainly represented by interfaces provided by Business Manager. Business Manager allows for user registration, user authentication, it allows listing of the available products and available SLA templates for the product, and it allows starting the negotiation with sending the SLA offer, and finally it allows creation of the agreement.

Framework internal interactions in the negotiation phase are depicted in Figure 14. More details about presented phases can be found in the [Reference Architecture guide](#) [5].



**Figure 14: Framework internal negotiation interactions**

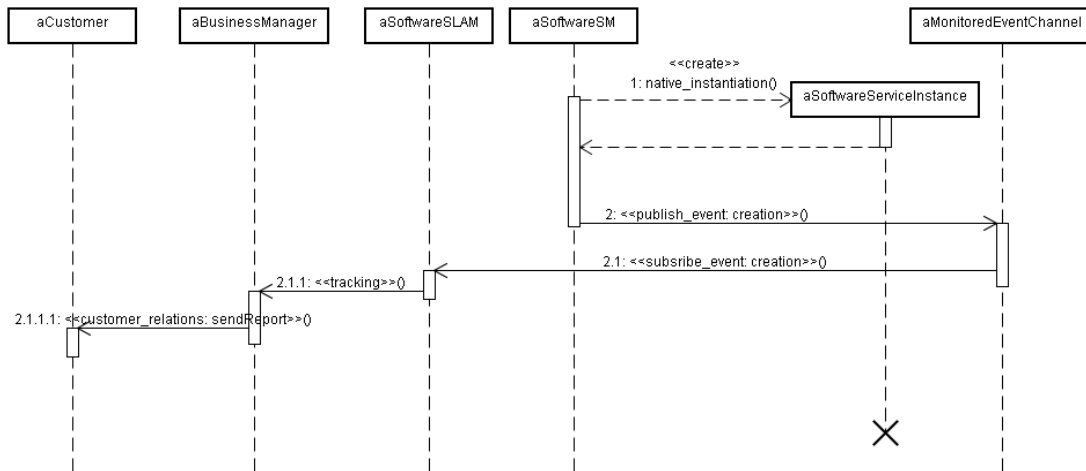
When the negotiation process has been completed, the customer can select one of the returned SLA templates and create an agreement. Internal framework interactions during the agreement creation are depicted in Figure 15.



**Figure 15: Interactions during creation of an agreement**

The creation of the agreement process books all the resources needed to provision the software and it schedules the provisioning of the service. Finally, the

agreed upon SLA is returned to the Business SLA Manager that returns it to the customer after a customization by the Business Manager.



**Figure 16: Provisioning of services triggered by Service Manager**

The last step is the provisioning of the service, which is depicted in Figure 16. For a more detailed description about the internal architecture of the framework please see [Reference Architecture guide](#) [5].

The Open Reference Demonstrator UI offers also a view for managing the simulation of the workload. The communication between UI and workload simulation (ORC user simulation) enables workload configuration (how heavy the user’s consumption of the ORC services should be) and visualizing of the load (completion time and arrival rate graphs for each of the ORC services – compared to the SLA thresholds). This communication also enables some further customization of the ORC application, for example configuration of the bank service, which can be used for slowing down the service execution to simulate the SLA violations.

# 4 *Open Reference Demonstrator Implementation*

The Open Reference Demonstrator consists of a graphical user interface and a simulation environment. The demonstrator can be deployed on an arbitrary infrastructure supporting the SLA@SOI framework. The graphical user interface provides three different perspectives on the execution environment:

- **Customer Interface** provides a simple service and SLA template browser. It also allows users to start negotiation with the service provider based on selected Service Level Objectives as well as monitoring of active SLAs.
- **Framework Overview** provides an overview of the execution environment in runtime to get a deeper understanding of the SLA@SOI framework. It does so by providing logs and a visual representation of the framework components.
- **ORC Simulation** provides means of controlling the actual usage profile of the customer, independent of the profile that was used during the negotiation phase. The purpose of this panel is to simulate different customer's usages and to monitor the behaviour of the SLA@SOI framework.

## 4.1 *Simulation Environment*

---

### 4.1.1 *Introduction*

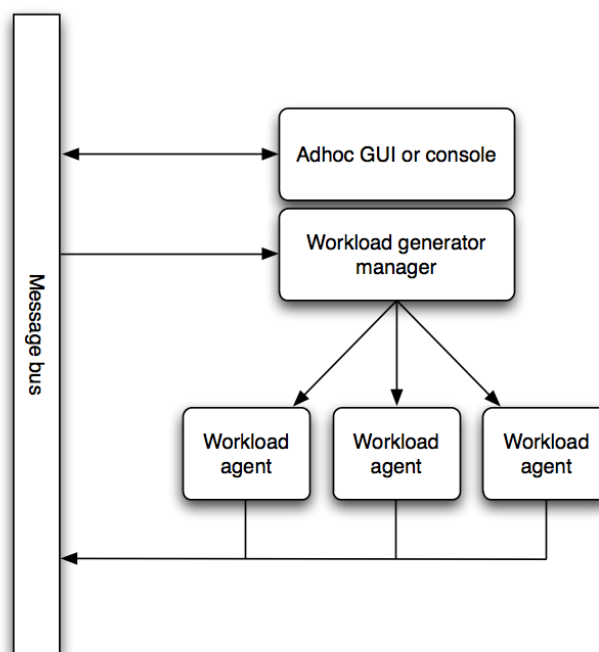
The simulation environment provides a tool for simulating a real-world sales process on multiple cash desks running in parallel. The simulation environment component remained the same as it was in the year 1, just a few additional configurations options have been added in order to simplify the configuration of the workload via UI. It consists of the workload generator manager responsible for the distribution of workload and a distributed and multithreaded network of workload agents invoking services offered by the service-oriented application. Although the sales process is hard-coded in every workload agent it provides configuration options for every step in the process allowing users to tailor the workload to their needs and/or actual usage profile of the customer.

The workload manager and agents are loosely coupled with the remaining components of the Reference demonstrator by using the message bus interface. Detailed description of the message bus can be found in Section 3.3.5 Infrastructure Messaging in D.A4a [11]. The message bus supports full-duplex communication, i.e., it is used to transfer commands from the workload manager panel/console to the manager as well as for reporting the status of each workload agent to the graphical user interface.

The workload agent is an abstraction that defines the sales process independently of the underlying Web Service access used. Workload implementation supports two client types, namely JAX-WS [12] integrated with Sun Java 6, and Apache Axis [13].

The architecture of the simulation environment is depicted in Figure 17. When the workload manager is initialized it starts listening to a pre-configured message bus channel for commands and configurations. It also starts sending its own heart-beat message to advertise its ability to receive commands. These messages contain ID of the manager to enable Open Reference Demonstrator to differ between multiple instances. Heart-beat messages are sent in regular intervals (1 second by default) and are used by the user interface to detect irresponsive workload managers. Initially, the manager does not start any workload agent.

Once the workload configuration is sent from the Reference GUI over the message bus all workload managers receive the message but only the one whose ID matches the ID in the request processes the message and initialises agents (threads). Initially, all threads are in *idle* state, waiting for further instructions. Once they are started they are reporting their status over the message bus, which is, eventually, displayed in the GUI panel.



**Figure 17: The architecture of the simulation environment.**

Besides heart-beat and configuration messages, there are three additional message types sent over the message bus: commands, command responses, and free text messages.

Using commands it is possible to control the behaviour of the manager and agents by starting, pausing, resuming and stopping any one of them. The following list describes all available commands:

- **start** starts all workload agents that were already configured
- **startInstance <agent> <instance>** starts certain instance of the given workload agent
- **startAllInstances <agent>** starts all instances of an agent
- **stop** stops the workload generator (all its agents and instances)
- **stopInstance <agent> <instance>** stops certain instance of the given workload agent
- **stopAllInstances <agent>** stops all instances of an agent

- **pause** temporarily pauses all workload agents
- **pauseInstance** **<agent>** **<instance>** pauses certain instance of the given workload agent
- **pauseAllInstances** **<agent>** pauses all instances of an agent
- **resume** resumes all workload agents
- **resumeInstance** **<agent>** **<instance>** resumes certain instance of the given workload agent
- **resumeAllInstances** **<agent>** resumes all instances of the given agent type
- **kill** kills all workload agents that were already configured
- **killInstance** **<agent>** **<instance>** kills certain instance of the given workload agent type
- **killAllInstances** **<agent>** kills all instances of the given agent type
- **listAgents** returns the list of all agents with their statuses
- **quit** terminates all workload agents and quits the workload generator manager.

Command response format depends on the underlying command: it can be free text or JSON [14] / XML serialised object. Free text messages are also used by workload agents reporting their status, e.g., waiting for the Web Service operation to return.

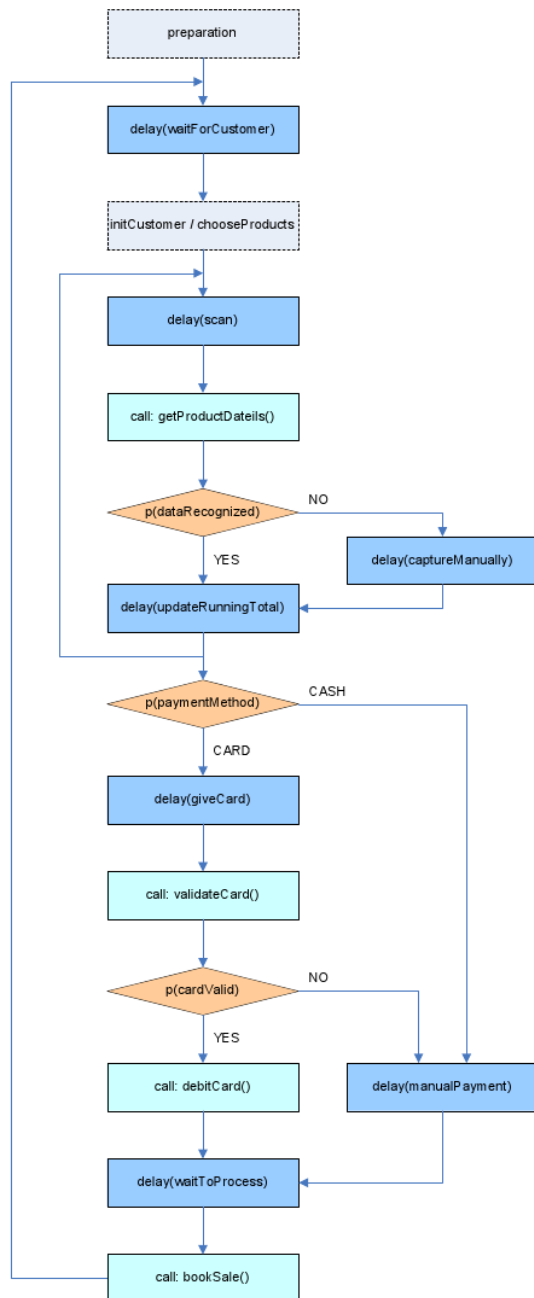
### 4.1.2 *The sales process*

Every workload agent is imitating the sales process depicted in Figure 18 starting with the selection of products that are used to update the running total of the purchase. Once all products are processed, the workflow simulates the payment and completes the flow with a call to book the sale and update the stock. Real-world-like workload is achieved with additional delays and various probabilities that guide the workflow. Below is a list of most important steps in the workflow:

1. **Preparation.** This step is omitted from the main workflow. Its only purpose is to retrieve available products from the InventoryService so that successive steps work on valid products. Only barcodes are stored at this stage.
2. **Init customer.** This step initialises the sale. Products are randomly chosen from the list of all available products. This step simulates the buyer's shopping cart.
3. **Get product details** service call is used to retrieve all the details of the current product. Although all the details are retrieved, only the price is used to update the running total.
4. **Is data recognised?** Based on pre-configured probability, it is possible that the product is not recognised. In this case, additional delay is used to simulate manual capturing of product's price.
5. **Payment method selection** controls the amount of credit card payments and cash payments.
6. **Credit card payment** calls the Payment service to validate the card. As we are not working with real data, we use another probability to decide whether to send correct PIN number or not. If the card is rejected, the manual payment is executed. Otherwise the Payment service calls the PaymentDebitService internally to debit the given credit card.
7. **Book Sale** is the last call in the workflow that sends the list of bought products and updates the stock.

Probabilities and delays are all part of the configuration that is sent to the workload manager. A normal (Gaussian) distribution is used to describe delays

appearing in the workflow, resulting in parameters for each delay, namely the mean value and the standard factor (deviation). Alternatively, Poisson distribution is also supported by workload generators (workload generators can also be configured via means of a configuration file or by XConsole client [15]). These parameters are described in detail in the following section.



**Figure 18: The sales process workflow. Normal service calls are displayed in light blue. Dark blue rectangles are used for delays. Decision symbols are depicted in orange colour with the name of the probability used.**

### 4.1.3 Configuration options

Configuration of workload generators is done via message bus. Message is formatted as JSON string consisting of two sections. The first section describes global configuration, reflecting common properties of all agents, while the second

section describes actual workload agents. The latter supports the configuration of an arbitrary number of workload agents in a single configuration string. An example of a JSON formatted configuration is shown in Table 2.

**Table 2: An example of the JSON formatted workload configuration.**

```
{
  // Global settings
  "id" : "test",
  "webClientType" : 1,
  "agents": [

    // Configuration of the first agent
    {
      "id" : 1,
      "probabilities" : {
        "dataRecognized" : 0.5,
        "paymentMethod" : 0.5,
        "cardValid" : 0.5
      },
      "numInstances" : 3,
      "url" :
      "http://172.16.118.209:8081/Store1/",
      "webClientType" : 1,
      "numberOfBoughtProducts" : 10,
      "randomSeed" : 32453,
      "timeDistribution" : 2,
      "delay" :
      {
        "waitForCustomer" : 5000,
        "scan" : 2000,
        "captureManually" : 5000,
        "updateRunningTotal" : 3000,
        "giveCard" : 4000,
        "manualPayment" : 5000,
        "waitToProcess" : 7000
      }
      "randomStandardFactor" :
      {
        "waitForCustomer" : 2000,
        "scan" : 1000,
        "captureManually" : 2000,
        "updateRunningTotal" : 1000,
        "giveCard" : 3000,
        "manualPayment" : 3000,
        "waitToProcess" : 5000
      }
    },

    // Configuration of the second agent
    {
      "id" : 2,
      // ...
    },

    // Additional agents
    // ...
  ]
}
```

```
}

```

Since the message bus channel is public all messages are sent to all workload managers connected to the channel. Each workload manager checks the ID of the configuration message and it executes the configuration, if the message ID corresponds to its own ID. Current implementation supports two types of clients, namely JAX-WS and Axis. Appropriate client is selected based on the *webClientType* property. Although this property is global, every agent can override it. For further information about the global section refer to table Table 3.

**Table 3: Description of global configuration properties.**

Field	Type	Description
id	String	the name of the workload manager
webClientType	Integer	type of the interface used (1 for JAX-WS, 2 for Axis)
Agents	JSONArray	array of agents configurations (see description below)

Agent configuration properties are described in tables Table 4, Table 5, and Table 6. The manager uses the *numInstances* property to initialize a given number of workload agents with the same configuration. To configure agents with different workload properties, specify separate agent descriptions.

**Table 4: Agent configuration properties.**

Field	Type	Description
id	String	id of the agent, usually numbered sequentially
probabilities	JSONArray	Workflow probabilities (refer to Table 5)
numInstances	Integer	Number of Instances of this agent to start
url	String	Url of the service, the agent is calling
webClientType	Integer	Client type (1 for JAX-WS, 2 for Axis)
numberOfBoughtProducts	Integer	Number of products to buy
randomSeed	Long	Random seed used by random number generator.
timeDistribution	Integer	Random distribution to use for delays.
delay	JSONArray	Base delays for workflow tasks (refer to Table 6)
randomStandardFactor	JSONArray	Standard factor for Gaussian distribution (Table 6)

**Table 5: Description of workflow probabilities controlling branches in the workflow.**

Field	Type	Description
dataRecognized	Double	Probability that data on an article is recognized: 1 – always recognized 0 – never recognized
paymentMethod	Double	Probability that customer will pay with credit card: 1 – always pay with credit card 0 – always pay with cash

cardValid	Double	Probability that card is valid: 1 – card is always valid 0 – card is never valid
-----------	--------	--

**Table 6: Description of workflow delay. Each delay is described in terms of the normal distribution, thus there are two arrays used (delay and randomStandardFactor).**

Field	Type	
waitForCustomer	Distribution	Time to wait for new customer
scan	Distribution	Time to wait for scanning of a product
captureManually	Distribution	Time needed to enter barcode manually
updateRunningTotal	Distribution	Time to update running total
giveCard	Distribution	Time needed for a customer to give the credit card
manualPayment	Distribution	Time needed for manual payment (cash)
waitToProcess	Distribution	Time needed to process the bill

## 4.2 Simulation Environment Implementation

Implementation of the simulation environment is split into several Java packages, see Table 7.

**Table 7: Main classes and packages constituting the Simulation Environment**

Class	Description
org.slasoi.adhoc.workload.WorkloadGenerator.java	Main program that subscribes to the message bus and delegates the work to the manager.
org.slasoi.adhoc.workload.AdhocJobManager.java	Class that manages all agents and handles commands received from the message bus (more specifically, from the main program that is listening to the message bus).
org.slasoi.adhoc.workload.agent.*	Abstract class for the sales process and concrete implementations based on JAX-WS and Axis.
org.cocome.tradingsystem.webservices.{jaxws,axis}	Classes generated by maven plugins for accessing services.
DIXI (eu.xtreemos.*)	The console command parser and code generator is based on the DIXI library.

### 4.2.1 WorkloadGenerator.java

This class contains the main entry-point and is responsible for:

- Creating unique GUID of the workload generator. All the agents running on this generator use GUID to distinct themselves from other workload generators.
- Starting communication via the message bus.
- Intercepting message bus messages depending on the message type. Commands are tunnelled into the command line parser, auto-generated by DIXI, configurations are parsed and passed to the generator and its agents.

- Stopping the communication via the message bus.

Access to the message bus is configured from a property file called *manager.properties*. Default values are shown in Table 8: Example of *manager.properties* configuration. The messaging engine is based usually on *XMPP* although it might be also based on *AMQP*, because the *AMQP* implementation of the messaging library has been provided in Y3. Since we are working with Pub/Sub type of messaging, we need to specify the *PubSub* engine in the configuration as well.

*Properties xmpp\_username* and *xmpp\_password* need to be changed for every workload program. Alternatively, the anonymous user is used in case of removing these two properties from the configuration file. It is important to note that the *xmpp\_channel* needs to be the same as the one that is used by the Adhoc GUI. Its name, however, can be arbitrary.

Properties under "XMPP – server" section depend on the *XMPP* server and should be adapted as well.

**Table 8: Example of manager.properties configuration.**

```
#Messaging engine
messaging=xmpp

#PubSub engine
pubsub=xmpp

# XMPP - user
xmpp_username=wg1
xmpp_password=wg1
xmpp_channel=testChannel-adhoc1

# XMPP - server
xmpp_host=192.168.0.23
xmpp_port=5222
xmpp_service=virtuoz
xmpp_resource=workloadgenerator
xmpp_pubsubservice=pubsub.virtuoz
xmpp_chatservice=conference
```

## 4.2.2 AdhocJobManager.java

This class is the connector between workload manager main program and agents. It consists of two kinds of methods:

- **Job list management** methods manage agents, create and initialise them, retrieve them, retrieve the number of idle jobs, etc.
- **Methods used to command workload generator, agents and instances** are used by message bus command parser. They are all annotated with Java Annotation

`@XOSDCONSOLE(consoleString = "_console_command_")`

This annotation is used by XtreamOS DIXI service processor that generates the code needed for emulating a console. It creates a file *eu.xtreemos.xconsole.command.XConAdhocJobManager* with methods needed for command parsing. In this manner, a parser for user-friendly command language is automatically created. This command language is currently used by communicating software components to control each other's behaviour, but is on the other hand suitable to be used by a human operator in console window as well.

## XOSDCONSOLE Example

Table 9 shows an example of a real function, annotated with the XOSDCONSOLE annotation. DIXI command parser and code generator parses existing source code and generates stub functions for all functions, annotated with the XOSDCONSOLE annotation. The name of the command may be changed with the `_console_command_` parameter, while command parameters are generated from the method signature found in the source file.

At the moment, DIXI works best with textual parameters although support for numeric parameters is also implemented.

**Table 9: Example of a function, annotated with the XOSDCONSOLE annotation.**

```
@XOSDCONSOLE(consoleString = "startInstance" )
public static void startInstance(String agent, String instance) {
    // implementation
}
```

### 4.2.3 Agent.java

This class implements the sales process workflow from Figure 18. It is an abstract class, extended by AgentJaxWs and AgentAxis that provide access to web services using JAX-WS and Axis, respectively.

### 4.2.4 Generating Java code from WSDL

Two Maven plug-ins were used for generation of Java code needed to access Web Services. These classes are then used by AgentJaxWs and AgentAxis classes in the `org.slasoi.adhoc.workload.agent` package. The use of the plug-ins is briefly described in the following sub-sections.

## JAX-WS

For the creation of JAX-WS code from a given WSDL the URL of the WSDL must be added to `pom.xml` file in the following section. Entries `packageName` and `wSDLUrls` contain the data about the Web Service.

**Table 10: JAX-WS pom.xml configuration.**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>jaxws-maven-plugin</artifactId>
      <configuration>
        <sourceDestDir>src/main/java</sourceDestDir>
        <packageName>org.cocome.tradingsystem.webservices.jaxws</packageName>
        <verbose>>true</verbose>
        <wSDLUrls>
          <wSDLUrl>http://virtuoz:8081/InventoryService?wsdl</wSDLUrl>
        </wSDLUrls>
      </configuration>
    </plugin>
  </plugins>
</build>
```

The code is generated using the following maven command:  
# mvn jaxws:import

*Verbose* parameter, although not required, can help in finding problems with the plug-in and/or WSDL configuration.

## Axis

Axis client classes are generated with a similar Maven plugin.

**Table 11: Axis pom.xml configuration.**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.axis</groupId>
      <artifactId>axis-wsdl2code-maven-plugin</artifactId>
      <version>1.4</version>
      <executions>
        <execution>
          <goals>
            <goal>wsdl2code</goal>
          </goals>
          <configuration>
            <packageName>org.slasoi.adhoc.webservices.axis</packageName>
            <wsdlFile>workload/src/main/axis/InventoryService.wsdl</wsdlFile>
            <databindingName>xmlbeans</databindingName>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

The code needed is then created by command:  
# mvn wsdl2code:wsdl2code

At the time of writing, the plug-in had some difficulties downloading WSDLs from the server. To this end, we needed to download WSDL files manually and use *wsdlFile* property in Maven configuration.

## 4.3 Graphical User Interface

### 4.3.1 Introduction

The Graphical User Interface for Open Reference Demonstrator is a Java-based web application with the frontend developed in [Adobe Flex](#) [16]. Its purpose is to demonstrate the features and usage of the framework.

The advantage of developing the demonstrator as a web application over Eclipse Rich Client Platform [19] (Ad-hoc Demonstrator from year 1) lies in a simplified way for the users to test and use the application – instead of having to install the application on a local computer, it is deployed on a remote public server and can be accessed using a web browser.

The Graphical User Interface consists of three main sections: SLA Management, Framework Overview and ORC Simulation and are organized in tabs that are always available to the user. A more detailed explanation about the UI with a

number of screenshots attached is available within the [Open Reference Demonstrator guide](#) [2].

The ORC Simulation section makes use of the Workload Generator – a Java application for simulating a real-world sales process, which is used to demonstrate the behaviour of the framework in runtime, see 4.1.

A detailed documentation on installation and usage of the Reference Demonstrator GUI can be found in the [Open Reference Demonstrator guide](#) [2] document.

### 4.3.2 Graphical User Interface Implementation

Frontend of the GUI is implemented in Adobe Flex, which is an open source web application development tool that deploys across all major browsers, desktops, and operating systems.

The application incorporates an event-driven programming architecture technique with the help of [Mate Framework](#) [17] and uses Model-View-Controller (MVC) architecture. This enables easier separation of the application logic into three separated and thus easier to manage parts.

Basic structure of the frontend application is divided into the following packages:

- Business
- Events
- Maps
- Model
- Views

**Business package** contains business logic of the application. It is responsible for processing framework responses, issuing commands to the Java backend application, which in turn communicates with the framework and storing data to the model.

**Events package** contains definitions of the events that get dispatched on various occasions. An example of an event is FrameworkEvent, which is dispatched in case of framework interaction.

**Maps package** connects events with event handlers. For example if the framework notifies that it finished negotiation process, then a corresponding event is triggered and with the help of an event map the right event handler in business logic is called. Here is an example of an event handler:

```
<EventHandlers type="{FrameworkEvent.NEGOTIATE}">
  <MethodInvoker generator="{FrameworkManager}"
method="negotiate" arguments="{[event.slat]}"/>
</EventHandlers>
```

**Model package** contains models where the actual data is stored and injected to the view.

**View package** contains implementation of view components that the user interacts with (panels, buttons, tables, etc.). View components are directly connected with the Model and get updated as soon as contents of the model changes.

The frontend communicates with the Java Web application using BlazeDS [18]. BlazeDS is a server-based Java remoting and web messaging technology that enables developers to easily connect to backend data and push data in real-time to Adobe Flex applications for more responsive web application experience.

Backend Java Web application, which resides on a web servlet container (e.g. Tomcat [20]) encapsulates several functionalities:

- Using the Message Bus
- Communicating with the Framework through a Web Services Interface
- Managing the Workload Generator

Implementation of the simulation environment is split into several Java packages (Table 12).

**Table 12: Reference Demonstrator Java Web Application packages**

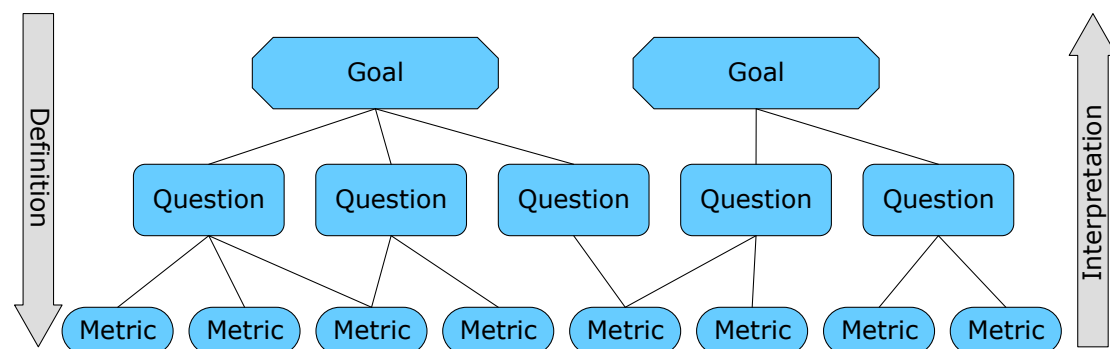
Package	Description
org.slasoi.businessmanager	Package containing generated web services stub classes for communication with the Business Manager.
org.slasoi.gslam	Package containing generated web services stub classes for communication with the GSLAM (Generic Service Level Agreement Manager) and related components (e.g. Syntax Converter).
org.slasoi.ord	Package containing classes that get imported into frontend application. These classes define an interface between frontend and backend application.
org.slasoi.ord.framework	Package containing classes that encapsulate framework interaction.
org.slasoi.ord.messaging	Package containing classes for message bus handling.
org.slasoi.ord.workload	Package containing classes that encapsulate functionality for managing the Workload Generator.

## 5 Evaluation

The evaluation plan presented in this section is an extension of the version used for the evaluation of the Ad-hoc demonstrator. The results of the Ad-hoc demonstrator's evaluation at the end of year one were the starting point for the development of the reference demonstrator. Hence, we re-evaluated some of the Y1 questions. Using a common set of questions for both evaluations of the reference demonstrator allows for comparing the evaluations and thereby supports the demonstration of the improvements implemented in the reference demonstrator. The plan is based on the Goal/Question/Metric (GQM) approach proposed by Basili, Caldiera, and Rombach [24], which is summarized first. Then we present the concrete instantiation of a GQM plan used to evaluate the Reference demonstrator.

### 5.1 The Goal/Question/Metric Approach

The Goal/Question/Metric (GQM) approach is a process model for measurements targeting a particular set of issues (goals) and a set of rules for the interpretation of the measured data. In order to be meaningful, measurements must be goal-oriented and, thus, are defined in a top-down fashion. Basili, Caldiera, and Rombach argue that measurements that are not performed in a goal-oriented way are likely to be inefficient. The absence of concrete goals carries the risk of collecting large amounts of unnecessary data. Large amounts of data and missing goals may complicate the interpretation of measurements.



**Figure 19: Relations between goals, questions, and metrics**

The GQM method starts with the explicit definition of a measurement goal. Several questions serve to refine the goal and to identify its major components that need to be answered by the measurements. Questions are further refined by metrics. Figure 19 depicts the relation between goals, questions, and metrics. When the measurements for each metric have been taken, the resulting data is interpreted bottom up. Each metric is directed towards specific questions. The collected data allows for answering the questions with respect to the goal. This evaluation allows for deciding whether the goal has been attained or not. Figure 19 further indicates that the same metric can answer different questions.

In GQM, **goals** are located on a conceptual level. They strongly depend on the context in which measurements take place. The context subsumes the objects, the reasons, the points of view, and the environment of the measurements, as well as the considered models of quality. Possible objects of measurements are products (such as artefacts, deliverables or other documents), processes (such as software related activities) or resources (such as personnel, hardware or

software). To correctly embed a goal into a given context, the GQM method requires the explicit definition of the goal's issue, object or process, viewpoint, and purpose. **Questions** determine the assessment of a specific goal. They characterise the object of measurement (product, process, resource) with respect to selected quality attributes from the selected viewpoint. On a quantitative level, **metrics** associate a set of data to each question. The data answer the questions in a quantitative way. In GQM, there exists a distinction between objective and subjective metrics. While objective metrics depend only on the object under measurement (e.g., lines of code), subjective metrics depend on the viewpoint from which the measurements are taken (e.g., readability of a text). When selecting metrics, various factors have to be considered. Basili et al. summarise the most important ones as follows:

- **Amount and quality of existing data:** To minimise the effort during data collection, the use of existing data sources should be maximised.
- **Maturity of the objects of measurement:** Objective metrics are preferable for more mature measurement objects, while subjective evaluations are better suited for informal or unstable objects.
- **Learning process:** GQM plans need iterative refinement and adaptation. The defined metrics have to evaluate not only the object of measurement but also the reliability of the model in use.

## 5.2 GQM-based Evaluation

The evaluation plan for the Reference demonstrator is based on the aforementioned GQM approach. The goals represent requirements and features from the business perspective as well as requirements and features of the framework from a developer and service provider perspective. The evaluation has its main focus on the usability of the framework. However, it also considers the extendibility of the framework as well as the use of the ORC as a reference application for further developments in the context of SLA management. The goals are:

1. Support of the service lifecycle by the SLA framework in the open reference use case
2. Usability of the framework from a service provider's point of view
3. Usability of the framework from a customers' point of view
4. Extensibility of the framework to support new requirements caused by an evolution of the ORC
5. Extensibility of the ORC itself to evaluate extensions of the SLA@SOI framework

The goals are broken down into several questions. For each question, a number of metrics are defined that allow for answering the respective question.

The following table summarizes the goals, questions and metrics. The results of the evaluation are added to each metric.

**Table 13: Goals, Questions and Metrics.**

G1	Support of the service lifecycle by the SLA framework in the open reference use case
	In deliverable D.A1a a number of base scenarios that should be supported by the SLA framework are defined. In the open reference use case some of these scenarios are instantiated. This goal focuses on evaluating, which of the base scenarios are supported.
Q1.1	Does the SLA framework support the software and service provider in offering new or modified service

	offers?
M1.1.1	<p>Is there an interface for service providers to deploy/install software on the virtual images or update software on these images?</p> <p><b>Results:</b> Yes: Virtual machine images are provided and can be accessed using SSH and SCP.</p>
M1.1.2	<p>Level of automation of the steps required for installing the ORC.</p> <p><b>Results:</b> 100%: The installation of the ORC can be performed in a fully automated way using the installation script for Ubuntu-based systems. However, it can also be installed manually from source code. Both installation options are described in the <a href="#">ORC deployment guide</a> [1].</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
M1.1.3	<p>Level of automation of the steps required for updating the ORC.</p> <p><b>Results:</b> 100%: The update can be performed similar to the installation.</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
Q1.2	Does the SLA framework support the service provider in provisioning the ORC for a new customer?
M1.2.1	<p>Is there an interface for service providers to create and start new instances of the ORC image?</p> <p><b>Results:</b> Yes, a new instance of the ORC image can be created and started using the interface <code>IServiceProvisioning</code> which allows an infrastructural service (such as a VM hosting an ORC image) to be created and started.</p>
M1.2.2	<p>Is it possible to run several independent instances of the ORC?</p> <p><b>Results:</b> Yes: The ORC configuration service enables the Service Provider to reconfigure the mapping of names and IP addresses at runtime.</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
M1.2.3	<p>Is there an interface for service providers to see which and how many instances of the ORC are running?</p> <p><b>Results:</b> Yes. The ISM Proxy interface includes a method <code>query()</code> which allows the details of provisioned VMs to be retrieved.</p>
M1.2.4	To what extent is the creation and start of new instances of

	<p>ORC images automated?</p> <p><b>Results:</b> 100%: The creation and start of new instances of ORC images is completely automated.</p>
Q1.3	<p>Does the framework support the customer and the service provider in offering, selecting and negotiating SLAs?</p>
M1.3.1	<p>Is there an interface for customers to browse and select offered services and the associated SLA offers?</p> <p><b>Results:</b> Yes: It is integrated into the Business Web Tool, which is described in deliverable DA2.a</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
M1.3.2	<p>Is there an interface for service providers to adapt SLA offers and SLA templates?</p> <p><b>Results:</b> A2: Yes: There are three ways of negotiation products for providers: automatic, semiautomatic, and manual negotiation. In case of manual negotiation provider can modify manually the SLA offer to send to the customer in runtime. A5: Yes, the SLA-Editor developed in Y3 can be used for creating and editing SLA templates as well as SLAs. It can be extended to hook into on-going negotiations to manipulate values during negotiations.</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
M1.3.3	<p>To what extent is the selection and negotiation of SLAs automated from the customer's perspective?</p> <p><b>Results:</b> 100%: The user only needs to select an appropriate SLAT. He can vary some quality parameters within predefined ranges if necessary. He can also access the framework in a fully automated manner using web service interfaces, so that no manual interaction is required.</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
M1.3.4	<p>To what extent is the negotiation of SLAs automated from the service provider's perspective?</p> <p><b>Results:</b> 100%: Fully automated – no human intervention required. Details are described in D.A5a.</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
Q1.4	<p>Does the SLA framework support the service and infrastructure provider in operating the ORC and detecting SLA violations?</p>
M1.4.1	<p>Are there events for SLA violations on the infrastructure level?</p> <p><b>Results:</b></p>

	<p>Yes: The low level monitoring system publishes infrastructure SLA violation events to the XMPP event bus.</p>
M1.4.2	<p>Are there events that inform the infrastructure provider about a forthcoming SLA violation?</p> <p><b>Results:</b>                  Yes: The low level monitoring system publishes infrastructure SLA violation warning events to the XMPP event bus. A warning event is emitted when a possibility of SLA violation is detected. Warnings are possible and reasonable only for some numerical QoS terms (e.g. Service Availability, MTTF, MTRR, MTTV). They happen when the QoS term value exceeds the warning threshold (from above or from below, depends on the QoS term) and is nearing the violation threshold.</p>
M1.4.3	<p>Are there warning events on the infrastructure level that are used to enhance the SLA monitoring on the software level?</p> <p><b>Results:</b>                  Yes: The infrastructure component raises events about SLA violation, which can be consumed by the Software PAC component.</p>
M1.4.4	<p>Are there events for SLA violations on the software level (services of the ORC)?</p> <p><b>Results:</b>                  Yes: The instrumentation of the ORC emits events, which are analysed by the monitoring engines. If a violation is detected a SLA violation event is emitted. Described in D.A3a.</p> <p><b>→ Improvement compared to Y1 evaluation</b></p>
M1.4.5	<p>Are there events for SLA violations of business SLAs?</p> <p><b>Results:</b>                  Yes: The SLA allows specifying different kind of reports, which should be used by the framework to report SLA violations or other information.</p> <p><b>→ Improvement compared to Y1 evaluation</b></p>
M1.4.6	<p>Do SLA violation events provide sufficient information to identify the (root) cause of the violation?</p> <p><b>Results:</b>                  Yes: If a violation of an SLA is detected and is defined to be reported, the framework sends a detailed report to the customer or allows the customer to get this report.</p> <p><b>→ Improvement compared to Y1 evaluation</b></p>
M1.4.7	<p>Are there events for SLA violations from the customer's side?</p> <p><b>Results:</b>                  Yes: The Monitoring engines can track all service executions and calculate the throughput with respect to call frequency.</p>

	<p>If the calculated value is higher than the value in the SLA, an SLA violation is emitted</p>
M1.4.8	<p>Does the SLA framework provide an overview on the currently active SLAs?</p> <p><b>Results:</b> Yes: The GSLAM component provides a method to receive a list of all SLAs. An additional tool provides a user interface to receive and visualize this information. More details in D.A5a.</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
Q1.5	<p>Does the SLA framework support the infrastructure provider in adapting the resources to fulfil the guaranteed SLA if required?</p>
M1.5.1	<p>Is there an interface that allows changing the resources associated to the virtual machine running the ORC?</p> <p><b>Results:</b> Yes: The entire infrastructure stack has been implemented to support dynamic re-provisioning. This includes the adjustment of, for example, CPU and memory allocations. Note that support for adjustment depends on the features of the underlying provisioning system. The reference implementation uses a modified version of Apache Tashi.</p>
M1.5.2	<p>To what extent is the adjustment of resources for virtual machines automated?</p> <p><b>Results:</b> The Infrastructure PAC supports the automatic and dynamic adjustment of resources to meet SLAs. Furthermore the Infrastructure Service Manager and Provisioning System support automatic, dynamic adjustment of resources to meet internal infrastructure provisioning policies. E.g. server consolidation versus load balancing.</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
M1.5.3	<p>Is there an interface that allows the infrastructure provider to monitor the resource utilization of virtual machines running the ORC?</p> <p><b>Results:</b> Yes: All historical infrastructure monitoring data is accessible from the historical monitoring database. This can be accessed through either an LLMS interface or an XMPP interface.</p> <p>The operation <code>GetMetricHistory</code> returns metric value history for given metric and time interval for the specified VM/infrastructure service. A client sends a <code>GetMetricHistoryRequest</code> message and the LLMS returns <code>GetMetricHistoryResponse</code> message.</p> <p>→ <b>Improvement compared to Y1 evaluation</b></p>
G2	<p>Usability of the framework from a service provider's point of view</p>
	<p>The service provider is the most important stakeholder of the SLA@SOI</p>

	<p>framework. Therefore the usability of the framework from a service provider's point of view is an essential factor determining the overall framework usability.</p>
<b>Q2.1</b>	<b>What support is provided for installing the framework?</b>
M2.1.1	<p>Is there a documentation available describing the building of the framework from source code?</p> <p><b>Results:</b> Yes: The building of the framework is described on a dedicated page in the SLA@SOI Source Forge project.</p>
M2.1.2	<p>Is the building of the framework tool-supported?</p> <p><b>Results:</b> Yes: The building of the framework is based on Maven and fully automated.</p>
M2.1.3	<p>Is there a documentation available describing the installation of the framework?</p> <p><b>Results:</b> Yes: There is a dedicated page in the SLA@SOI Source Forge project.</p>
M2.1.4	<p>Is there an automated installation script available?</p> <p><b>Results:</b>  No: Several additional tools, e.g. an SQL database server are required. Thus a fully automated installation is not possible.</p>
M2.1.5	<p>How many additional software packages are required to run the framework?</p> <p><b>Results:</b>  Following additional tools are required to run the complete framework</p> <ul style="list-style-type: none"> <li>• SLA@SOI Infrastructure Monitoring Agent</li> <li>• SLA@SOI Prediction Server</li> <li>• SLA@SOI Business Web Tool</li> <li>• Everest</li> <li>• Pax-Runner</li> <li>• XMPP Server</li> <li>• SQL-database server</li> <li>• Java</li> </ul>
M2.1.6	<p>List of use case specific models that need to be specified in advance of using the framework.</p> <p><b>Results:</b></p> <ul style="list-style-type: none"> <li>• SLATs</li> <li>• Prediction Model</li> <li>• Service Construction Model</li> </ul>
<b>Q2.2</b>	<b>What support is provided for the initial configuration/adaptation of the framework?</b>
M2.2.1	<p>Is there a documentation available describing the configuration of the framework?</p> <p><b>Results:</b></p>

	Yes: There is the SLA@SOI Adoption Guide and the SLA@SOI tutorial
M2.2.2	Which tools and editors are provided by the framework supporting the configuration of the framework? <b>Results:</b> <ul style="list-style-type: none"> <li>• SCM-Editor, see D.A3a</li> <li>• Prediction model editor (PCM), see D.A6a</li> <li>• Business Web Tool, see D.A2a</li> <li>• SLA@SOI Studio (see <a href="#">SLA@SOI Studio Guide</a> [4])</li> <li>• SLA Editor, see D.A5a</li> </ul>
M2.2.3	Is there an example available documenting and demonstrating the adaptation of the framework? <b>Results:</b> Yes: In addition to the adoption guide a <a href="#">SLA@SOI tutorial</a> [3] is available which describes the configuration of the framework based on the ORC scenario.
M2.2.4	Which required models are accompanied by an editor supporting the creation and adaption of a model instance? <b>Results:</b> <ul style="list-style-type: none"> <li>• Prediction Model</li> <li>• Service Construction Model</li> <li>• SLA Model</li> </ul>
Q2.3	What support is provided to control and adapt the framework at runtime (e.g. change/update SLATs, adjustment/negotiation rules)?
M2.3.1	Is there an interface to add a new SLAT programmatically to the SLAT registry? <b>Results:</b> Yes, templates can be added to the template registry via Java- and SOAP-interfaces.
M2.3.2	Is there a UI provided to create new SLATs manually? <b>Results:</b> Yes, the SLA Editor can be used to create SLA templates by hand with a graphical user interface.
M2.3.3	Is there an interface provided to update SLATs programmatically? <b>Results:</b> Updating SLATs can be done using the SLA Editor on an existing SLA template.
M2.3.4	Is there a UI provided to update SLATs manually? <b>Results:</b> Yes, the SLA Editor.
M2.3.5	Is there an editor for SLATs available? <b>Results:</b> Yes, SLA Editor. See <a href="#">SLA@SOI Studio</a> [4] documentation for more details.
M2.3.6	Is there an interface to manipulate the negotiation process (e.g. changing thresholds)? <b>Results:</b> Yes, the negotiation process can be adjusted and customized using the Y3 customization techniques in the protocol-

	engine.
M2.3.7	<p>Is there an editor provided to adapt the rules/parameters used for SLA negotiation?</p> <p><b>Results:</b> No, such an editor can be included in future as a part of the SLA@SOI Studio.</p>
<b>Q2.4</b>	<b>What is the effort required to configure the framework?</b>
M2.4.1	<p>How much time was required to create the service construction model for the ORC?</p> <p><b>Results:</b> Using the provided editor the required time was about 30 minutes.</p>
M2.4.2	<p>How much time was required to create the prediction model for the ORC?</p> <p><b>Results:</b> The model creation itself required 2h, the calibration of the model and determination of resource demands requires some load tests and took 1 day.</p>
M2.4.6	<p>How much time was required to create SLATS for the ORC?</p> <p><b>Results:</b> As an editor was not available and the language is very expressive and thus complex, specifying the SLATs took some days and required much interaction with the designer of the SLAT language.</p>
M2.4.6	<p>How much time was required to build and install the framework, except the definition of models?</p> <p><b>Results:</b> It took approximately one hour to build and install the framework.</p>
<b>G3</b>	<b>Usability of the framework from a customers' point of view</b>
	<p>The customer satisfaction is an important aspect for the success of a product or service. As customers directly use the SLA framework, its usability heavily influences the satisfaction of the service provider's customers.</p>
<b>Q3.1</b>	<b>What is the complexity of negotiating a new SLA?</b>
M3.1.1	<p>Is there a UI provided to adapt and negotiate SLA offers and SLAs?</p> <p><b>Results:</b> The Business Web Tool is a web based UI.</p>
M3.1.2	<p>Is there a documentation available describing the interaction with the framework from a customer perspective?</p> <p><b>Results:</b> The Business Web Tool is documented in D.A2a.</p>
M3.1.3	<p>Minimal number of interaction steps to negotiate an SLA.</p> <p><b>Results:</b> 5:</p> <ul style="list-style-type: none"> <li>• Get Products</li> <li>• Get Templates</li> </ul>

	<ul style="list-style-type: none"> <li>• Authenticate</li> <li>• Negotiate</li> <li>• Create Agreement</li> </ul>
Q3.2	To which extent does the framework provide support for the interaction with the customer?
M3.2.1	Is a graphical UI provided to interact with the framework? <b>Results:</b> The Business Web Tool is the interaction point of framework and customer. See D.A2a.
M3.2.2	List of technical requirements for the customer to interact with the framework? <b>Results:</b> The customer can interact using the Business Web Tool or using web service interfaces provided by the framework.
M3.2.3	Is there a public API available to interact with the framework programmatically? <b>Results:</b> Yes, web service interfaces.
G4	Extendibility of the framework to support new requirements caused by an evolution of the ORC
	Software systems often evolve over time (e.g. new functionality is added, a new technology is introduced). The framework should support such an evolution of the system. In this goal we evaluate the effort required to adapt the framework to support the evolving system. We define several evolution scenarios as questions.
Q4.1	What is the effort required to support a new quality term (e.g. confidentiality of data or security) that should be supported?
M4.1.1	List of affected components. <b>Results:</b> Predicting new quality attributes requires an extension of the software evaluation (customizing prediction model). Additionally, the SSLAM component (Software SLAM) might need some extensions.
M4.1.2	List of affected models. <b>Results:</b> Depending on the prediction technique the prediction model needs to be extended. The new attribute must be integrated into the SLATs.
M4.1.3	Estimated effort. <b>Results:</b> The effort for implementing a new prediction technique depends on the quality attribute and cannot be estimated.
Q4.2	What is the effort required to support a new service added to the ORC?
M4.2.1	List of affected components. <b>Results:</b> Possibly the SSLAM component needs to be extended.

M4.2.2	List of affected models. <b>Results:</b> SLATs, SCM, and Prediction Model need to be updated with the new service.
M4.2.4	Estimated effort. <b>Results:</b> As only small changes of the implementation are expected, we estimate the required effort to be 1 day.
Q4.3	What is the effort required to adapt the framework if some internals of the ORC implementation are changed?
M4.3.1	List of affected components. <b>Results:</b> No component is affected.
M4.3.2	List of affected models. <b>Results:</b> Eventually, the prediction model needs to be updated.
M4.3.4	Estimated effort. <b>Results:</b> Less than 1 day.
Q4.4	What's the effort required to adopt the framework if new external services are required
M4.4.1	List of affected components. <b>Results:</b> SSLAM
M4.4.2	List of affected models. <b>Results:</b> SCM
M4.4.4	Estimated effort. <b>Results:</b> Depending on the negotiation algorithm 1 day up to 2 weeks.
G5	Extendibility of the ORC itself with the aim to evaluate extensions of the SLA@SOI framework
	In addition to the demonstration of implemented features of the SLA@SOI framework, the ORC should also allow to be used as reference application to evaluate new research results in the context of SLA management. Within the following goals we envision several potential research results, that could be demonstrated using the ORC.
Q5.1	What is the required effort to demonstrate a new virtualization technique on the infrastructure level?
M5.1.1	List of required changes. <b>Results:</b> Two techniques are available in the Tashi version that framework version v2 is using (KVM and Xen). In order to support a new virtualization technique a new Tashi driver has to be implemented for a new virtualization technique. Another possibility is to provide a new IaaS package instead of Tashi. If IaaS package supports OCCI, this can be done

	quickly.
M5.1.2	Estimated effort. <b>Results:</b> One hour if IaaS supports OCCI (most of IaaS packages already supports OCCI – Open Nebula, OpenStack...).
Q5.2	What is the required effort to demonstrate the use of more detailed internal software monitoring data?
M5.2.1	List of required changes. <b>Results:</b> The monitoring needs to be integrated into the ORC's implementation.
M5.2.2	Estimated effort. <b>Results:</b> Using standardized monitoring extensions for Tomcat requires low effort. Extending the code of the ORC might result in more effort, depending on the used technique.
Q5.3	What is the required effort to demonstrate a Platform as a Service (PaaS) scenario?
M5.3.1	List of required changes. <b>Results:</b> The ORC has to be re-implemented.
M5.3.2	Estimated effort. <b>Results:</b> The effort will be very high, as everything has to be implemented anew, using the new programming model and API, offered by the PaaS solution.
Q5.4	What is the required effort to demonstrate new and more complex adjustment scenarios?
M5.4.1	List of required changes. <b>Results:</b> The SLA@SOI Dynamic Orchestration Engine (DOE) needs to be integrated. Its installation is documented in the ORC installation guide: <a href="https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/ord/doc/ORC_Deployment_Guide.pdf">https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/ord/doc/ORC_Deployment_Guide.pdf</a> [1]
M5.4.2	Estimated effort. <b>Results:</b> Less than 1 day.
Q5.5	What is the required effort to demonstrate new and more complex negotiation scenarios, which include other external service providers with additional SLA management frameworks?
M5.5.1	List of required changes. <b>Results:</b> The ORC already includes the bank services, which can be used as external services. Thus no changes are required.

M5.5.2	Estimated effort.
	<p><b>Results:</b> No effort, as no changes are needed.</p>
Q5.6	<p>What is the required effort to demonstrate new quality features like security or data confidentiality?</p>
M5.6.1	List of required changes.
	<p><b>Results:</b> The ORC needs to be extended with code that allows for varying this attributes. Currently security aspects are not covered within the ORC.</p>
M5.6.2	Estimated effort.
	<p><b>Results:</b> The required effort cannot be estimated, as it highly depends on the quality attribute and the required extensions.</p>

## 6 Conclusions

### 6.1 Contributions and Achievements

This document presented the Open Reference Demonstrator, the final demonstrator of the SLA@SOI framework including its evaluation using a predefined evaluation plan based on the Goal Question Metric approach. Open Reference Demonstrator has been developed on the base of the Ad-hoc demonstrator from the first year of the project. The UI component has been upgraded to a web site application and it has been extended in order to comprise most of the SLA@SOI framework features that have been developed since Ad-hoc framework version from Y1, especially the separation of concerns related to SLAs (SLA managers) and services (service managers). The user simulation component remained basically the same as it was in the first year, but some new configuration options have been added.

Open Reference Demonstrator has been made public and open source and an extensive documentation is available on Source Forge. [Demonstrator guide](#) [2] is available for those who would like to install the SLA@SOI framework together with other demonstrator components and start testing the framework features. Those who would like to adopt the SLA@SOI framework for their use case should refer to the [SLA@SOI tutorial](#) [3] based on the ORC. A quick customization of the framework SLAMs is possible using SLA@SOI studio and the modified framework components can be tested using Open Reference Demonstrator GUI.

The evaluation of the framework in the context of the ORC scenario is based on Goal-Question-Metric approach and it is an extension of the Y1 evaluation. The evaluation plan consists of overall 5 different goals, which are focused on the implementation of SLA management features, the usability of the framework, and extendibility of ORC and framework. Overall, we used more than 80 metrics in the evaluations. In comparison to the Y1 evaluation the results have been improved in more than 65 % of all metrics already used in the Y1 evaluation.

### 6.2 Lessons learned

Development of the SLA@SOI framework demonstrator and in particular the integration of all involved components has been a strong experience which provided the following lessons learned.

**Lesson 1: The integration of different SLA management layers (business, software, and infrastructure) is complex, but possible.**

Integrated SLA management is a highly complex, but following the tutorials on Source Forge and using different SLA@SOI tools developed in the last year of the project (Studio, SLA Editor) should significantly simplify the process of such integration.

**Lesson 2: Tools that help integrating a complex framework like SLA@SOI framework are essential.**

Editing and customizing SLATs for a specific use case can be quite a complex task, the same is true for a applying prediction and service construction models. A specific emphasis must be put on developing tools that makes these processes easier.

**Lesson 3: Demonstrators should be web-based, easy to install and should allow further customizations, testing and conducting experiments.**

The Ad-hoc demonstrator from the first year of the project was developed as a desktop application, which required some additional work for those who wanted to test and customize the framework behaviour. Therefore, the demonstrator should be web application, which does not require any installation steps. Furthermore, the customization of the framework should be simple enough to quickly apply the changes and to be able to immediately test a modified framework from a demonstrator GUI application.

### **6.3 Outlook**

Following the Open Reference Demonstrator guide and other documentation available on Source Forge, watching the screencasts that were recorded and are available on the SLA@SOI YouTube channel, and finally using the live demo that is available on a public address should suffice as a starting point for everyone interested in the SLA management and would like to adopt some if not all of the SLA@SOI framework components.

A customization of the framework is also possible using the SLA@SOI studio, which enables modifying various framework components and monitoring the changes in the framework behaviour via the Open Reference Demonstrator GUI.

Live demonstrator will stay on a public address and will be available for various presentations after the project ends.

## 7 References

- [1] ORC deployment guide on SourceForge. URL: [https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/ord/doc/ORC\\_Deployment\\_Guide.pdf](https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/ord/doc/ORC_Deployment_Guide.pdf)
- [2] Open Reference Demonstrator guide on SourceForge. URL: <https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/ord/doc/ReferenceDemonstrator-guide.doc>
- [3] SLA@SOI tutorial on SourceForge. URL: <https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/doc/SLA@SOI-tutorial.doc>
- [4] Studio user guide on SourceForge. URL: [https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/doc/Studio-User\\_Guide.doc](https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/doc/Studio-User_Guide.doc)
- [5] Reference Architecture guide on SourceForge. URL: [https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/doc/SLA@SOI-Reference\\_Architecture.pdf](https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/doc/SLA@SOI-Reference_Architecture.pdf)
- [6] COCOME, available at <http://www.cocome.org>
- [7] Adoption guide on SourceForge. URL: [https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/doc/SLA@SOI-Adoption\\_Guide.doc](https://sla-at-soi.svn.sourceforge.net/svnroot/sla-at-soi/platform/trunk/doc/SLA@SOI-Adoption_Guide.doc)
- [8] SLA@SOI Deliverable D.A1a Framework Architecture
- [9] WS-BPEL, available at [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel)
- [10] ActiveBPEL, available at <http://www.activevos.com/community-open-source.php>
- [11] SLA@SOI Deliverable D.A4a SLA-aware Infrastructure Management
- [12] JAX-WS, available at <http://en.wikipedia.org/wiki/JAX-WS>
- [13] Apache Axis, available at <http://ws.apache.org/axis/>
- [14] JSON, available at <http://www.json.org/>
- [15] XConsole, available at <http://xtreemos.org/publications/plonearticlemultipage.2008-06-26.0232965573/project-deliverables/d3-3-3-4final.pdf>
- [16] Adobe Flex, available at <http://www.adobe.com/products/flex/>
- [17] Mate framework, available at <http://mate.asfusion.com/>
- [18] BlazeDS, available at <http://opensource.adobe.com/wiki/display/blazeds/BlazeDS>
- [19] Eclipse Rich Client Platform, available at [http://wiki.eclipse.org/index.php/Rich\\_Client\\_Platform](http://wiki.eclipse.org/index.php/Rich_Client_Platform)
- [20] <http://tomcat.apache.org/>
- [21] SLA@SOI Deliverable D.A2a Business SLA Management
- [22] SLA@SOI Deliverable D.A5a SLA Foundations and Management
- [23] Log4j, available at <http://logging.apache.org/log4j/1.2/index.html>
- [24] V.R. Basili, G. Caldiera, and H.D. Rombach. The Goal Question Metric Approach. Encyclopedia of Software Engineering, 1:528-532, 1994

## 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 asterisk \*.

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

Service Level Objective	availability. Entities can dynamically establish and manage agreements via Web service interfaces. 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
CRM	Customer Relationship Management
EMF	Eclipse Modelling Framework
ERP	Enterprise Resource Planning
IE	Interaction Event
FCR	Finite capacity regions
ISLAM	Infrastructure SLA Manager
ISM	Infrastructure Service Manager
IoC	Inversion of Control
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LLMS	Low Level Monitoring System
LQN	Layered Queuing 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	Queuing Petri Nets
PAC	Provisioning and Adjustment Component
POC	Planning and Optimization Component
POJO	Plain Old Java Objects
SaaS	Software as a Service
SE	Service Evaluation
SLA	Service Level Agreement
SLAM	SLA Manager
SLAT	Service Level Agreement Template
SM	Service Manager
SME	Small and Medium-sized Enterprise
SOA	Service Oriented Architecture
SW-SLAM	Software SLA Manager
SW-SM	Software Service Manager
TCO	Total Cost of Ownership
TOGAF	The Open Group Architecture Framework
VM	Virtual Machine

## ***Appendix C: Open Reference Demonstrator Guide***

The Open Reference Demonstrator guide is served as separated appendix to this document. The most current version can be retrieved from the SourceForge project [2].

## ***Appendix D: ORC Deployment Guide***

The ORC deployment guide is served as separated appendix to this document. The most current version can be retrieved from the SourceForge project [1].

## ***Appendix E: Tutorial***

The SLA@SOI tutorial is served as separated appendix to this document. The most current version can be retrieved from the SourceForge project [3].