Project no. FP7- 216556

Instrument: Integrated Project (IP)

Objective ICT-2007.1.2: Service and Software Architectures, Infrastructures and Engineering

Deliverable D.B1a
Scientific Evaluation Report

Keywords:
Web Site, Service Level Agreement, Service-Oriented Infrastructure

Due date of deliverable: 31st July 2010
Actual submission to EC date: 17th September 2010

Start date of project: 1st June 2008
Duration: 38 months

Lead contractor for this deliverable: Engineering Ingegneria Informatica Spa
Revision: V.2.0 (17th September 2010)

<table>
<thead>
<tr>
<th>Dissemination level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU Public</td>
</tr>
</tbody>
</table>

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)
### Document Status

<table>
<thead>
<tr>
<th>Deliverable Lead</th>
<th>ENG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewer 1</td>
<td>Francesco Torelli</td>
</tr>
<tr>
<td>Reviewer 2</td>
<td>Joe Butler</td>
</tr>
<tr>
<td>PMT Reviewer</td>
<td>Robbert Zeeuw</td>
</tr>
<tr>
<td>Complete version submitted to reviewers</td>
<td>August 11, 2010</td>
</tr>
<tr>
<td>Comments of reviewer 1 received</td>
<td>September 14, 2010</td>
</tr>
<tr>
<td>Comments of reviewer 2 received</td>
<td>September 2, 2010</td>
</tr>
<tr>
<td>Revised deliverable submitted to PMT</td>
<td>September 16, 2010</td>
</tr>
<tr>
<td>PMT Approval</td>
<td>September 17, 2010</td>
</tr>
</tbody>
</table>

### Contributors

<table>
<thead>
<tr>
<th>Partner</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENG</td>
<td>Francesco Torelli, Paolo Zampognaro, Keven Kearney</td>
</tr>
<tr>
<td>TA</td>
<td>Mark Evenson, Alfred Tellian, Andreas Woratschek</td>
</tr>
<tr>
<td>XLAB</td>
<td>Gregor Berginc, Miha Stopar, Gregor Pipan, Miha Vuk</td>
</tr>
<tr>
<td>SAP</td>
<td>Tariq Ellahi, Wolfgang Theilmann</td>
</tr>
<tr>
<td>Intel</td>
<td>John Kennedy, Michael Nolan, Joe Butler, Victor Bayon</td>
</tr>
<tr>
<td>TID</td>
<td>Juan Lambea Rueda, Carlos Bueno Royo, Alfonso Castro, S. G. Gomez, Beatriz Fuentes,</td>
</tr>
<tr>
<td>UDO</td>
<td>Philipp Wieder, Constantinos Kotsokalis</td>
</tr>
<tr>
<td>FZI</td>
<td>Christof Momm, Franz Brosh</td>
</tr>
<tr>
<td>GPI</td>
<td>Giampaolo Armellin</td>
</tr>
<tr>
<td>City</td>
<td>Davide Lorenzoli, Howard Foster</td>
</tr>
<tr>
<td>PMI</td>
<td>Luciano Baresi</td>
</tr>
</tbody>
</table>

### Notices

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

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

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

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>22nd Jun 10</td>
<td>Torelli</td>
<td>Changed template</td>
</tr>
<tr>
<td>1.2</td>
<td>25th Jun 10</td>
<td>Gomez</td>
<td>First TID contribution</td>
</tr>
<tr>
<td>1.3</td>
<td>26 Jun 10</td>
<td>Ellahi</td>
<td>Added comments</td>
</tr>
<tr>
<td>1.4</td>
<td>27 Jun 10</td>
<td>Fuentes</td>
<td>Sections related to Business Adjustment, SLA Management and Reporting</td>
</tr>
<tr>
<td>1.5</td>
<td>29 Jun 10</td>
<td>Brosh</td>
<td>Updated design.time prediction feature</td>
</tr>
<tr>
<td>1.6</td>
<td>29 Jun 10</td>
<td>Baresi</td>
<td>Added text to manageability modelling</td>
</tr>
<tr>
<td>1.7</td>
<td>29 Jun 10</td>
<td>Kotsokalis</td>
<td>Updated status of A5 features</td>
</tr>
<tr>
<td>1.8</td>
<td>1st Jul 10</td>
<td>Theilmann</td>
<td>Added comments; updated feature &amp; evaluation service dependency decoding</td>
</tr>
<tr>
<td>1.9</td>
<td>1st Jul 10</td>
<td>Kotsokalis</td>
<td>Updated &quot;monitoring coordination&quot;</td>
</tr>
<tr>
<td>1.10</td>
<td>12th Jul 10</td>
<td>Kennedy</td>
<td>Updated A4 features</td>
</tr>
<tr>
<td>1.11</td>
<td>12th Jul 10</td>
<td>Bot</td>
<td>Added Runtime Prediction Information</td>
</tr>
<tr>
<td>1.12</td>
<td>14th Jul 10</td>
<td>Kennedy</td>
<td>Updated A4 features</td>
</tr>
<tr>
<td>1.13</td>
<td>20th Jul 10</td>
<td>Brosh</td>
<td>Updated details of Section 6.1.8 Runtime Infrastructure Metrics Prediction</td>
</tr>
<tr>
<td>1.14</td>
<td>20th Jul 10</td>
<td>Baresi</td>
<td>Updated A3 features</td>
</tr>
<tr>
<td>1.15</td>
<td>20th Jul 10</td>
<td>Foster</td>
<td>Updated Software monitoring rules extraction from SLA</td>
</tr>
<tr>
<td>Date</td>
<td>Day</td>
<td>Author</td>
<td>Notes</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>----------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>21st</td>
<td>July</td>
<td>Zampognaro</td>
<td>Updated section on dynamic binding setting, automatic binding</td>
</tr>
<tr>
<td>22nd</td>
<td>July</td>
<td>Fuentes</td>
<td>Update to Business Adjustment, SLA Management, and Reporting</td>
</tr>
<tr>
<td>23rd</td>
<td>July</td>
<td>Kearney</td>
<td>Updated evaluation of features &quot;SLAT Definition&quot; and &quot;Discovery of Services and SLAT&quot;</td>
</tr>
<tr>
<td>26th</td>
<td>July</td>
<td>Kotsokalis</td>
<td>A5-related updates.</td>
</tr>
<tr>
<td>27th</td>
<td>July</td>
<td>Theilmann</td>
<td>Updated A1 feature evaluation (lessons learned &amp; next steps)</td>
</tr>
<tr>
<td>28th</td>
<td>July</td>
<td>Fuentes</td>
<td>Lessons learned: 3rd party configuration</td>
</tr>
<tr>
<td>29th</td>
<td>July</td>
<td>Bot</td>
<td>Added Infrastructure Runtime Prediction</td>
</tr>
<tr>
<td>30th</td>
<td>July</td>
<td>Brosh</td>
<td>Updated lessons learned for &quot;Design of QoS-predictable Systems&quot; feature</td>
</tr>
<tr>
<td>5th</td>
<td>August</td>
<td>Fuentes</td>
<td>Update on Business Adjustment</td>
</tr>
<tr>
<td>5th</td>
<td>August</td>
<td>Edmonds</td>
<td>added lessons learned on A6</td>
</tr>
<tr>
<td>5th</td>
<td>August</td>
<td>Baresi</td>
<td>contribution on A3 features</td>
</tr>
<tr>
<td>6th</td>
<td>August</td>
<td>Stopar</td>
<td>Virtual hardware infrastructure adjustment lessons learned described.</td>
</tr>
<tr>
<td>6th</td>
<td>August</td>
<td>Vuk</td>
<td>Lessons Learned for Virtual Hardware Infrastructure Observation and Violation Detection and Virtual Hardware Infrastructure Reporting</td>
</tr>
<tr>
<td>8th</td>
<td>August</td>
<td>Kotsokalis</td>
<td>A5 related lessons learned.</td>
</tr>
<tr>
<td>9th</td>
<td>August</td>
<td>Vuk</td>
<td>added Lession Learned to Virtual Hardware Monitoring Rules Extraction from SLA</td>
</tr>
<tr>
<td>9th</td>
<td>August</td>
<td>Lorenzoli</td>
<td>Filled Runtime Software SLA Violations Prediction, Software Violation Detection</td>
</tr>
<tr>
<td>9th</td>
<td>August</td>
<td>Ellahi</td>
<td>Filled A3 sections</td>
</tr>
<tr>
<td>11th</td>
<td>August</td>
<td>Torelli</td>
<td>Added ENG sections</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>1.34</td>
<td>7&lt;sup&gt;th&lt;/sup&gt; Sept 10</td>
<td>Foster</td>
<td>Updated Software monitoring rules extraction from SLA</td>
</tr>
<tr>
<td>1.35</td>
<td>10&lt;sup&gt;th&lt;/sup&gt; Sept 10</td>
<td>Torelli</td>
<td>Implemented review suggestions</td>
</tr>
<tr>
<td>1.36</td>
<td>14&lt;sup&gt;th&lt;/sup&gt; Sept</td>
<td>Evenson</td>
<td>Updated TA contributions to section 3 and section 5.</td>
</tr>
<tr>
<td>1.37</td>
<td>15&lt;sup&gt;th&lt;/sup&gt; Sept</td>
<td>Torelli</td>
<td>Check before PMT review.</td>
</tr>
<tr>
<td>2.0</td>
<td>17&lt;sup&gt;th&lt;/sup&gt; Sept</td>
<td>Torelli</td>
<td>Applied PMT suggestions.</td>
</tr>
</tbody>
</table>
Executive Summary

The present document is the second version of DB1a. Some of the sections of this document are unchanged with respect to the first version, while others are new or updated. The new or changed sections are marked with a left-side blue border.

The main goal of DB1a is to present the requirements collected by work package B1 and to evaluate the scientific results produced by the A-Line with respect to such requirements.

In the first version of this deliverable we presented the process for requirements collection, the main features of the SLA@SOI framework identified during the execution of this process and a preliminary assessment of their implementation in Y1, performed by the A-Line. During Y1 we collected more than 200 requirements from internal and external sources. After a first analysis of the specified use cases against the scientific and technical objectives of the SLA@SOI framework, the B-Line decided to change or extend some of the use case scenarios, in order to better demonstrate the upcoming results from the A-Line. After the consolidation of this requirements, we found that the new scenarios covers all the scientific WP outcomes in a balanced way.

In this second version we added the technical evaluation of Y2 results performed both by the A-Line and by the B-Line use cases.

Following the production of a first version of the Framework components offering the requested features, the A-Line has provided a self-evaluation of the status of the implementation of each features. The A-Line evaluation has been performed through both automatic tests and a human assessments performed through a survey. The B-Line evaluation of the developed software has also been performed through a specific survey that collects the judgements of each use case work-package on the quality of each one of the developed features. Such judgments are of course based on the experience accumulated by the B-Line during the usage of the framework for the development of the use cases. This evaluation also include lessons learned and planned next steps. The results of this evaluation will be the main source for the identification of requirements still to be implemented in Y3. Another source will be the feedbacks received by external parties during the SLA@SOI presentation day and any important issue reported by use cases during the adoption process in the next months. The third version of DB1a will report on the requirements implemented during Y3.

As the technical evaluation is strictly related to the state of the art and to the scientific innovations of the Framework, also this version of the deliverable is accompanied with an updated report [1] on the state of the art. The A-Line specified in its evaluation the relationship of each feature with the scientific innovations.

While in the first version of DB1a we presented also a sketch of the business evaluation approach, in this version we describe just the technical evaluation, as in Y2 the WP B1 releases a specific deliverable [28] for the business evaluation. So any business related consideration has been moved to that deliverable.
# Table of Contents

1  Introduction  .................................................................................................................. 11  
2  State of the Art Analysis ............................................................................................... 12  
3  Requirements Process .................................................................................................... 14  
   3.1 Requirements Collection ......................................................................................... 15  
   3.1.1 Y3 Requirements Collection ............................................................................ 17  
   3.2 Requirements Validation ......................................................................................... 18  
   3.2.1 Y3 Requirements Validation ............................................................................ 20  
   3.3 Requirements Consolidation ................................................................................... 20  
   3.3.1 Y3 Requirements Consolidation ....................................................................... 22  
   3.4 Requirements Implementation ................................................................................. 22  
   3.4.1 Y3 Requirements Implementation .................................................................... 22  
   3.5 Industrial use case overview ................................................................................... 23  
   3.5.1 Use case B3: ERP Hosting (SAP) ...................................................................... 23  
   3.5.2 Use case B4: Enterprise IT (Intel) ..................................................................... 23  
   3.5.3 Use case B5: Service Aggregator (Telekom Austria) ........................................ 24  
   3.5.4 Use case B6: E-Government (ENG) ................................................................. 24  
   3.6 Overview of External Sources .................................................................................. 24  
   3.7 Requirements Analysis ............................................................................................. 25  
4  Consolidated Requirements ........................................................................................... 30  
   4.1.1 Framework Management ..................................................................................... 35  
   4.1.2 Design & Development ....................................................................................... 36  
   4.1.3 Pre-offering ......................................................................................................... 37  
   4.1.4 Service Offering .................................................................................................. 38  
   4.1.5 Service Negotiation ............................................................................................. 40  
   4.1.6 Service Provisioning ............................................................................................. 42  
   4.1.7 SLA Enforcement ................................................................................................. 44  
   4.1.8 Runtime Prediction .............................................................................................. 45  
   4.1.9 Service Monitoring .............................................................................................. 46  
   4.1.10 Service Reporting ............................................................................................. 47  
5  Evaluation Approach ....................................................................................................... 49  
   5.1 Evaluation Process .................................................................................................... 49  
   5.2 Scientific Metrics Evaluation .................................................................................... 50  
      5.2.1 Testing .............................................................................................................. 53  
      5.2.2 Survey ............................................................................................................. 54  
6  Evaluation Results ........................................................................................................... 56  
   6.1.1 Framework Management ..................................................................................... 62  
   6.1.2 Design & Development ....................................................................................... 66  
   6.1.3 Pre-offering ......................................................................................................... 70  
   6.1.4 Service Offering .................................................................................................. 72  
   6.1.5 Service Negotiation ............................................................................................. 77  
   6.1.6 Service Provisioning ............................................................................................. 82  
   6.1.7 SLA Enforcement ................................................................................................. 89  
   6.1.8 Runtime Prediction .............................................................................................. 94  
   6.1.9 Service Monitoring .............................................................................................. 96  
   6.1.10 Service Reporting ............................................................................................. 101  
7  Conclusions ...................................................................................................................... 106  
   7.1 Summary ................................................................................................................. 106  
   7.2 Outlook on Future Work ............................................................................................ 106  

References  107  
Appendix A: Glossary ........................................................................................................ 109
Table of Figures

Figure 1: Steps Of the Requirement Process ............................................................... 15
Figure 2: Data model of Requirements ................................................................. 17
Figure 3: Percentage of requirements by source .................................................. 26
Figure 4: Percentage of significant requirements .................................................. 26
Figure 5: Number of requirements by number of requests ...................................... 27
Figure 6: Use cases distribution of requirements .................................................. 28
Figure 7: Percentage of high and medium requirements by Action Line WPs .......... 28
Figure 8: Offline Features of the SLA@SOI Framework ........................................ 33
Figure 9: Runtime Features of the SLA@SOI Framework ....................................... 34
Figure 10: Requirement Evaluation Process ......................................................... 49
Figure 11: Average status reported by the A-Line for each category of features.57
Figure 12: Average quality of the implementation judged by the B-Line for each category of features. ................................................................. 57
Figure 13: Average level of complexity reported by the B-Line for each category of features.................................................................................................. 58
Figure 14: Average quality of the documentation judged by the B-Line for each category of features. ......................................................................................... 59
Figure 15: Average quality of the Architecture judged by the B-Line for each category of features. ................................................................................................. 60
Figure 16: Average quality of the APIs judged by the B-Line for each category of features. ........................................................................................................... 61
Figure 17: Number of Issues reported by the B-Line for each category of features. ...................................................................................................................... 61

List of Tables

Table 1: Required Framework Features (updated to Y2) ........................................... 32
Table 2: Requirements Evaluation Goals ............................................................... 51
Table 3 Evaluation Survey ..................................................................................... 55
Table 4: Technical evaluation of Use Case B3 ...................................................... 116
Table 5: Issues reported by Use Case B3 ............................................................... 116
Table 6: Technical evaluation of Use Case B4 ...................................................... 118
Table 7: Issues reported on Use Case B4 ............................................................... 120
Table 8: Technical evaluation of Use Case B5 ...................................................... 124
Table 9: Issues reported by Use Case B5 ............................................................... 124
Table 10: Technical evaluation of Use Case B6 .................................................... 127
Table 11: Issues reported by Use Case B6 ............................................................. 128
1 Introduction

The present document is the second version of DB1a. Some of the sections of this document are unchanged with respect to the first version, while others are new or updated. The new or changed sections are marked with a left-side blue border.

In the first version we presented the process for requirements collection, the main features of the SLA@SOI framework identified during the execution of this process and a preliminary assessment of their implementation in Y1, performed by the A-Line. In this second version of the deliverable we added the technical evaluation of Y2 results performed both by the A-Line and by the B-Line use cases. This evaluation also include lessons learned and planned next steps. The results of this evaluation will be the main source for the identification of requirements still to be implemented in Y3. Another source will be the feedbacks received by external parties during the SLA@SOI presentation day and any important issue reported by use cases during the adoption process in the next months. The third version of DB1a will report on the requirements implemented during Y3.

As the technical evaluation is strictly related to the state of the art and to the scientific innovations of the Framework, also this version of the deliverable is accompanied with an updated version of the state of the art and the A-Line reported in its evaluation the relationship of each feature with the scientific innovations.

The rest of this document is organized in the following way: Section 2 summarizes some of the considerations related to the State of the art that have influenced the project decisions. Section 3 describes the process used to collect requirements. Chapters 2 and 3 have not been changed with respect to the first version, a part some additional sub-sections (3.1.1, 3.2.1, 3.3.1 and 3.4.1) in Chapter 3. The consolidated requirements (requested framework features) resulting from the process are described in section 4. Also this section is unchanged with respect to the first version, with the exception of Table 1, where the mapping between features and requiring use cases has been updated on the base of the last development decisions of use cases.

Sections 5 describes the evaluation approach. It has been updated with respect to the first version and now describes in more detailed the technical evaluation. While in the first version we presented also a sketch of the business evaluation approach, in this version we describe just the technical evaluation, as in Y2 the WP B1 releases a specific deliverable for the business evaluation (DB1b). So any business evaluation has been moved to that deliverable.

Section 6 is new and presents the results of the evaluation. For each feature both the A-Line and B-Line evaluations are reported. As each use case provided its own evaluation, section 6 reports the consolidated results, while the evaluation of each use case is reported in Appendix D.

Finally, section 7 gives the conclusions and a summary of the future work.
2 State of the Art Analysis

The state of the art analysis of topics relevant to SLA@SOI includes input from all A-line work packages. It comprises 90 contributions to various topics such as SLA modelling, SLA translation, information modelling, business processing, SLA monitoring and prediction. Due to the huge number of works considered, detailed results of the analysis are presented in a separate report [1] accompanying this deliverable.

The report [1] reviews several types of significant works such as research papers, projects and standards. It includes a description of various functional and non-functional requirements taken into account by the analysed works, that may be potential candidates for inclusion in the SLA@SOI requirements (see also section 3.5). Furthermore, the report analyses the possibility of applying the results from the state of the art work to SLA@SOI and anticipates the necessary actions. To do the analysis, not only at a conceptual, but also at an implementation level, the report includes existing frameworks, components or services which realise the work and, where possible, states license model and maturity level.

The analysis has highlighted several limitations that the SLA@SOI project should try to overcome. First of all, we haven’t found any SLA management framework with complete coverage of eContracting, design-time and runtime prediction, software management, infrastructure management, as SLA@SOI attempts.

Covering such a huge set of functionalities in an integrated way requires a comprehensive model shared between the different components of the SLA@SOI platform.

We identified two main proposals pertinent for domain modelling in SLA@SOI: SID (NGOSS Shared Information and Data Model [2]), i.e. a set of standardized information definitions for modelling business entities developed by TeleManagement Forum, and WS-Agreement [3], i.e. an XML model and protocol developed from Open Grid Forum (OGF), for establishing agreement between parties. While the SID suffers of a limited support to SLA related concepts such as guarantee terms, penalties and rewards, WS-agreement is a very open standard that leaves many relevant aspects without a concrete definition both from the semantic and syntactic point of view. Moreover SID is more oriented to eContracting, whilst WS-Agreement is more concerned with software components, explicitly taking into account technical and low level aspects such as the WSDL interface of a service and its XML structure. In a sense the two specifications are complementary and a conceptual integration of the two would be valuable. A conceptual generalization is required also because SLA@SOI tries to apply the concept of service uniformly to all layers of the software stack.

There is no widely adopted solution for modelling an SLA hierarchy corresponding to a service composition hierarchy. Current works are often related to the composition of some specific QoS parameters, without proposing a general approach for relating SLAs between different layers.

The hierarchy of SLAs is strictly tied to the relationships between the software components that implements a service. Moreover the quality constraints expressed in a SLA are related to the quality of such components and of the related resources. Therefore, in order to automatically provision, predict and adjust the quality of services for satisfying SLAs, the SLA@SOI framework must
also supply a model to describe the software components that implement the services and their relationships.

As the SLA@SOI platform is also a software system realized through components, it would be desirable to use such a component model also to describe the SLA@SOI platform itself.

As a unifying standard for architectural components and in particular service components, we selected SCA (Service Component Architecture [4]) as the most promising specification. It has several implementations available and good chance to replace today's component technologies such as Java EE and .Net in the future. This model however does not cover several needs. In particular there is no possibility to associate, to a component, non-functional properties and information needed for prediction of their values. More in general there is no support in SCA for SLA aware management.

While various research efforts have centred on service management based on SLAs, currently there is no widely adopted solution for monitoring of SLAs or for dynamic binding of services based on constrains expressed in SLAs. Solutions, need to be found, which are as compatible as possible with current standards and tools supporting SLA aware service management.

A particularly challenging objective is QoS prediction. Some of analysed approaches take into account single system properties only, e.g., CPU load, disk usage or network traffic. Many of the approaches use regression models, trend analysis and machine learning techniques, which all require a training session before being able to predict. To extend the applicability of these methods it is necessary to remove some of the above limitations.

At the infrastructure level we lack flexible and technology-independent models and interfaces through which both functional and non-functional aspects of infrastructure can be described and managed. For instance the Open Virtual Machine Format (OVF) allows to describe functional properties of a virtual machine but it does not describe non-functional properties. In contrast, specific technologies such as Ganglia for monitoring, can be considered very mature and have been adopted in SLA@SOI without any need for extensions. Also XMPP (set of open XML technologies for presence and real-time communication) has been identified as a good standard for distributed messaging at infrastructural level. No standard or widely adopted solution, however, is available for dynamic provision and re-provisioning.

Details on all the relevant technologies that are being used and improved by each scientific WP are given in Section 4 “Status of Implementation”.
3 Requirements Process

The abstract requirement process for SLA@SOI consists of four sequential steps:

1. Collection
2. Validation
3. Consolidation
4. Implementation

During the collection phase for Y2 we sought to gather the requirements from four sources:

1. Internal Questionnaire
2. Internal requirements from Use Cases
3. external requirements (partners)
4. external requirements (SotA)

A first order plausibility check was then performed, comprising the categorization of these requirements on a web-based database system. By placing the requirements online we sought to facilitate communication between the consortium members.

The purpose of the validation phase was to improve the quality of the requirements by normalizing the values in relationship to the other requirements and to get stable and cross-checked requirements from B-line.

The goal of the consolidation step in the process was to assign requirements to a defined feature or feature-group. Following which, the requirements were checked by A-line, and their feasibility verified.

The aim of the implementation phase is to have implemented the features which are necessary for the satisfaction of the requirements.

For the third iteration of gathering requirements for SLA@SOI, we will retain the abstract structure followed in Y2 as depicted in Figure 1 on page 15. But we specialize this abstraction to fit the constraints that this will be the last iteration of the process in the SLA@SOI project and that we expect that further refinement will be undertaken by the community that arises out of the integrated Open Source release starting in the end of M28 (September 2010). As such we anticipate that not all requirements we gather in the current phase will necessarily be implemented within the scope of the SLA@SOI project, but might well serve to guide development external to the consortium or which occurs after the termination of the consortium agreement.
Figure 1 shows the relationships between the four steps of the process. In Sections 3.1-3.4 we give a more detailed description of each phase. In Sections 3.5 and 3.6, we give a brief overview of the internal and external sources of requirements. Finally in sections 3.7 and chapter 4 on Consolidated Requirements we describe the actual results of the requirement process, first from a quantitative point (Requirements Analysis) of view and then from a functional point of view (Consolidated Requirements).

3.1 Requirements Collection

This section describes the requirements gathering process through the first year of the project, with the modifications for Y3 contained in section 3.1.1. Requirements were collected from sources both internal and external to the consortium covering as broad range of use cases as possible.

The complete process consisted of four phases, namely

- Collection of requirements from internal industrial use cases
- Collection of requirements from external entities (research community, industry etc.) and SoTA analysis
- Incorporation into a ticket tracking system (Trac)
- Review of requirements made by action line A and B work packages.

The process is described in detail below. It is evident that this is an iterative process of constantly refining and improving the catalogue of requirements.

For the internal requirements, we started by collecting a list of questions and topics for each scientific work package. Due to many overlaps in the list, we performed an additional clustering and categorization of questions to create a questionnaire which consisted of about 140 questions. Questions were divided into two main categories, namely Business View and Technological View. Business View questions were concerned with an end user perspective and the involved stakeholders. It consisted of questions about involved stakeholders, the nature of
services, the IT systems, user stories, service discovery, modelling assumptions, monitoring aspects and design time prediction. Technological questions solicited information on the customer profile, the service description standards, infrastructure implementation details, ability for dynamic re-provisioning, interaction with third-party providers, and service manageability. The purpose of the questionnaire was to elicit expectations of technologies and features for implementations of industrial use cases which would start in M12.

The results of the surveys as filled out by each industrial use case partner were then collated into a single document. Out of the collation of the summarized results, 116 specific requirements were synthesized, each described in detail and listing relevant use cases and an educated guess as to which A-line work package would be involved in the implementation of these requirements. To capture use case specific requirements that could not be elicited directly by such a general survey, the individual use case specifications were analysed as an additional source of requirements (in particular; internal report D.B[3-6].1a released at M6, draft of D.B[3-6].1 available at M10, and direct communication from use case leaders).

External requirements were gathered from both the state-of-the-art survey [1] and by seeking feedback from projects which had been identified as potential collaborators.

All this information was entered into an online database associated with the shared code repository of the consortium and accessible to it. The software we used was a customized version of the Open Source project management and feature tracking solution Trac [20]. We customized this database to fit the data model shown in Figure 2. The purpose of this customisation was to differentiate the requirements by the affected work package and to facilitate the assignment of priorities to each requirement. When we entered the collected requirements, we sought to “collapse” requirements as much as possible by consolidating requirements where it made sense. We intended that this online, shared solution should be a lightweight coordination mechanism, which would be easier to track than something like a centrally maintained spreadsheet. As a bonus, we intend to use the database to track the actual software issues as the project enters the phase where these requirements are to be implemented.
Following online collation of the raw requirements, the industrial use case partners then went over all the collected information, clarifying the description and understanding of the requirements as summarized, and noting the revised dependencies.

### 3.1.1 Y3 Requirements Collection

For the collection of requirements for the final iteration of SLA@SOI development we again seek to obtain information from both internal and external sources, but face severely compressed time horizons in which to complete this task.

For the gathering of requirements from internal to the SLA@SOI, we first needed the evaluation of the Y2 implementations to be completed, and then sufficient time for these results to be properly considered to come up with perspective. Driven by the need to get the initial Open Source release ready for the end of September 2010 (M28) along with the continuous daily improvements contributed to the source tree, we do not really have a moment at which forward motion of the project can be said to cease to have the proper Archimedean point from which to form such an evaluation. Fortunately, the customized online tooling in TRAC for gathering requirements that we spent a fair amount of effort in the first two-
thirds of the project can be utilized to provide an asynchronous mechanism by
which requirements can be entered. We will solicit a small (from 5 – 10) number
of requirements from each use case partner to be entered into this database that
represents the crucial requirements needed for implementation in Y3.

For gathering requirements from external partners, we shift our primary attention
from our state-of-the-art survey and collaboration partners to that afforded by
the prospective users of the FOSS (Free and Open Source Software) release
heralded by the events surrounding the ICT information day at the end of
September 2010 (M28). Expecting to have at least twenty representatives from
different companies present at our hosted event, we will present a small survey
to understand anticipated uses of the framework outside the internal use cases.
We do not expect to be able to collect fully featured requirements from such a
short interaction, we will assess the possibilities of implementing an interesting
subset of ideas so gathered within the context of the Y2 iteration of the B2 Open
Reference Case so that we will have a stakeholder for their implementation.

In the final round of requirements collection, there may well be requirements
which are certainly interesting but for which our resources are too limited for a
reasonable chance of successful implementation. For the sake of the FOSS legacy
following the funded aspects of SLA@SOI, such requirements will be retained for
possible future adoption.

3.2 Requirements Validation

After the completion of the requirement collection, the requirement validation
process was instigated to improve the quality of the requirements. This process
entailed that the collected requirements again be checked by all B-line partners.
In particular, the B-line WP-partners set priorities for each requirement,
expressing the importance of the requirement for each B-line-WP.

The first step in the validation process was to export the data; converting the
requirements from the TRAC-system into a spreadsheet including all fields. A
report based on this spreadsheet was used for statistical and monitoring
purposes. From the spreadsheet a statistical analysis concerning the status of the
validation was done. This information was sent to the WP-leads.

The B-line WP-leads set again priorities of requirements on Trac. A new priority
value "zero" was then added by which we could denote that a use case had
evaluated the requirement but had no need for the fulfilment of the requirement
for their work. For each requirement that a use case-partner was interested in,
the email address of the nominated partner was added to the field "CC" in Trac. If
there were any changes or new comments in the Trac concerning the
requirement, the nominated partners contained in this field were sent an
automatic notification via e-mail. The comment-field in Trac was then used for
collaborating with other use cases to improve the description of the requirement.

The improvement of the description of the requirements has been done in
collaboration with other use cases that shared the need for the same
requirement. On Trac, a comment to the requirement which should be improved
was then added for producing an improved description. Trac automatically sends
an email to all interested parties when any aspect of the specific requirement is
modified. Any receiver of this notification has the ability to reply by adding
another comment on TRAC. If all related use cases accept the proposed modification, the use case initiator of the proposed change then modifies the description text. The overall priority is set once the description is judged sufficient. Otherwise the priority is set to zero and additional requirements proposed if necessary.

Any missing requirements were directly added on TRAC from B-line WP-partners, with the responsible WP-partner for the new requirement adding a ticket in TRAC including a precise description.

The monitoring of the progress during the validation of the requirements was done via TRAC. Every week a spreadsheet with the requirements from TRAC was created, and an evaluation of the current state of each requirement performed. The results were documented and depicted in a report (including statistics) and sent to all responsible A- and B-line WP-partners. The status field was set depending on the B-line WP-priorities from B1 (checked or cancelled) during the weekly TRAC-reports (monitoring). The requirements with status "checked" were then assigned to a "responsible" A-line-WP and signalled to A-Line.

At that point A-Line continued with the next step of the requirement process: the "consolidation".

In the case that all B-line-WP-priorities were set to "zero" the status of the requirement was set to "cancelled" (B1 TA) after a check and acceptance from B1.

General conditions:
The following prioritisations were defined in Trac and chosen from each B-line WP:

- Zero (not relevant for this use case)
- Low (nice to have)
- Medium (must-have for year 3)
- High (must-have year 2)

Before the start of the validation of the requirements all priorities in Trac were set to "none" to get a defined initial state. The first time a requirement was entered into the Trac-system, its status was set to "proposed".

During validation, the "topic expert" updated also the following fields in Trac:

- Check/adaptation of the short description
- Priority (B-line-WP-priority)
- Role/candidate role
- Scientific WP
- Key words (name of the corresponding features)
- Component (could be overwritten by A-line)
- Framework lifecycle (could be overwritten by A-line)
- SLA layer/Subject
The update of the fields ‘subject’, ‘priority’, ‘layer’ and ‘role’ was made mandatory in order to collect information necessary for the normalization of the requirements into features.

A definition of the candidate roles can be found in the D.A1a glossary [21]. Each requirement is related to a candidate role within Trac.

The following status values were used in Trac to capture the current state of each requirement:

- proposed
- checked
- accepted
- rejected
- designed
- implemented
- tested
- issues
- approved
- cancelled
- failed

A definition/description of the above status values that are used in TRAC can be found in appendix C.

### 3.2.1 Y3 Requirements Validation

Once the requirements have been collected in the online database they will have been edited to achieve some degree of consistency by a single organizational entity. Again initial guesses at initial A-line responsibilities will be made, and the A-line scientific workpackages asked to check the coherency and valid responsibility. Initial priority of all requirements will be considered “MUST”, but given time constraints it will be possible to an A-line that considers the requirement useful but beyond the resources of our project, the priority of “FOSS” may be assigned, indicating that its implementation must occur outside the scope of the funded projected within the open source framework we endeavour to engender. With attendant emails, online summaries, and conference calls we anticipate this phase to be completed by the middle of October 2010 (M29).

### 3.3 Requirements Consolidation

Once all priorities and fields in TRAC were filled accordingly, the status of the requirement was set to “checked” during the monitoring/reporting from B1 (TA).

Based on the validated requirements from B-line (requirements in state “checked”) an analysis from A-line was done. The result of the evaluation from A-line was a set of features, which are necessary for an implementation of the requirement.

The features were clustered in groups called topics. The following topics were defined:
1. Business
2. Provisioning, Monitoring and Adjustments
3. Prediction
4. Integration (including modularization, configuration, SW development guidelines)

The resulting feature set and topics gathered from the analyses of the requirements formed the basis for the further consolidation of the requirements.

For each topic, a topic expert, belonging to B1 WP, was nominated. As B1 partners are also A-Line partners, they have the knowledge to take the responsibility of scientific topics. The topic experts have the responsibility for the quality and coverage of requirements for their topic.

The nominated persons assigned by B1 are the same persons who have already researched or were going to research the relevant topic in the A-Line. These persons were committed to participate in B1 and were in charge of performing the consolidation process, which is a B1 responsibility. In this way A-Line people have one single point of contact for each topic who participates also to the B1 work and is able to directly answer to A-Line questions and/or contact relevant use case leaders to resolve open issues.

**Tasks of Topic Experts:**

During the task of correlation of requirements to features, the topic experts also checked if all (or at least the mandatory) fields in Trac were filled. The topic experts could also decide to split a requirement if that would improve the intelligibility of the requirement.

If there were duplicated requirements in Trac, the topic experts should decide to cancel one of them. In this case a reference in the Trac field ‘keywords’ is used to record the cancellation. The field ‘keywords’ should be used to note that a duplicated requirement has been cancelled and reference the remaining requirement.

If during the analysis, topic experts noticed that a requirement was missing, they could decide to add a new requirement in TRAC. If non functional features needed be added (as current features represent functionalities of the framework) this was done by adding them on the feature-list and notifying B1.

The topic experts were individually responsible for deciding how to make the consolidation. The main constraint was the further use of TRAC for the requirement process. When the screening of the requirements by the topic experts was completed, the A-line was informed to check the feasibility of the requirements.

Each responsible A-line-WP first checked that they understood the requirements. If the responsible A-line WP-partner did not understand a checked requirement (s)he added a comment to it, containing or referring to a request for clarification to the use cases and setting the status of the requirement to "issues". The clarification was done by contacting the responsible B-line partner. If this was not possible, the clarification happened via the topic expert. The topic expert was the single point of contact for A-line-WP-partners in this case. If a requirement involved more than one component, A-line created a new sub-requirement for each component and related it to the original B-Line requirement. The related WP also associated with each sub-requirement a specific unit-test. During the consolidation phase, the responsible A-line workpackage may discover that several B-Line requirements are specializations of a more general requirement. In this case the A-line added on Trac such a generalized requirement, related the B-
line requirements to it and specified an implementation for the generalized requirement. When the status of a generalized requirement was changed by A-Line, its related B-Line requirements were also be set by A-line to the same value.

After its evaluation by the A-line, a requirement was set to “rejected” if an implementation of the requirement was considered not to be possible or if the realisation had no impact to the architecture from the perspective of A-line. In the other case, the status of the requirement was instead set to “accepted” by the A-line.

The consolidation phase was completed after all requirements were checked and assigned to a defined feature.

### 3.3.1 Y3 Requirements Consolidation

We will follow the same procedure as outlined for Y2, with the same topic experts providing the cross work package feasibility and possibility checks. We anticipate that the work will be completed by the end of October 2010 (M29).

### 3.4 Requirements Implementation

The implementation phase follows the first cycle of requirements consolidation. The requirements have been checked. The analyses of the requirements provided a stable feature-list (see Section 4).

The responsible A-line WP-partner decides how to realize the requirements associated to a same feature and add a comment that directly specifies or refers to a wiki-page or SVN document providing details on the use of the feature. Suggested details for functional features are the following:

1. Components which realize that requirements.
2. If some configuration of the framework or some special deployment option must be adopted from the use case that has the requirements.
3. An UML sequence diagram, pseudo code or some other equivalent representation that specifies how the use case must interact with the framework interfaces in order to access the feature.
4. An integration test for checking if the requirements has been satisfied.

After acceptance of the proposed architecture (defined through step (a) – (d)) by B-line, A-line incorporates the requirement in the feature-list and sets its status to “designed”. Once A-line has completed the implementation of the requirement, its status in TRAC is set to “implemented” by A-line.

The process then continues with the evaluation process (see Section 5).

If during the implementation phase a feature split, additional/new requirements emerge, or an adaptation of requirements is deemed necessary, the process for these requirements will start again with the consolidation step.

### 3.4.1 Y3 Requirements Implementation

The most glaring problem with the evaluation of the requirements implementation procedure we followed for Y2 lay in the gross underestimation of how much time it would take to implement and then integrate the result of these implementations. We will place even greater stress on the integration scenarios
for Y3 development, lessening even further the emphasis on unit tests to assess the progress of implementation. Other than that modification, we currently intend to utilize the requirement evaluation metrics as we modified them during their use in Y2 as described in Section 5.

### 3.5 Industrial use case overview

In order to fulfil the scientific goals set by the SLA@SOI consortium, a number of industrial use cases were included in the project. Industrial use cases were chosen with a very broad field of interested stakeholders, including infrastructure providers, large service providers; social and communication services. In the following sections, we provide a brief introduction to all these use cases.

#### 3.5.1 Use case B3: ERP Hosting (SAP)

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

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

#### 3.5.2 Use case B4: Enterprise IT (Intel)

The Enterprise IT use case deals with SLA aware dynamic provisioning through Service Oriented Architectures. It will examine SLA negotiation and automation and how IT can dynamically reflect the priorities of the business.

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

---

1. This deliverable does not consider the use case B7, (“Financial Grid”) as insufficient progress has been achieved in its definition.
3.5.3 Use case B5: Service Aggregator (Telekom Austria)

The service aggregator use case will demonstrate the aggregation of SLA-aware telecommunication and third party web-based services. The use case will involve the construction of a telecommunications as a service (TaaS) platform over telco and compute infrastructures. The TaaS will then be used to examine scenarios in which multi-party, multi-domain SLAs for aggregated services are offered to customers. The use of information provided by customer relationship management systems will be utilised to personalise the business aspects of the SLA lifecycle to improve the quality and satisfaction of the aggregated service.

3.5.4 Use case B6: E-Government (ENG)

The E-Government use case is intended to apply automatic SLA management to social services that mixes activities with human based activities. While most government services cannot be automatic, several management activities such as monitoring, resource planning, selection, negotiation can be at least partially automated thanks to the adoption of formal SLAs.

The e-Government use case will integrate two kind of social services: the medical treatment services and the mobility services for elderly people. Three scenarios will be developed. The first one will demonstrate the application of the monitoring framework of SLA@SOI to the integration of human based and automatic services. The second scenario will show how third party providers can be dynamically and automatically chosen by the system on the base of the citizen needs and of pre-signed SLAs. Finally the third scenario will apply automatic resource adjustment and automatic SLA negotiation with third parties to adapt to unforeseen changes in consumer demand.

3.6 Overview of External Sources

In the second round of requirements gathering a shorter questionnaire was prepared and distributed through established collaboration channels to external entities. We've received and included the following projects:

- SOA4ALL, a Large-Scale Integrated Project funded by the European Seventh Framework Programme. It aims at realizing a world where billions of parties are exposing and consuming services via advanced Web technology.
- Q-ImPrESS aims to bring service orientation to critical application domains, such as industrial production control, telecommunication and critical enterprise applications, where guaranteed end-to-end quality of service is particularly important
- TEXO contributes to service economy by creating infrastructure components for Business Webs in the Internet of Services.

Moreover other external requirements were identified by analysis of the literature[1]. In particular we selected requirements coming from the following sources:
• SCENE, a service composition execution environment supporting binding, re-binding and self-reconfiguration operations. SCENE is part of SeCSE.

• SeCSE, a European IP, which ended in November 2008 and was aimed at creating new methods, tools and techniques to support the cost-effective development and use of dependable services and service-centric applications.

• TMF SID, a set of comprehensive standardized information definitions, developed by the TeleManagement Forum (TMF), acting as the common language for building easy to integrate OSS (Operational Support System) and BSS (Business Support System) solutions.

• TMF SDF, Service Delivery Framework. A work aimed to facilitate service syndication and end-to-end management of services.

• TM Forum SLA Management Handbook, a handbook to assist two parties in developing a Service Level Agreement, by providing a practical view of the fundamental issues.

• WBEM, Web-based Enterprise Management (WBEM) Standards. A set of standards aimed to unify the management of distributed computing environments.

• JMX, Java Management Extensions. A standard developed by Sun Microsystems, part of the Java Platform Standard Edition, used to manage resources such as applications, devices and services with the Java Programming Language.

• DIANE, an approach and a language developed by University of Jena and University of Karlsruhe for Automated Service Discovery, Matchmaking and Composition.

• CentraSite, a standards-based SOA registry and repository jointly developed by Fujitsu and Software AG that aims at greater visibility and control of integrated SOA based applications, better support on decision-making, and increased productivity.

3.7 Requirements Analysis

At the end of the consolidation process, we counted a total of 233 requirements. Fifteen percent (15%) of these requirements came from external sources (collected through direct interaction with external parties or by analysis of the literature identified as part of the state of the art review). Figure 3 shows the percentage of requirements identified through the different types of targeted resources.
During the revision steps, 48 requirements were closed as they were considered out of scope after further analysis. If all priorities of a requirement had been set to “zero” by all B-line WPs, the status of the requirement had been set to “closed” after a check from B1-WP-lead and A-line.

Of the remaining 185 requirements, 70 were asked by all the use cases (although with different priority), and 51 were considered useful by at least 3 use cases.

An overall priority was assigned to each requirement. Possible values for the “overall priorities” were:

- **Blocker**: at least one use case assigned high priority
- **Critical**: at least two use cases assigned medium priority
- **Major**: at least one use case assigned medium priority
- **Low**: at least one use case assigned minor priority
We defined as “significant” the requirements whose priority is higher than Low. As 16 requirements are considered of Low priority, we collected 169 significant requirements. The important percentage of requirements closed or with low priority may be ascribed to different factors such as changes in the use cases objectives, evolution of the overall project understanding, non-applicability of external requirements to SLA@SOI project and so on.

![Figure 5: Number of requirements by number of requests](image)

Figure 5 shows how many requirements correspond to the overall priorities “blocker”, “critical”, “major” and “minor”. Of the significant requirements, 116 are blocker, i.e. they are judged of high priority from at least one use case. Therefore these 116 requirements were identified as the most important candidates for v1 of the SLA@SOI framework.

Figure 6 shows how many requirements were required by each use case. It is evident that use cases B5 and B3 are the most demanding in terms of number of requirements, especially medium ones.
Despite the apparent homogeneity of these figures, there are actually important differences between the use cases. There are 59 requirements that have been assigned priority “high” by more than 2 use cases. Moreover amongst the “blocker” requirements there were 10 requirements that were required by just one use case.

It is also interesting to see the distribution of requirements with respect to Action Line A WPs. The next figure shows the percentage of requirements needed from each action line A WP by each use case. The figure refers just to the significant requirements.

All use cases are distributed on scientific WPs in a similar way and have the highest percentage of requirements on A5. From the picture there is evidence of some use cases peculiarities. B6 is the use case with fewest requirements impacting on A4; indeed most of the requirements associated to A4 are about infrastructure hardware management, while B6 is more concerned with human based infrastructure. Use cases B4 and B6 represents a little portion of the
requirements on prediction, indeed most prediction requirements are about software services, which are less relevant in the aforementioned use cases.

It is evident that the most critical scientific WP, with respect to use cases requirements, is A5. This fact was expected, as A5 is a foundation for the other WPs. Most of the A5 requirements concern the SLA Model. Some of these requirements need a relatively simple implementation because they just require adding a particular kind of information to the SLA model. Others, however, are more difficult to implement as they have important repercussions on the other WPs too (e.g. monitoring any new kind of constraint included in the SLA Model). Therefore it is important to appreciate that the number of use case requirements associated with a WP should not be viewed as a measure of the implementation effort of that WP, but as an indication of which is the first WP, in the implementation dependency chain, that is impacted by the requirements.

The minor relevance of A1 with respect to use case requirements is also obvious, as at least at this stage the use cases are more concerned with functionalities offered by the SLA@SOI framework, rather than with architectural requirements or other non functional requirements. We have to consider that at this stage there are likely several technical requirements not explicitly expressed because they are implicit in the SLA@SOI assumptions, such as the development of an integrated framework.
4 Consolidated Requirements

In this section we describe the final formulation of the SLA@SOI framework requirements, coming from the consolidation process and the further feedbacks received during Y2.

As of month 17, some aspects of the use cases were refined or extended. Those changes were needed to assure that the use cases all together are able to demonstrate all the scientific and technical results of the SLA@SOI project. At the end the formulation of some requirements were outdated with respect to the current expectations of the B-Line for the A-Line. For this reason the consolidation process was preceded by a request to all the use cases to update the description of requirements and their priorities.

Afterwards, the collected requirements were analysed by WP B1 in order to remove redundant requirements and reduce the ambiguities in each requirement. To this end the requirements were split into four big groups according to 4 topics: business, provisioning-monitoring-adjustment, prediction, and integration. Each group of requirements was analysed by a person (called Topic Expert) belonging to a B1 partners. Each topic expert closed duplicates of requirements, clarified the relationship between them, and aggregated the remaining requirements to a defined and specific functionality or characteristic of the framework, called “feature”. These features were then reorganized in categories corresponding mainly to service lifecycle phases and application development phases.

The initial list of features was agreed during a dedicated meeting between A-Line and B-Line called “matchmaking meeting” and then refined. Each feature was associated with a specific A-Line WP and a specific step of the service lifecycle. While we maintain the association between the feature and the original collected requirements, the features represent the consolidated version of the relevant requirements and will be used as reference for the implementation of the framework, the adoption by use case and the evaluation process.

Each feature is defined in order to assign the responsibility of its development to just one scientific work-package. For this reason, we included features coming from the splitting of the needs of use cases to the various scientific work packages. Therefore the features take into account both the needs of the use cases and the technical dependencies between scientific development tasks.

Some of the features in the category “Framework Management” and some features in “Design and Development” correspond to non functional aspects or to development facilities. All the other features correspond to functionalities that the framework is expected to offer. Some of these functionalities are internal to the framework, while others are offered to external actors or applications. Some features are specific to a certain layer, while others are cross layer. Therefore when applicable each feature has been associated to a specific external actor (user) and to a specific layer, in accordance with the terminology standardized by the SLA@SOI glossary.
The features have been validated (i.e. accepted) by the A-Line, i.e. each scientific WP has recognized that they are able to understand the feature and the original requirements associated to them.

The following table gives a summary of all the identified features organized according to the decided categories. The table shows the scientific work package associated to the feature and the use cases that will use them according to the last definition of use case scenarios:

<table>
<thead>
<tr>
<th>WP</th>
<th>Feature</th>
<th>Use Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>B3</td>
</tr>
<tr>
<td></td>
<td>Framework Management</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>Framework Configuration &amp; Setup</td>
<td>X</td>
</tr>
<tr>
<td>A1</td>
<td>Framework Model Configuration</td>
<td>X</td>
</tr>
<tr>
<td>A1</td>
<td>Framework Operation</td>
<td>X</td>
</tr>
<tr>
<td>A1</td>
<td>Framework Access</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Design &amp; Development</td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>Reference Software Manageability Components and Configuration</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>Service properties dependencies coding</td>
<td>X</td>
</tr>
<tr>
<td>A6</td>
<td>Designing of QoS predictable systems</td>
<td>X</td>
</tr>
<tr>
<td>A6</td>
<td>Manageability Design</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pre-offering</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>Customer Registration</td>
<td>X</td>
</tr>
<tr>
<td>A2</td>
<td>3rd Parties Configuration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Service Offering</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>Discovery of Serv. and SLAT</td>
<td>X</td>
</tr>
<tr>
<td>A2</td>
<td>Product Management</td>
<td>X</td>
</tr>
<tr>
<td>A2</td>
<td>Product Discovery</td>
<td>X</td>
</tr>
<tr>
<td>A2</td>
<td>Product Definition</td>
<td>X</td>
</tr>
<tr>
<td>A5</td>
<td>SLAT Definition</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Service Negotiation</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>Business Negotiation</td>
<td>X</td>
</tr>
<tr>
<td>A2</td>
<td>SLAT Customization</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>Business SLA planning for negotiation</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>Automatic orchestration of customisable (re)negotiation protocol</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>Out-of-band SLA Registration</td>
<td>X</td>
</tr>
<tr>
<td>WP</td>
<td>Feature</td>
<td>Use Cases</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B3</td>
</tr>
<tr>
<td>Service Provisioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>Coordination of Provisioning</td>
<td>X</td>
</tr>
<tr>
<td>A5</td>
<td>On the fly deploy of monitoring</td>
<td>X</td>
</tr>
<tr>
<td>A3</td>
<td>Dynamic binding setting</td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>Virtual hardware infrastructure provisioning</td>
<td>X</td>
</tr>
<tr>
<td>A3</td>
<td>Software provisioning</td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>Software landscape</td>
<td>X</td>
</tr>
<tr>
<td>SLA enforcement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>Automated SLA enforcement coordination</td>
<td>X</td>
</tr>
<tr>
<td>A4</td>
<td>Virtual hw. infrastructure Adjustment</td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>Software Adjustment</td>
<td>X</td>
</tr>
<tr>
<td>A3</td>
<td>Automatic Binding</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>Business adjustment</td>
<td></td>
</tr>
<tr>
<td>Runtime Prediction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>Runtime infrastructure metrics prediction</td>
<td>X</td>
</tr>
<tr>
<td>A6</td>
<td>Runtime Software SLAs violations prediction</td>
<td></td>
</tr>
<tr>
<td>Service Monitoring</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>Monitoring coordination</td>
<td>X</td>
</tr>
<tr>
<td>A4</td>
<td>Virtual hardware infrastructure monitoring rules extraction from SLA</td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>Virtual hardware infrastructure observation and violation detection</td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>Software violation detection</td>
<td>X</td>
</tr>
<tr>
<td>A3</td>
<td>Software monitoring rules extraction from SLA</td>
<td>X</td>
</tr>
<tr>
<td>Service Reporting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>SLA Management</td>
<td>X</td>
</tr>
<tr>
<td>A2</td>
<td>Push/Pull Business SLA Reporting</td>
<td>X</td>
</tr>
<tr>
<td>A5</td>
<td>SLA Status &amp; Resource Reporting</td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>Virtual hardware Infrastructure Reporting</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1: Required Framework Features (updated to Y2)
Figure 9 and Figure 10 depict the main relationships between the functional features (i.e. features that represent activities or functionalities) offered by the SLA@SOI Framework and the relationship with the external actors.

Each feature is represented as a UML Use Case. We distinguish between offline features and runtime features. An offline feature represents an activity that the SLA@SOI Framework must support in the phases that precede the provision of the services, such as the design and implementation of the service components and the configuration of the Framework. Such operations typically happen when the Framework is not running. Runtime features are functionalities offered by the Framework at runtime. Different colours are used to distinguish UML Use Cases belonging to different Feature Categories.

![Feature Categories Diagram]

**Figure 8: Offline Features of the SLA@SOI Framework**

The UML diagram for Runtime Features clearly distinguish functionalities that are directly accessed by external actors, from functionalities that are internal to the framework, but needed to perform the functions asked by the external actors. Classical UML Use Cases dependencies are used to show the relationship between the features. A dependency between the features will likely imply a dependency between the implementation of the features at least at interface level.
In the following sections we describe each of the identified features. A-Line is expected to further enrich the definition of features during the design and development phases, to obtain a more general and integrated framework. Moreover the features might be enriched during the second half of Y2 based on new requirements coming from the experience of use of the framework.
4.1.1 Framework Management

Framework Configuration & Setup

**Responsible WP:** A1  
**External Actors:** Framework Architect  
**SLA Layers:** all  
**Overall priority:** Critical

The feature “Framework Configuration & Setup” comprises a set of four validated requirements describing the use case needs on how the overall SLA framework can be configured and set up. The main need in this area is about the flexibility of using the framework in different ways, i.e. the framework configurability in terms of (a) which components are used, (b) the usage of components in multiple instances and (c) the replacement of components with domain-specific instances. Furthermore, use cases require multi-domain setups, including 3rd parties and the operation of a service registry by the service provider.

Framework Model Configuration

**Responsible WP:** A1  
**External Actors:** Framework Architect  
**SLA Layers:** all  
**Overall priority:** Blocker

The feature “Framework Model Configuration” comprises two validated requirements describing the use case needs on how to parameterise the basic data models of the framework. The main need in this area is about flexibility in adapting and extending the basic SLA model with additional basic metrics and domain specific parameters (on guarantee terms, guaranteed actions, etc.).

Framework Operation

**Responsible WP:** A1  
**External Actors:** Framework Architect  
**SLA Layers:** all  
**Overall priority:** Blocker

The feature “Framework Operation” comprises two validated requirements describing the use case needs on how the overall SLA framework shall operate. Basically, the framework shall work under Linux, Unix and Windows operating systems and should ideally also come with default implementation support for managing common middleware such as Apache or Tomcat.
**Framework Access**

*Responsible WP: A1*

*External Actors: Framework Architect*

*SLA Layers: all*

*Overall priority: Blocker*

The feature “Framework Access” comprises a set of five validated requirements describing the use case needs on how the overall SLA framework can be accessed by different kinds of internal and external clients. Main needs in this area include basic library access via Java, remote interfaces for building blocks via WSDL/SOAP and browser-based graphical user interfaces.

**4.1.2 Design & Development**

**Reference Software Manageability Components and Configuration**

*Responsible WP: A3*

*External Actors: Service Provider*

*SLA Layers: all*

*Overall priority: Blocker*

This feature must offer modelling and development solutions to simplify the monitoring of common SOA software resources. This must include the implementation of specific instrumentations for the monitoring of service workflows and web services. This is needed in particular to simplify the adoption of the platform for applications that need to monitor composed services such as use case B6.

**Service properties dependencies coding**

*Responsible WP: A1*

*External Actors: Software Designer*

*SLA Layers: Software Layer*

*Overall priority: Blocker*

This feature, used by the software designer at service development time, allows the encoding of how a service depends on other services. It is needed to support automatic negotiation and planning, especially with lower level SLA managers and third parties, and for determining responsible parties for violations of the composed SLA.
Designing of QoS predictable systems

**Responsible WP:** A6

**External Actors:** Software Designer

**SLA Layers:** Software Layer

**Overall priority:** Blocker

This feature is aimed at the creation of QoS-enabled architectural models, including performance annotations that can be used for early design-time prediction of component-based software. The feature includes a design-time model repository for facilitating the reuse of QoS aware models, Eclipse base tooling to support the design and development process and methodologies for calibration and validation of component’s performance.

Manageability Design

**Responsible WP:** A6

**External Actors:** Software Designer

**SLA Layers:** Software Layer

**Overall priority:** Major

This feature will enable software developers to specify basic and event-based monitoring capabilities for software components at design type, resulting in a monitoring-enabled architectural design. The models will also present management (long term) and control (short term) interfaces in order to support the manageability life-cycle of the software layer.

4.1.3 Pre-offering

Customer registration

**Responsible WP:** A2

**External Actors:** Service Customer

**SLA Layers:** Business

**Overall priority:** Critical

This feature corresponds to the requirement that the customers must be able to register themselves in the platform, through some GUI. The registration is a prerequisite for participating in negotiation. The customers define their contact details, billing information, information about their preferences and reporting content.
3rd Parties Configuration

**Responsible WP:** A2

**External Actors:** Framework Manager

**SLA Layers:** Business, Software

**Overall priority:** Blocker

This feature allows the Framework Manager to register third party service providers in the platform. The specification of third party providers is needed to enable, at software level, the dynamic binding of abstract composed services and to enable the business manager to statically compose new services to offer as products. Requirements corresponding to this feature require that the framework allows registering for each third party provider at least the endpoints of the provided services, the associated SLATs and the SLAs already established without the support of the framework.

### 4.1.4 Service Offering

**Discovery of Services and SLAT**

**Responsible WP:** A5

**External Actors:** no

**SLA Layers:** all

**Overall priority:** Major

This feature corresponds to an internal functionality of the framework that allows all components to access, through a specific API, the list of available services and corresponding SLATs, from both internal and third party providers. This feature groups technical requirements such as the possibility to access this information using a WSDL/SOAP interface and the possibility to restrict queries to specific service providers.

**Product Management**

**Responsible WP:** A2

**External Actors:** Business Manager

**SLA Layer:** Business

**Overall priority:** Blocker

This feature is intended to manage all the information on a product during its entire lifecycle. In particular this feature included the features “Product Discovery” and “Product Definition” and adds the possibility of defining rules to conduct negotiation and to provision the service. It also includes the possibility for the provider to activate/deactivate a service for a certain customer and to
manually configure the related SLA. This feature is required by the use cases B4, B5 and B6. For instance, use case B6 needs to register the rules to use for the automatic negotiation with 3rd party providers and to manually set SLAs negotiated without the platform.

**Product Discovery**

*Responsible WP: A2*

*External Actors: Business Manager, Service Customer*

*SLA Layer: Business*

*Overall priority: Major*

This feature allows the end customer and also the provider to query for available products based on product description, categorization and other specific information from the SLAT. The requirements of this feature require to permit querying just to registered users, to allow searching for providers based on SLATs expressed by the customer, and to allow the browsing of existing services through a GUI. This feature depends on the feature “Discovery of Services and SLAT”.

**Product Definition**

*Responsible WP: A2*

*External Actors: Business Manager*

*SLA Layers: Business*

*Overall priority: Blocker*

This feature allows the Business Manager to define the offered products. Use Cases asked to be able to add/change service descriptions and related SLATs. A Product may be defined by composing other available services and related SLATs. The composed service can be just an aggregate of other services or correspond to a composition done through a specific language such as BPEL. Component services of a composed service may be delegated to third parties. The provider and SLA of component services can be statically specified on the base of rule specified at product definition time.

Product definition leverages on the feature “SLAT Definition”.

**SLAT Definition**

*Responsible WP: A5*

*External Actors: No*

*SLA Layers: all*

*Overall priority: Blocker*

This feature corresponds to the internal functionality of the framework used to register SLA Templates associated to services. Numerous requirements from use
cases constrain the content of the SLAT that this functionality must support. All use cases ask to be able to customize an SLA both with respect to the offered functionalities and with respect to the QoS. In particular all ask for the support of “classic” QoS parameters such as Availability, Cost, Reliability and for the possibility to specify: specific responsible parties for each SLA term, context dependent constraints, time dependent ranges, statistical constraints and penalties. Moreover use cases B3, B4, B5 ask for the possibility to constraint in SLAs more specific terms such as: number of computer units, data storage, compute power, period of peak hours, geographical region of data receiver, maximum budget per month. Use cases B3 and B5 ask for the support of the TMF Model terms. Other QoS parameters are instead very specific of each use case. Use cases B4, B6 ask for the possibility to express SLOs that depend on other SLOs and B5 ask for the possibility to express SLOs preconditioned by the satisfaction of third party SLAs. B6 asks for the possibility to express constraints depending on I/O data exchanged with the service. Use Cases B3, B4, B6 ask for the possibility to specify the receiver of the monitoring reports. Use Case B4 asks for the possibility to assign a priority to the terms in a SLA.

4.1.5 Service Negotiation

Business SLA Negotiation

Responsible WP: A2
External Actors: Service Customer, Business Manager
SLA Layers: Business
Overall priority: Blocker

This feature of the framework offers a Web interface and an API to the customer to select and negotiate offered products. Such interfaces allow the customer to offer SLAs that conform to the selected product SLAT. This feature includes feature “SLAT Customization” and is coordinated in an automatic way by the feature “Business SLA planning for negotiation”. The behaviour of the business negotiation is constrained by some use case specific requirements. In particular use cases B3 and B5 ask the possibility for the provider to offer to the customer a specific SLAT, selected on the base of the registered customer profile. Use Cases B4 and B6 add the precondition that a Customer must already have a pre-contract with the provider in order to participate to the negotiation (in particular in the case of automatic negotiation on the side of the provider).

Business negotiation, as with negotiation at other layers, is based on the “Automatic orchestration of customisable (re)negotiation protocol”. Moreover it depends on “Product Managements” for the configuration of negotiation behaviour.

SLAT Customization

Responsible WP: A2
External Actors: Business Manager
SLA Layers: Business
**Overall priority:** Blocker

This feature is to automatically adapt the template sent to an end customer based on his profile or segmentation. This feature is asked by use cases B3 and B5.

**Business SLA planning for negotiation**

**Responsible WP:** A2  
**External Actors:** Service Customer, Business Manager  
**SLA Layers:** Business  
**Overall priority:** Blocker

This is automatic coordination of the Business SLA Negotiation. The Business manager defines the parameters to achieve an automatic negotiation. Only in the case that the parameters cannot be filled, the manager must be allowed to use a GUI to try to achieve a manual negotiation. Such an automatism is required by use cases B4, B5, B6.

**Automatic orchestration of customisable (re)negotiation protocol**

**Responsible WP:** A5  
**External Actors:** No  
**SLA Layers:** all  
**Overall priority:** Blocker

This feature corresponds to a customisable functionality of the framework to automatically orchestrate the negotiation between provider and customer. Each layer can customise this functionality adding specific negotiation protocols. This internal functionality of the framework is constrained by several requirements. All the use cases ask for the possibility of the provider to make contra offers, for the customer to reject the offer of the provider, and for both parties to be allowed to start re-negotiation at any time. Moreover use cases B3, B4 and B5 ask that the provider always reply to a customer offer. Support for both manual and automatic negotiation is asked by B3, B5, B6. Use Cases B5 and B6 ask for customer negotiation through API and for the support of multiparty negotiation (needed for composed services with third party providers). B3 and B6 ask the support for the classic quotation process.

**Out-of-band SLA registration**

**Responsible WP:** A5  
**External Actors:** No  
**SLA Layers:** All
**Overall priority:** Blocker

This feature asked by use case B5 and B6, allows to register an SLA negotiated without the platform. Such a contract must be treated as the ones negotiated with the platform.

### 4.1.6 Service Provisioning

**Coordination of Provisioning**

**Responsible WP:** A5  
**External Actors:** Service Provider  
**SLA Layers:** all  
**Overall priority:** Blocker

This feature is based on the requirement that several layers, providers and components must be involved during provisioning of SLA. It will orchestrate the provisioning of SLAs in a recursive manner, depending on the SLA hierarchy of the involved services.

**On the fly Deploy of Monitoring**

**Responsible WP:** A5  
**External Actors:** Service Provider  
**SLA Layers:** Software, Infrastructure  
**Overall priority:** Critical

This feature corresponds to the requirement that runtime configuration of monitoring should be supported. It also requires coordination of the provisioning of all monitoring modules. Based on the Guarantee Terms in the SLAs, the involved monitoring modules must be provisioned on the fly.

**Dynamic Binding Setting**

**Responsible WP:** A3  
**External Actors:** Service Provider  
**SLA Layers:** Software  
**Overall priority:** Critical

This feature must allow the addition/replacement/removal of binding rules for invocation activities of already deployed WS BPEL processes. This is used to support the monitoring and deployment of processes independently from service providers chosen during the negotiation or execution phase of services.
Virtual Hardware Infrastructure Provisioning

Responsible WP: A4
External Actors: Infrastructure Provider
SLA Layers: Infrastructure
Overall priority: Blocker

This feature aggregates requirements about provisioning of physical and virtual low level resources, in particular compute and storage resources. The creation/deletion/migration of virtual machines must be supported, as well as the configuration of local networks. The lower level must support different virtualisation (e.g. Xen) and Cloud computing systems (OpenNebula, etc.). The provisioning must be performed on request from Provisioning coordination.

Software Provisioning

Responsible WP: A3
External Actors: Service Provider
SLA Layers: Software
Overall priority: Blocker

This feature corresponds to requirements for provisioning, deployment and configuration of software artefacts, which are required for service functionality. As the services are mostly customised, these mechanisms must be generic and extendable to support all such kinds of services.

Software Landscape

Responsible WP: A3
External Actors: Service Provider
SLA Layers: Software
Overall priority: Blocker

This feature corresponds to the requirement that each layer must support some kind of registry for all service and resource related information. These are required during the operation of the framework (negotiation, provisioning, adjustment, etc.).
4.1.7 **SLA Enforcement**

**Automated SLA Enforcement Coordination**

*Responsible WP:* A5  
*External Actors:* Service Provider  
*SLA Layers:* all  
*Overall priority:* Blocker

This feature corresponds to a requirement that the resources and sub-services must be automatically adjusted when required to ensure the SLA. Besides changing the parameters, also migration to other resources and sub-services should be supported. Those decisions are based on policies expressed in configuration.

**Virtual Hardware Infrastructure Adjustment**

*Responsible WP:* A4  
*External Actors:* Infrastructure Provider  
*SLA Layers:* Infrastructure  
*Overall priority:* Blocker

This feature implements the mechanism used by Automated SLA enforcement coordination to perform the adjustment of the Infrastructure. It is about runtime re-provisioning of compute resources to minimise SLA violations and maximise compliance with internal provider policies.

**Software Adjustment**

*Responsible WP:* A3  
*External Actors:* Service Provider  
*SLA Layers:* Software  
*Overall priority:* Blocker

This feature implements the mechanism used by Automated SLA enforcement coordination to perform the adjustment of the Software. It is about runtime reconfiguration of software resources to correct situations where SLA violations have occurred and restore the normal functioning of the system.

**Automatic Binding**

*Responsible WP:* A3  
*External Actors:* Service Provider  
*SLA Layers:* Software
Overall priority: **Major**

This feature corresponds to the requirement to support the deployment and execution of a WS-BPEL process having some service invocation not explicitly bound, but determined at runtime using associated rules. This feature depends on the feature “Dynamic Binding Setting”.

**Business Adjustment**

*Responsible WP:* A2  
*External Actors:* Business Manager  
*SLA Layers:* Business  
*Overall priority:* Blocker

This feature implements the requirements related to business aspects of SLA violations. The penalties/bonuses must be calculated and the actions must be taken (renegotiation, change the 3rd party, etc.) to optimise the business value of the SLA status and enhance the customer perception of the service.

**4.1.8 Runtime Prediction**

**Runtime Infrastructure Metrics Prediction**

*Responsible WP:* A6  
*External Actors:* no  
*SLA Layers:* Infrastructure  
*Overall priority:* Blocker

This feature allows to get warnings at run-time when infrastructure quality goals are likely to be violated in the near future. This is needed to automatically adapt resources for services based on SLA constraints.

**Runtime Software SLA Violations Prediction**

*Responsible WP:* A6  
*External Actors:* no  
*SLA Layers:* Software  
*Overall priority:* Major

This feature allows to get warnings at run-time when software service quality goals are likely to be violated in the near future. As the corresponding feature at
infrastructure level, this is needed to automatically adapt software resources for services based on SLA constraints.

4.1.9 Service Monitoring

Monitoring Coordination

*Responsible WP:* A3  
*External Actors:* Service Provider  
*SLA Layers:* all  
*Overall priority:* Blocker

This feature corresponds to general requirements related to provisioning of monitoring system. Several monitoring modules (even 3rd party) are involved in this process, so coordination is required. The aim of the monitoring system is to observe KPIs, detect undesired states, trigger automatic managements and adjustment actions and offer/report these data to all interested parties.

Virtual Hardware Infrastructure Observation and Violation Detection

*Responsible WP:* A4  
*External Actors:* Infrastructure Provider  
*SLA Layers:* Infrastructure  
*Overall priority:* Blocker

This feature corresponds to a requirement that arbitrary infrastructure related metrics (CPU, memory, response time, failures, etc.) must be monitored. Comparisons with predicted values must also be supported. The granularity of measurements must be configurable in order to detect the violations of predefined conditions at hardware infrastructure level. Those conditions are derived from SLAs and other sources and are used to support adjustment and reporting. Derived/aggregate monitoring metrics must also be supported in conditions.

Software Violation Detection

*Responsible WP:* A3  
*External Actors:* Service Provider  
*SLA Layers:* Software  
*Overall priority:* Blocker

This feature corresponds to requirements about monitoring provisioned software and detecting violations. Real-time alerting of violations of configured conditions must be supported. The supported monitoring mechanisms must include SOAP
message interception and BPEL process instrumentation. The granularity of measurements must be configurable. Aggregate monitoring metrics must also be supported.

**Monitoring Rules Extraction from SLA**

*Responsible WP: A3, A4*
*External Actors: Service Provider*
*SLA Layers: all*
*Overall priority: Blocker*

The configuration of the monitoring system is implied by the SLA. This feature requires the capability to extract rules from the agreed SLA that determines the provisioning of the monitoring system. This feature is replicated both in A3 and A4 to extract both software and infrastructure monitoring rules.

### 4.1.10 Service Reporting

**SLA Management**

*Responsible WP: A2*
*External Actors: Service Customer, Business Manager*
*SLA Layers: Business*
*Overall priority: Major*

This feature is asked by all the use cases, and allows both the customer and business manager to see and manage any contract related to the signed SLAs.

**Push/Pull Business SLA Reporting**

*Responsible WP: A2*
*External Actors: Business Manager*
*SLA Layers: Business*
*Overall priority: Blocker*

This feature generates reports in both push and pull ways. Requirements ask to communicate to the end-customer monitoring data and SLA status including violations and billing.

**SLA Status & Resource Reporting**

*Responsible WP: A5*
*External Actors: Service Provider*
**SLA Layers:** all  
**Overall priority:** Blocker

This feature is asked by all the use cases and allows any Service Provider to see the history of the status of all provided SLAs. This information should be available to all parties of the SLA. This feature also allows the manager of each layer to see the properties of each resource and their history, in particular the allocation of resources to their services. The history information is aggregated on more coarse grained timescales as we go further into the past.

**Virtual Hardware Infrastructure Reporting**

**Responsible WP:** A4  
**External Actors:** Service Provider  
**SLA Layers:** all  
**Overall priority:** Blocker

This feature requires the possibility to inform interested components of the values of measured compute infrastructure resources properties and their history. This is needed both for composing monitoring reports and more in general to provide information needed for prediction and adjustment.
5 Evaluation Approach

The evaluation process has the goal to evaluate the project results from two points of view: 1) to evaluate how well the framework developed by the scientific work packages implements the use case requirements (evaluation of scientific results) and 2) how the demonstrators developed with the developed framework provide business value to the committing stakeholders (both industries and no profit organizations).

In the next section we describe the evaluation process as it relates to its evaluation and in the following sections the metrics for the technical evaluation and the business evaluation.

5.1 Evaluation Process

Here we explain the process by which requirements are evaluated with respect to the requirements lifecycle. We divide the requirements lifecycle into four sequential phases which we denote as Specification, Design, Implementation, and Integration. We depict this process in Figure 10: Requirement Evaluation Process on page 49.

![Figure 10: Requirement Evaluation Process](image)

During the specification phase the requirements are derived from the use case scenario.

The design phase occurs after a number of use case requirements have been successfully communicated from the B-line use cases to the A-line workpackages. The preconditions of a successful communication of an individual requirement are:

1. a shared understanding of the requirement by all B-line industrial use cases in the form of written requirements within a single TRAC ticket
2. a prioritisation of the requirement by each B-line industrial use case as to the relative necessity of the fulfilment of the requirement for the development of their respective prototype
3. a specific A-line work-package has been assigned responsibility for the fulfilment of this requirement
4. this specific A-line work-package has acknowledged that it has a sufficient understanding to begin design of the software necessitated by the requirement based on the written description associated with the TRAC ticket

With these pre-conditions satisfied, the A-line work-package begins the process of designing the software by identifying one or more features that will fulfil the requirement. Since requirements are not considered in isolation but as a group there will probably be a necessarily complex inter-relationship between the set of features that satisfy the set of requirements. Accordingly, we initially assumed a many-to-many ("m:n") relationship between requirements and features. After reviewing initial plans for this proposal, however, we felt that this many-to-many relationship would be too complicated to track and would provide ambiguous information. Instead, therefore, we introduced an additional step within the consolidation phase whereby requirements would be further refined and split so that each requirement would be associated with precisely one feature. At the end of the design phase the following preconditions for the implementation phase will have been met:

1. each requirement will be identified as corresponding to one feature
2. an implementation plan for the features will exist with a planned timeline and the necessary resources
3. design artefacts exist which communicate the plan for turning the requirements into software
4. a notion of how the testability of the features should occur is specified

After these preconditions have been met, the implementation phase of the requirements occurs. Of importance to the evaluation of the requirement implementation is the testability of the features delivered for each requirement. For each feature one or more tests is created to provide a measure of the success of the software in its implementation. These tests are constructed as unit tests which mean they are amenable to inclusion in an automated suite of tests. The initial source of the tests comes from the implementing A-line work-package which specifies the functional aspects of the software. After the implementation has proceeded far enough for the architectural APIs of the software to be stable, we anticipate that the B-line use cases needing the feature will also provide unit-tests. All tests will be included in the shared source tree.

At some point in the implementation process, the feature will be declared ready for its integration into the requesting use-cases. After an suitable initial period of integration effort by the use case partner (currently thought to be about two weeks), we will ask each of the use-cases to fill out a short survey to provide a point to collect information on the evaluation of the requirement implementation that need subjective feedback.

We expect that at each phase change (i.e. from Specification to Design, or from Implementation to Integration), there will not be such a neat and cleanly definable transition as presented in the diagram. We anticipate that there will be a fair amount of iteration between the two states, as the requirement or its implementation is presented, evaluated by the receiving party, and then the reworked to satisfy the revised information.

5.2 Scientific Metrics Evaluation

Following the Goals, Questions, Metrics methodology [1], we now present the derivation of the metrics that will be used in the evaluation of these requirements from the scientific work-packages. We start by listing the high level goals we have
defined for the evaluation process in Table 2: Requirements Evaluation Goals on page 51.

<table>
<thead>
<tr>
<th>ID</th>
<th>Goal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Intelligibility</td>
<td>Each requirement should be understood by all involved parties.</td>
</tr>
<tr>
<td>G2</td>
<td>Monitoring</td>
<td>The progress towards the implementation of each requirement shall be continuously monitored.</td>
</tr>
<tr>
<td>G3</td>
<td>Traceability</td>
<td>The effects of each requirement in the creation of various artefacts during the implementation process must be traceable back to the originating requirement.</td>
</tr>
<tr>
<td>G4</td>
<td>Implementation</td>
<td>The goal of a requirement is the successful implementation as a software component that satisfies the requirement in the use case application context.</td>
</tr>
</tbody>
</table>

**Table 2: Requirements Evaluation Goals**

For each of these goals we now provide questions whose answers provide information as to the satisfaction of the goal. For each question, we provide one or more metrics to provide answers for these questions. The metrics fall into one of three categories

1. Metrics whose measurement is embedded in the requirements process based on the customized TRAC instance we denote as **Process-based** metrics. The formula for their calculation is given in the text below.

2. Metrics whose measurement is **Testing-based** on the performance of software features we detail in 5.2.1 below.

3. Metrics whose measurement is the result of the short set of questions each use case answers after a period of integration are known as **Survey-based** we provide forward references to the survey. We present the survey in 5.2.2 below.

**G1: Intelligibility**

By the goal of intelligibility we denote the need that requirements be understood by all parties involved in the process of their specification and fulfilment. Without understanding it is quite likely that requirements will not be implemented in a manner useful to the requesting party.

Q1.1: Is the requirement intelligible to all of the B-line use cases?

The B-line use cases are firmly placed in differing application domains, yet are expected to provide joint requirements. Since the requirements need to be
identified as shared between use cases where possible, we evaluate the intelligibility by other use case partners.

M1.1.1: We use the assignation of per use case priorities as an indication of the intelligibility of the requirement, since this cannot be set if a given use case has not understood the requirement enough to produce a ranking. Essentially this metric will measure the number of requirements that we discard in the process. This metric will be collected for all requirements both open and closed. The metric will consist of the requirement #(use cases that have set a priority) / 4. This rational value ranges from 0 (bad) to 1 (good).

Q1.2: Is the A-line responsibility for the implementation of the requirement understood?

M1.2.1: We use the assignation of a responsibility of per A-line work-package as the metric for whether the responsibility has been understood, as the acceptance of a requirement by an A-line work-package is a volitional step indicating that the requirement has been understood. It has is a Boolean value, with true indicating that such an assignment has been made, false indicating that it has not been made.

M1.2.2 Measured by response to survey question 1.

Q1.3 Does that A-line work-package understand the requirement enough to implement it?

M1.3.1: When the requirement has been assigned to an A-line work-package, and that work-package provides identification with a feature, this indicates an understanding of the requirement. This metric shall consist of the total number of requirements assigned to a work package divided by the total number of requirements ever entered into the system. The resulting number will be the percentage of all requirements which were successfully assigned a work package.

**G2: Monitoring**

The progress of the implementation of a requirement must be able to be continuously monitored. This is an explicit requirement in SLA@SOI description of work. By requiring this goal out of our evaluation process, we gain a tool that can assess the progress of our shared work that can assist in the identification of problems early enough for corrective action to be undertaken

Q2.1 How does the code associated with the feature stem from the requirement perform with respect to the unit tests?

M2.1.1: Measured by testing-based metrics.

Q2.2 At which stage of the implementation life-cycle is the feature which satisfies the requirement?

M2.2.1 Each requirement will be positioned at one of the four steps in the lifecycle indicating the progress towards implementation. We assign an integer for each requirement: 1 for specification, 2 for design, 3 for implementation, 4 for integration and 5 for completion. This metric shall be composed of the average of these values for all fetures.
**G3: Traceability**

The requirements should be traceable during the Specification, Design, Implementation, and Integration phases with the artefacts associated with their fulfilment. The goal of traceability assists in the comprehension of the resulting software architecture. If trade-offs between implementation choices need to be made, precisely knowing the requirements comprising the feature allows the framework of impact of these trade-offs to be assessed.

Q3.1: Is the requirement traceable to a feature?
M3.1.1: For each requirement, the result of the consolidation portion of the design phase will result in a unique feature.
This process-based metric shall consist of a Boolean value as to whether the requirement has an associated feature.

Q3.2: Which software components are necessary to implement each feature?
M3.2.1: For each feature, one or more software components will be associated with the implementation of that feature. In order to maintain the traceability back to the original requirement we require the association to be maintained.
For every identified software component we will keep track of the feature and its associated requirements. For this metric will will compute a percentage of requirements which maintain this traceability property.

**G4: Implementation**

The goal of a requirement is the successful implementation as a software component that satisfies the requirement in the requesting application context.

Q4.1: Does the implementation function as was expected in the requirement?
M4.1.1 Measured by survey question 2.
M4.1.2 Measured by success of tests.

Q4.2: Does the implementation introduce unintended complications arising from its implementation choice when integrated with the use case?
M4.2.1: Measured by survey question 3. Further explanation of these unintended complications are anticipated to be present in the free-form answers to the evaluation survey.

**5.2.1 Testing**

For the software testing part of the metrics, we initially followed the strategy we outlined in M18 of using unit tests to evaluate the convergence of the functionality of individual components to the specified requirements. Through the use of the easy integration of tests specified via the JUnit framework in the SLA@SOI Maven build system, we steadily accumulated a number of unit tests, achieving 257 such tests by the middle of M27 (July 2010), with 90% of the tests being reported as having successfully passed. The SureFire Maven plugin provided an online report visible by all developers which is updated daily by the
continuous integration environment introduced to support the overall technical support. By using the continuous availability of the reporting we were able to ensure that every component was covered by at least some amount of tests. The results of these tests are provided per tested component in section 6.

By the end of M27, it was widely felt that while the unit testing process was providing feedback on the development state of individual modules, we were missing a crucial view into the actual behaviour of the SLA@SOI as a platform. We had posited in M18 that such an integrated and validated view would be provided by the testing performed by each use case as they integrated the software into their prototypes. With the aggressive development schedule adopted after the M18 review, it quickly became apparent that feedback from the use cases would be both too little and too late to provide corrective behaviour in the component implementation. Fortunately, in the Open Reference Case demonstrator created for WP B2 we had developed three scenarios to verify the integration tests of the software. These integration scenarios covered quite sophisticated negotiation, provisioning, and runtime interactions between all of the major components, with the most complex (Negotiation) comprising over thirty discrete testable steps. For each testable step in the scenario, data was prepared that would constitute the expected result of the tests. Development resources were iteratively redistributed until each of the steps were judged to have successfully passed. While not providing as nice a measurable metric as the passing of individual unit tests, it is widely felt throughout the project that the development and maintenance of these integration tests contributed immeasurably more to the quality of the implementation than the unit tests ended up doing. For Y3, we will endeavour to examine manners in which we might successfully adopt metrics covering these scenarios (and their successors) to provide the quantitative feedback we expected from the unit tests.

5.2.2 Survey

We now provide an overview of the version of the survey which was completed by each use-case for their domain specific evaluation. The intention was to provide a light-weight mechanism to collect information for evaluation of the requirement. We initially expected to implement the survey via an online mechanism using a free tool such as SurveyMonkey\(^2\), but in the end we utilized an approach consolidating the collected information into a common spreadsheet.

We solicited feedback to the following questions as outlined in Table 3: Evaluation Survey whose results were consolidated into a spreadsheet. We planned to use integers from 1 to 5 to provide the answers to the questions ranging from “highly satisfactory” to “highly unsatisfactory”, but upon revision of the questions we decided that higher quality information would be solicited if 1) we used a simple true or false response for whether a given feature was needed by the use case, 2) we represented the answers for ‘Qualitative1’ as “insufficient”, “nearly sufficient”, “sufficient”, “good”, and “excellent”, while for the question listed as ‘Qualitative2’, we supplied the values “negligible”, “low”, “medium”, “high”, and “very high” and 4) we provided an opportunity for free form remarks about the implementation to be provided.

\(^2\) http://www.surveymonkey.com/
<table>
<thead>
<tr>
<th>Topic</th>
<th>Question</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Needed</td>
<td>Whether the feature was used in this iteration of the use case.</td>
<td>Boolean</td>
</tr>
<tr>
<td>Documentation</td>
<td>Does the documentation provided cover in a clear way the aspects needed to know related to this feature and its implementation?</td>
<td>Qualitative1</td>
</tr>
<tr>
<td>Architecture</td>
<td>Does the SLA@SOI architecture associated with this feature satisfy the functional needs for the given feature (and if the concrete Java/WSDL APIs is not suitable to your Use Case due to other constraints?)</td>
<td>Qualitative1</td>
</tr>
<tr>
<td>API</td>
<td>Do the Java/WSDL interfaces satisfy both functional and non-functional needs (and if the use case used a different implementation of the interface as provided from A-line?)</td>
<td>Qualitative1</td>
</tr>
<tr>
<td>Implementation</td>
<td>How complete is the implementation of this feature?</td>
<td>Qualitative1</td>
</tr>
<tr>
<td>Integration Complexity</td>
<td>How much complexity does integration of the feature require for your implementation</td>
<td>Qualitative2</td>
</tr>
</tbody>
</table>

**Table 3 Evaluation Survey**
6 Evaluation Results

As anticipated in section 5, several metrics has been identified to assure a good Intelligibility, Monitoring, Traceability and Implementation..

The intelligibility of features was already assured in Y1, when each use case assigned a priority to each feature and each feature was assigned to a responsible work-package, as reported in section 4.

The metrics related to the other goals have been assessed in Y2 through a survey distributed to B-Line users and through a self evaluation of A-Line providers.

The survey, measure the quality of the framework from 5 points of view (quality of Documentation, quality of Architecture, quality of API, quality of Implementation and Complexity of Integration). For each one of these aspects, five possible rating was possible (“insufficient”, “nearly sufficient”, “sufficient”, “good”, “excellent” for the quality aspects, and “negligible”, “low”, “medium”, “high”, “very high” for the integration complexity).

The A-Line has evaluated the status of the implementation, by means of both subjective judgements and automatic tests.

The next sub-sections show the results of the evaluation for each feature. In this introduction, instead we present the different evaluations aggregated by feature categories. They show the status of the results at high level and gives an interesting insight on the different perception of maturity level of the issues between A-Line and B-Line. To improve readability, all the figures adopt a same colour schema (orange for negative evaluations, blue for sufficient and green for more positive evaluations) for both the bars and the Y-axis tabs.

The Figure 11 shows the average status of each feature category as reported by the A-Line. As we can see, most of the features have a complete implementation (all tests passed), but several features lack a good integration with the rest of the framework, this is probably the main reason for the complexity experienced by the use case developers (see Figure 13).
Figure 11: Average status reported by the A-Line for each category of features.

This evaluation reported by the A-Line may be contrasted with the evaluation of this implementation performed by the B-Line and shown in Figure 12.

Figure 12: Average quality of the implementation judged by the B-Line for each category of features.
As we can see, the most at-risk features, with respect to quality of implementation, are the monitoring ones, which quality is judged nearly sufficient by almost all the use cases. This is partially justified by the fact that the implementation is not completed yet, but more analysis is needed to understand if the current designed solution is really adequate.

For other categories of feature, while the level of the implementation is equally low, the quality is judged at least sufficient. Anyway we must consider that not all features have been equally analyzed by all use cases, as the features used for the implementation of Y2 scenarios, such as monitoring, have received a more deep analysis. Moreover, for most categories of features, while the implementation is considered at least sufficient from the functional point of view, the difficulty of the integration may be very high in certain cases, as shown in Figure 13.

![Integration Complexity](image)

**Figure 13: Average level of complexity reported by the B-Line for each category of features.**

This Figure 13 confirms the difficulties on Service monitoring, which integration complexity is judged the highest, but also other features show an high complexity. In particular, the complexity is judged high also for features belonging to the categories “Framework Management”, “Design & Development” and “Runtime Prediction”.

Most of the difficulties faced by the users of the SLA@SOI Framework are strictly related to the quality of the documentation. The following Figure 14 shows the average ratings for the documentation.
Again the level of documentation for monitoring features is generally judged nearly sufficient. The same applies to Framework Management features, and this may be one of the cause of the judged complexity. In general, a better documentation could simplify the usage of the framework, without necessarily require a change to the implementation.

Very few features have a documentation judged more than sufficient (see next sections). By sure, the quality of the documentation must be enhanced in order to release the Framework as an open source product.

The Figure 15 and Figure 16 shows the rating of quality of the architecture and the API. The rating of the Architecture is generally more than sufficient, a part the monitoring features. More improvements are needed at level of APIs, in particular for the features belonging to the category “Design & Development”.

The last figure (Figure 17) shows the number of reported issues on each category of features. The considered issues does not include bugs or low level concerns (that was managed by the project development process), but just high level issues that are more directly tied to the use case requirements. No issue is reported on Pre-offering features and the level of quality is judged high also with respect to the other aspects reported in the previous figures. So this category of features may be considered satisfactory with respect to the project requirements. The highest number of issues is reported on the category Framework Management, such issues specify the main source of complexity of the framework that should be solved or mitigated during Y3.

A high number of issues is also reported on Pre-offering. Most of these are related to the management of SLATs, that is considered not sufficiently user friendly with the current implementation.
Figure 15: Average quality of the Architecture judged by the B-Line for each category of features.
Four issues are reported on Design & Development and they are related to the performance of QoS prediction, that is considered too low, and to the need of a better documentation on how to model the dependency from external third party services.

Issues on reporting concerns the possibility to have more fine-grained reports and more access to historical data.

The issues on service monitoring asks to enhance the dynamicity of configuration the virtual hardware infrastructure. Similar issues are reported on virtual hardware with regard to the Service Provisioning features, that still need a static configuration.

Service Negotiation components are not used at virtual hardware level, where a custom solution is currently used.

Concerning runtime prediction, the main issues concerns the support of more KPIs, needed in particular by use case B6.

Other reported issues mainly concern the need for a better documentation and the lack of functionalities that will be ready only in Y3.

While these issues may be considered a first set of possible requirements for Y3, a better analysis is still needed to identify other unexpressed issues. It is expected that other issues will be raised during the usage of the Framework for the implementation of Y3 scenarios. For the time being each responsible WP have already taken into account the reported issues. In the following sub-sections, for each feature the A-Line WPs have reported what are the lessons.
learned from the issues reported by the A-Line and what are the next planned steps.

More in details, the following sections are organized by category of features. For each category the evaluation of the corresponding features is reported. Each evaluation is composed by 4 sub-sections. The first one, named “Traceability” specifies what are the components of the SLA@SOI Architecture that implements the feature and what is the innovation (if any) the feature is related to. Both the information are accompanied by a short explanation, when useful. The second sub-section is named “A-Line evaluation” and reports the number of automatic tests associated to the features, the number of passed tests and the status of the implementation estimated by the developer of the feature. The third sub-section, named “B-Line Evaluation” reports the quality and complexity assessments performed by the use cases and the reported issues. Finally the last subsection reports the “Lessons learned and next steps” for that issue.

Note that as each use case has performed a different evaluation, the results reported in the following subsections are aggregated data. Distinct evaluations and issues reported by each use case are provided in Appendix D.

6.1.1 Framework Management

Framework Configuration & Setup

Traceability

**Component/s:** Full Architecture

There is no single component or small set of components that implement this feature. Instead, the different means for configuration, wiring and setup of the framework are manifold and spread over different mechanisms. The possible means are described in deliverable D.A1b [22].

**Related to innovation:** “Architecture for a consistent SLA-management framework”

The feature is closely related to innovation “Architecture for a consistent SLA-management framework”, as it allows to support a wide range of stakeholder scenarios which realize different hierarchies of SLA and service managers.

**A-Line evaluation**

**Associated tests:** 3 (the integration tests for negotiation, provisioning and runtime)

**Passed tests:** 100%

**Status:** partially completed

Though the core feature implementation is completed, there are still aspects to be further improved in order to simplify and harmonize the various configuration capabilities.
**B-Line evaluation**

**Documentation:** Nearly Sufficient

**Architecture:** Sufficient

**API:** Nearly Sufficient

**Implementation:** Sufficient

**Integration Complexity:** High

**Issues:** Runs on non supported Java Version (version 5). Not finished yet. OSGi integration very complex. The adoption of OSGi delayes the adoption of the framework.

---

**Lessons Learned and next steps**

Overall, the evaluation shows that the developed framework architecture and the concepts for its configuration are well received by the use cases and largely meet their needs. However, there are also 2 aspects which still require strengthened efforts. The documentation requires further improvement which meets also the perception of the framework developers. Second, the integration complexity is rated as rather high, which is ok taking the fact that the current architecture is evaluated in its first version.

Next steps will explicitly address these 2 issues. First, significant effort will be invested in the documentation of the framework, in terms of whitepapers and detailed specification documents – complementing the open sourcing of the framework. Second, the integration approach will be analysed, documented and simplified as much as possible. Last, we will also address further improvements on the general architecture as the various components and their capabilities evolve.

---

**Framework Model Configuration**

**Traceability**

**Component/s:** Full Architecture

There is no single component responsible for model configuration but rather the general need that all relevant models can be configured. The main models to which this applies are the SLA model (A5), the service construction model (A1), the business rule models (A2), the software landscape (A3), the infrastructure landscape (A4), and the Palladio component model (A6). Their respective configuration mechanisms are described in deliverable D.A1b [22], D.A2a [23], D.A3a [24], D.A4a [25], D.A5a [26], and D.A6a [27].

**Related to innovation:** “Architecture for a consistent SLA-management framework”

The feature is closely related to innovation “Architecture for a consistent SLA-management framework”, as it allows to support a wide range of deployment scenarios which need different meta models for their domain.
**A-Line evaluation**

*Associated tests:* 0 *(feature cannot be tested in an automatic fashion)*

*Passed tests:* N/A

*Status:* partially completed

The core feature implementation is completed. However, the actual model configuration capabilities will be further expanded.

---

**B-Line evaluation**

*Documentation:* Sufficient

*Architecture:* Sufficient

*API:* Sufficient

*Implementation:* Sufficient

*Integration Complexity:* Medium

*Issues:* No editor for SLA models. Extensions are not supported in year 2. Must be updated in Y3. Documentation sparse in different documents.

---

**Lessons Learned and next steps**

Overall, the evaluation shows the usefulness of the various information models used throughout the framework and the ways for their usage and configuration in particular use cases. However, there are still some issues to be solved around the SLA model where extensions where not fully supported and the actual usage still bears a certain complexity due to the lack of a supporting SLA editor. Some updates are already clarified as requirements for year 3 and also the documentation requires further improvement.

While the actual improvement of SLA models and their creation relate to feature “SLAT Definition” the next steps on this feature will mainly focus on documentation of the interlinking between models and the architecture and address the harmonized integration of additional model features.

---

**Framework Operation**

**Traceability**

*Component/s:* Full Architecture

*Related to innovation:* N/A

The feature is not related to any specific innovation but required as an implicit technical aspect by the use cases.
Associated tests: N/A
Passed tests: N/A
Status: partially completed
The core of the operation feature is fully completed. Missing is still the default implementation for Apache/Tomcat servers.

B-Line evaluation
Documentation: Nearly Sufficient
Architecture: Sufficient
API: Sufficient
Implementation: Sufficient
Integration Complexity: Medium
Issues: Default deployment with PAX Runner is very complex & requires internet access.

Lessons Learned and next steps
Overall, the evaluation shows that this feature is well realized. Main issues raised concern the documentation and the simplification of the run-time operations via PAX runner.
Next steps will address both of this issues, and explore alternative ways for simplified runtimes (without Pax runner).

Framework Access

Traceability
Component/s: Full Architecture

Related to innovation: N/A
The feature is not related to any specific innovation but required as an implicit technical aspect by the use cases.

A-Line evaluation
Associated tests: 0 (not applicable)
Passed tests: not applicable
Status: partially completed
The core of the access feature is fully completed. However, no explicit support is given for access from C/.Net nor for integration with DMTF CIM management models.

**B-Line evaluation**

**Documentation:** Nearly Sufficient  
**Architecture:** Sufficient  
**API:** Sufficient  
**Implementation:** Sufficient  
**Integration Complexity:** Medium  
**Issues:** Would be nice to have REST and messaging based interfaces.

**Lessons Learned and next steps**

Overall, the evaluation shows that the provided mechanisms for framework access are well realized and largely complete. However, documentation requires still some improvements, in particular realizing a more balanced description of the various access points. A specific requirement raised by one use case asks for REST APIs.

Next steps will enhance and complement the existing documentation, so that it becomes also fully comprehensive for people outside the project. The offering of REST APIs will be analysed in depth and the benefit will be weighed against the required effort.

### 6.1.2 Design & Development

**Reference Software Manageability Components and Configuration**

**Traceability**

**Component/s:** N/A  
This is an interface which should be implemented by the domain specific ManageabilityAgents to be able to integrate with the SoftwareServiceManager.

**Related to innovation:** Unified Manageability Interface and Manageability Infrastructure for Service-Oriented Systems.

**A-Line evaluation**
Associated tests: 1
Passed tests: 100%
Status: Implemented

The interface is specified and is ready for the use cases to be implemented with their custom manageability agents. Reference implementations of this interface for the DOE and for Axis2 containers is also available.

B-Line evaluation
Documentation: Sufficient
Architecture: Nearly Sufficient
API: Nearly Sufficient
Implementation: Nearly Sufficient
Integration Complexity: Low
Issues: No facility for monitoring composed services is offered

Lessons Learned and next steps
Although reference implementations for the DOE and generic Axis2 containers have been provided, designers still find it difficult to understand the steps they need to perform to implement a compliant Manageability component. Documentation will need to be extended, with a special focus on a tutorial/guide view of the reference implementations.

Service properties dependencies coding

Traceability
Component/s: Service Manager, Service Construction Model

This feature is realized by the service construction model, which specifies the relations between service types, service implementations (incl. their dependencies) and service instances. It’s heavily used in the interface of the Software Service Manager, however could be also reused for other Service Manager domains. Details are described in deliverable D.A1b [22].

Related to innovation: “Architecture for a consistent SLA-management framework”

The feature is closely related to innovation “Architecture for a consistent SLA-management framework”, as it is a core prerequisite for supporting multi-layered SLA management.

A-Line evaluation
Associated tests: 1
Passed tests: 100%
Status: Implemented
The model and the related Software Service Manager is fully implemented, is specified and is ready for the use cases to be used.

B-Line evaluation
Documentation: Good
Architecture: Sufficient
API: Sufficient
Implementation: Sufficient
Integration Complexity: Medium
Issues: It's not clear how to treat dependency from third party services

Lessons Learned and next steps
Overall, the evaluation shows that this feature is well realized and the editor supports the relatively easy creation of correct models. Documentation still requires smaller improvements, e.g. the explanation on how dependencies from third party services can be modelled.

Designing of QoS predictable systems
Traceability
Component/s: ServiceEvaluation
The ServiceEvaluation component is used by SLAManager components at each level (business, software, infrastructure) to determine a-priori evaluation of service quality parameters. ServiceEvaluation offers its functionality via the IServiceEvaluation interface. For more details, refer to Deliverable D.A6a.

Related to innovation: Software Performance and Reliability Prediction
The approach considers architectural influences on the performance and reliability of a service-based system in a comprehensive way, namely, service component implementation, service usage, quality of required services, and execution environment (physical resources). Other approaches are generally limited in their expressiveness to a subset of these factors.

A-Line evaluation
Associated tests: 6
Passed tests: 100%

Status: Implemented

While the general ServiceEvaluation component is abstract, a concrete implementation for the case of software service performance has been provided and integrated into the open reference use case scenario. In the third project year, an implementation for software service reliability will be provided.

B-Line evaluation

Documentation: Good

Architecture: Good

API: Nearly Sufficient

Implementation: Good

Integration Complexity: Low

Issues: Model-driven performance evaluation turned out to be insufficient for use case B3. Usage profiles are important with regards to the provisioning of services.

Lessons Learned and next steps

Overall, the feature is well received by the industrial use cases B3, B5, and B6, which make use of the feature. However, it turned out that in case of B3, the model-driven performance prediction approach is insufficient. Therefore, we extended the approach with measurement-based information gathering, in order to fulfil the needs of WP B3.

Manageability Design

Traceability

Component/s: The manageability modelling tool has been developed as an EMF-based Eclipse plugin. It allows the designer to specify high-level KPIs that he/she wants the system to be able to provide at runtime. The tool also allows for the automatic synthesis of instrumentation directives for deploying appropriate sensors to the executing application. The tool is an external component with respect to the overall SLA@SOI platform. More details can be found in deliverable D.A6a.

Related to innovation: We provide an explicit support of service-based systems (web service compositions), including the configuration of monitoring and control capabilities. Complete instrumentation for monitoring is supported with white-box services (i.e., instrumentation of the internal BPEL code).
**A-Line evaluation**

*Associated tests:* 1  
*Passed tests:* 100%  
*Status:* Implemented  

The implementation is complete. The tool will be extended in the project’s third year to include support for new control-related meta-models.

**B-Line evaluation**

*Documentation:* Nearly Sufficient  
*Architecture:* Sufficient  
*API:* Sufficient  
*Implementation:* Good  
*Integration Complexity:* High  
*Issues:* No issue reported.

**Lessons Learned and next steps**

Initial evaluation shows that the approach is usable but still requires effort by part of the designer who must learn how to use the tool. This is due to insufficient documentation. There is special need for tutorials and guided examples. This will be an important issue we will tackle in our ongoing efforts.

**6.1.3 Pre-offering**

**Customer registration**

**Traceability**

*Component/s:* Business Manager is the component that implements this functionality. It offers two ways for customer registration. A web service interface that can be used by external systems framework and GUI inside Business Web Tools (part of the Business Manager). It is possible to look up in A2 official deliverable: “DA2a Business SLA Management” M26.

*Related to innovation:* “Comprehensive Business Management suite for e-contracting”. This feature is part of the novel marketplace through which customers will be enabled to contract and provision services.
Associated tests: 7

Passed tests: 100%
Status: Completed

B-Line evaluation
Documentation: Sufficient
Architecture: Sufficient
API: Sufficient
Implementation: Sufficient
Integration Complexity: Low
Issues: No issue reported

Lessons Learned and next steps
This feature is completed. However, a final analysis will be done in order to detect future needs from the use cases.

3rd Parties Configuration

Traceability
Component/s: N/A
The overall architecture of the SLA@SOI framework was designed from the beginning to be able to be used in a multiprovider environment. The main impact of the inclusion of external providers would be at the business layer. Therefore, database model designed as part of A2 workpackage (see deliverable D.A2a) has been extended so to include a registry of the 3rd parties

Related to innovation: N/A

A-Line evaluation
Associated tests: 6
Passed tests: 100%
Status: Implemented
Database model designed as part of A2 workpackage (see deliverable D.A2a) has been extended so to include a registry of the 3rd parties
B-Line evaluation

Documentation: Sufficient
Architecture: Sufficient
API: Sufficient
Implementation: Sufficient
Integration Complexity: Negligible
Issues: No issue reported

Lessons Learned and next steps

Overall, the evaluation shows that this feature meets the requirements of the use cases that need a multiprovider environment, and that the integration effort is negligible. Next steps will be to collect a detailed feedback of the use cases, to further improve the implementation of the feature. In particular, the documentation will be enhanced, with a more complete description of the functionality and with detailed examples that clarify the use of the feature.

6.1.4 Service Offering

Discovery of Services and SLAT

Component/s: SLAT Registry, PSS Utilities

The feature is implemented by the SLA Template registry, which provides persistent storage for the templates, and the Publish/Subscribe System Utilities, a client to the Pub/Sub system for template advertisements. Once a template is stored in the registry, it is automatically advertised making use of the PSS utilities. Receiving parties can then store it locally and use it for discovery purposes. The process is detailed in D.A5a.

Related to innovation: SLA Template-based Service Discovery

In SLA@SOI, we assume that the use of a service is always regulated by an (implicit or explicit) SLA between service provider & customer. This SLA contains both a functional description of the offered service, together with various non-functional terms, including:

- quality of service (QoS) parameters,
- behaviours that the customer or provider contractually guarantee to the other party,
- penalties to pay in case of infringements,
- monitoring policies relating to the detection of infringements

In this context, the concept of service discovery takes on added significance. In the traditional view, a customer typically searches for a service endpoint
providing specific functional properties. In an SLA context, however, the customer searches for a provider able to supply a specific kind of service and also willing to make certain guarantees and agree to certain terms. In other words, we assume that in real business scenarios, customers search for a specific kind of SLA – rather than just a specific kind of service.

Most service registries in common use (such as UDDI), however, do not provide any standard means to represent SLA Templates, and in particular they are not capable of matching semantically equivalent SLA terms expressed in different syntactic forms. Indeed, the problem of semantically matching customer queries against advertised SLA Templates is non-trivial & is best handled by logical reasoning & constraint-satisfaction mechanisms – as opposed to simple syntactic pattern matching.

**A-Line evaluation**

**Associated tests:** 14

**Passed tests:** 100%
**Status:** Partially integrated

**B-Line evaluation**

**Documentation:** Sufficient
**Architecture:** Sufficient
**API:** Sufficient
**Implementation:** Sufficient
**Integration Complexity:** Low
**Issues:** Searching of SLATs is a future feature;

**Lessons Learned and next steps**

Overall, the B-line partners have found the feature implemented and documented sufficiently well. The fact that these are generic components that do not require customization, seems to be well-received. Next steps include the completion of template queries based on offered guarantees, and filtering of incoming advertisements based on custom preferences.

**Product Management**

**Traceability**
**Component/s:** Business Manager is the component that implements this functionality. It offers two ways for the product management. An internal API and a GUI inside Business Web Tools (part of the Business Manager). It is possible to look up in A2 official deliverable: “DA2a Business SLA Management” M26.

**Related to innovation:** “Comprehensive Business Management suit for e-contracting”. This feature is part of the novel e-services marketplace through which the commercial offer can be activated and configured.

---

**A-Line evaluation**

**Associated tests:** 3  
**Passed tests:** 100%  
**Status:** Completed

---

**B-Line evaluation**

**Documentation:** Sufficient  
**Architecture:** Sufficient  
**API:** Good  
**Implementation:** Sufficient  
**Integration Complexity:** Low  
**Issues:** No issue reported

---

**Lessons Learned and next steps**

For the next version of this component, it will be required to have a better understanding of the needs of the use cases, and to help with the integration of the components, specially with a more detailed documentation.

---

**Product Discovery**

---

**Traceability**

**Component/s:** Business Manager is the component that implements this functionality. It offers two ways for product discovery. A web service interface that can be used by external systems framework and a GUI inside Business Web Tools (part of the Business Manager). It is possible to look up in A2 official deliverable: “DA2a Business SLA Management” M26.

**Related to innovation:** “Comprehensive Business Management suit for e-contracting”. This feature is part of the novel e-services marketplace through which customers can look for the products offered in the platform.
A-Line evaluation
Associated tests: 1
Passed tests: 100%
Status: Completed

B-Line evaluation
Documentation: Sufficient
Architecture: Sufficient
API: Sufficient
Implementation: Sufficient
Integration Complexity: Low

Issues: Cannot search SLAs by their internal agreement terms, only by ID's and/or other generated metadata, which is not as useful.

Lessons Learned and next steps
It will be analyzed, together with A5 how to provide support to search by the agreement terms, and not only IDs or metadata.

Product Definition
Traceability
Component/s: Business Manager is the component that implements this functionality. It offers two ways for the product definition. An internal API and a GUI inside Business Web Tools (part of the Business Manager). It is possible to look up in A2 official deliverable: DA2a Business SLA Management M26.

Related to innovation: “Comprehensive Business Management suit for e-contracting”. This feature is part of the novel e-services marketplace through which the services provides and configure the commercial offer (products, prices, SLAs, etc.).

A-Line evaluation
Associated tests: 10 (1 specific and 9 common tests)

Passed tests: 100%
Status: Completed
OSGi integration of the web bundle with the complete framework is still pending.
**B-Line evaluation**

Documentation: Sufficient

Architecture: Sufficient

API: Sufficient

Implementation: Sufficient

Integration Complexity: Low

Issues: No issue reported

**Lessons Learned and next steps**

A better understanding of use cases requirements will be carried out. It will be implemented a smart merging of agreement terms of aggregated SLATs.

---

**SLAT Definition**

**Traceability**

**Component/s:** SLAT Registry

The SLAT Registry provides persistent storage for SLA templates. It can be queried using templates as input and yielding all compatible (matching) templates as output. Search is possible either using template content (e.g. negotiable properties), or metadata.

**Related to innovation:** SLA Model

Describing a Service Level Agreement using machine-readable formats can be a very complex task. Balancing between expressiveness equal to the natural language used for describing traditional (paper) contracts, and computability of a representation, is challenging and requires careful design. The present work, committed as part of task TA5.1, takes into account the requirements of the project’s use cases, as well as existing State of the Art, to create an extensible model that is technology-independent and suits very diverse application scenarios. Existing prior art is typically bound to specific technologies for describing the SLA (XML in most cases). This is an approach we chose to avoid, the reason being that XML is not an efficient representation for certain activities within the scope of the project, such as reasoning over templates during discovery; negotiation; conformance checking; etc. Therefore, a more flexible approach was preferred; the SLA model needs to support (XML) serialization, but not be limited by it. Similarly, most efforts are specifically targeting Web Services, while SLA@SOI wishes not to be bound by specific service-orientation technologies.
**A-Line evaluation**

Associated tests: 14  
Passed tests: 100%  
Status: Partially Integrated

**B-Line evaluation**

Documentation: Sufficient  
Architecture: Sufficient  
API: Sufficient  
Implementation: Sufficient  
Integration Complexity: High

Issues: There is no editor in place that supports creation of properly validated SLATs. Yes SLATs can be defined, but requires expert level knowledge of the SLA model. Perhaps an editor would help with this, as per B3's suggestion. Very complex and difficult to use.

**Lessons Learned and next steps**

The industrial use cases seem to be satisfied by the expressiveness of the SLA model, but on the same time find it to be very complex when creating a template. Specifically, partners seemingly had problems creating valid templates (using the XML rendering). Due to these reasons, the possibility of a graphical editor that supports the task of SLAT creation is evaluated (this is significant effort that was not foreseen by the DoW and as such requires some planning).

One more comment received was a request to allow explicit SLA dependencies as part of the model. This is a B6 requirement, due to the need for regulatory and governmental authorities to be able to monitor SLAs together. The Y3 model update will take this requirement into account.

### 6.1.5 Service Negotiation

**Business SLA Negotiation**

Traceability

Component/s: The Business SLA Negotiation functionality is covered by Business Manager and the BSLAM. The web UI supports the definition of prices, promotions, offers, etc. and the Generic SLAM interface is reused to trigger the negotiation processes. More details in A2 official deliverable: "DA2a Business SLA Management" M26. Details on generic negotiation in A5 deliverable (DA5a).
**Related to innovation:** “Dynamic Negotiation of agreements”. This feature implements the protocols and strategies for the negotiation of SLAs from the business point of view (prices, offers, discounts, penalties, etc.

**A-Line evaluation**

*Associated tests:* 7

*Passed tests:* 100%

*Status:* Partially implemented

Some functionality will be implemented during year 3, as it is explained below.

**B-Line evaluation**

*Documentation:* Sufficient

*Architecture:* Sufficient

*API:* Sufficient

*Implementation:* Sufficient

*Integration Complexity:* Low

*Issues:* B4 does this themselves by provisioning through the UI and Apache Tashi. We need the feature but we are not using the A-line component so have not rated it.

**Lessons Learned and next steps**

New requirements will be gathered from the use cases. Among others, the following functionality will be addressed: business negotiation driven by customer and provider profiles and other business information, renegotiation and full business negotiation framework.

**SLAT Customization**

**Traceability**

*Component/s:* Business Manager

Business Manager is the component that implements this functionality. It offers an internal API to deliver this functionality. It is possible to look up in A2 official deliverable: “DA2a Business SLA Management” M26.

*Related to innovation:* “Customization of Business SLA definitions taking into account the requirement of customer and service provider”. This feature provides
the intelligence to assess and adapt the definition of the SLATs offered to different customers taking into account their characteristics and preferences, using pre-defined policies.

**A-Line evaluation**

*Associated tests:* 1  
*Passed tests:* 100%  
*Status:* Completed.

**B-Line evaluation**

*Documentation:* Good  
*Architecture:* Excellent  
*API:* Excellent  
*Implementation:* Excellent  
*Integration Complexity:* Low

**Issues:** B4 does this themselves through the UI. We need the but we are not using the A-line component so have not rated it.

**Lessons Learned and next steps**

New requirements will be gathered from the use cases. Special attention will be paid in making easier the integration with the use cases.

**Business SLA planning for negotiation**

**Traceability**

*Component/s:* Business SLA Manager

Business SLA Manager is the component that implements this functionality but the behaviour is driven by the Business Manager. It offers web services to achieve a business negotiation. The web services used are the defined in generic negotiation component and the business negotiation is customized from the Business Manager. It is possible to look up in A2 official deliverable: “DA2a Business SLA Management” M26.

**Related to innovation:** “Dynamic negotiation of agreements”. Proposal and experimentation of different negotiation protocols and strategies for multiparty negotiation at the different phases of the service delivery.
A-Line evaluation

Associated tests: The tests of "Business Negotiation" issue are valid for this feature.

Passed tests:

Status: Completed

B-Line evaluation

Feature: Business SLA planning for negotiation

Documentation: Nearly Sufficient

Architecture: Good

API: Sufficient

Implementation: Good

Integration Complexity: Low

Issues: No issue reported

Lessons Learned and next steps

The documentation of this feature will be reviewed and completed. The implementation of how to specify the minimum agreement criteria for SLA negotiation is still pending.

Automatic orchestration of customisable (re)negotiation protocol

Traceability

Component/s: Protocol Engine

The Protocol Engine (PE) implements a state machine that can orchestrate different protocols (i.e. message exchanges) for the purpose of negotiation. The PE is closely tied to the Planning/Optimization Component (POC) of the SLAM it belongs to, as the latter takes the decisions while the PE takes care of negotiation mechanics. This effectively decouples stateless decision making from stateful message exchange, and allows the easy integration of different algorithms and different negotiation protocols.

Related to innovation: SLA Negotiation

Currently there is a large gap between implemented SLA solutions and scientific proposals. Almost all solutions use the above-mentioned simple protocol (one exception is BREIN which uses the FIPA Contract Net protocol in an agent-based environment), although a number of (SLA) negotiation proposals exist. Task A5.4
will evaluate those in the light of its use cases and work together with the global community towards a generic SLA negotiation standard.

**A-Line evaluation**

**Associated tests:** 9  
**Passed tests:** 2  
**Status:** Integrated  
Fully implemented, only complete testing remaining.

**B-Line evaluation**

**Documentation:** Sufficient  
**Architecture:** Sufficient  
**API:** Good  
**Implementation:** Good  
**Integration Complexity:** Low  
**Issues:** No issue reported

**Lessons Learned and next steps**

Evaluation of the feature came only from B3 and B6 use cases. According to them, the feature is sufficiently well implemented and documented.

The plan for Y3 includes the implementation of additional protocols, and more specifically, the capability to handle 1-to-many negotiations.

**Out-of-band SLA registration**

**Traceability**

**Component/s:** SLA Registry  
The SLA Registry offers persistent storage for SLAs and their state changes. It includes interfaces that may be invoked either internally within a SLAM, after a negotiation has concluded successfully, or externally for storing an already existing SLA. This feature refers to the latter. It allows that a framework manager can store a SLA that has been negotiated among humans, has been formatted according to the SLA model and some respective rendering, and can be stored after invoking the exposed interfaces.

**Related to innovation:** None.  
Does not apply.
A-Line evaluation

Associated tests: 2

Passed tests: 2

Status: Completed

Fully implemented and tested (IRegister interface for inserting and updating an SLA).

B-Line evaluation

Documentation: Sufficient

Architecture: Sufficient

API: Nearly Sufficient

Implementation: Sufficient

Integration Complexity: Medium

Issues: No issue reported

Lessons Learned and next steps

Feedback was that the feature is sufficiently well implemented and documented. Use case B5 provided a lower rating for the API of the feature; as such, in Y3 WP A5 will work with B5 to enhance the API accordingly.

The feature is fully implemented and, other than the API update, no major further development is expected to take place within Y3.

6.1.6 Service Provisioning

Coordination of Provisioning

Traceability

Component/s: Provisioning/Adjustment Component

The Provisioning/Adjustment Component (PAC) addresses the requirement to handle provisioning-time tasks separately from negotiation-time tasks. Although provisioning MAY take place at the end of a negotiation, it may well be the case that it can happen much later. For instance, someone may require a number of VMs for a few days later, but wants to establish the SLA (and therefore, reserve the resources) well in advance. Should the VMs be activated right after the negotiation ends successfully, those resources would be wasted until the service described in the SLA is supposed to become active.
In addition, there is the need to synchronize complex provisioning tasks, where the order of provisioned s/w and h/w is critical (e.g., a VM running a DB image must be available before the DB can be configured to a particular scenario). This implies a “provisioning hierarchy” that reflects, to a large extent, the service hierarchy. The PAC, and this feature in specific, is expected to address this requirement using the respective interfaces as defined and explained in D.A5a.

**Related to innovation:** N/A

**A-Line evaluation**

**Associated tests:** 11

**Passed tests:** 11
**Status:** Partially integrated

While the Provisioning and Adjustment Component is domain-specific, a generic implementation has been provided, aiming to be easily reused/adapted by the use cases. Furthermore, concrete software and infrastructure components for the provisioning functionality have been implemented and integrated into the Open Reference Case basic scenarios.

**B-Line evaluation**

**Documentation:** Sufficient

**Architecture:** Good
**API:** Sufficient
**Implementation:** Good
**Integration Complexity:** Low
**Issues:** Have used it already;

**Lessons Learned and next steps**

Overall, the evaluation shows that the software provisioning feature implemented as part of the PAC component largely meets the requirements of the use cases. In particular, the architecture, implementation and facility of integration are well received. On the other hand, documentation needs further improvement, which will be addressed as next step, in order to guarantee a successful open source release of the SLA@SOI framework.
On the fly Deploy of Monitoring

Traceability

Component/s: Monitoring Manager

The Monitoring Manager is the SLAM component that decides what is the proper monitoring framework components that must be deployed, as part of the provisioning process, in order to properly monitor an established SLA. This involves a complex process to “translate” from the SLA model to a respective event-calculus based description, which is provided as input to the monitoring system. Deliverable D.A5a provides detailed information about the produced output, the translation process and the general functionality of the Monitoring Manager.

Related to innovation: N/A

A-Line evaluation

Associated tests: 3

Passed tests: 3

Status: Partially implemented

The Monitoring Manager can process SLAs according to the SLA model, and produce monitoring configurations for software services.

B-Line evaluation

Documentation: Sufficient

Architecture: Sufficient

API: Sufficient

Implementation: Nearly Sufficient

Integration Complexity: Medium

Issues: At the moment, its static, as A4 needs to resolve issues as A5 has not received these monitoring features to date;

Lessons Learned and next steps

Partners have received the feature sufficiently well, overall. Some further work needs to be done with regard to the B4 use case, which relies heavily on A4 and for which the monitoring features (the input to the Monitoring Manager) have not been defined. Given those definitions, B4 will integrate the feature as well.
In Y3 the Monitoring Manager implementation will be extended according to SLA model changes. In addition, some further documentation will be authored to better help users of the platform use the feature to its full extent.

Dynamic Binding Setting

Traceability

Component/s: Dynamic Orchestration Engine (DOE)

This feature is implemented within the Dynamic Orchestration Engine which is a specialized form of ManageabilityAgent.

Related to innovation: Development of a New Orchestration Engine: possibility to configure different bindings for the same deployed BPEL process, on the base of specific conditions (i.e. possibility to dynamically set binding rules)

A-Line evaluation

Associated tests: 1
Passed tests: 100%
Status: Implemented

The feature is completely implemented.

B-Line evaluation

Documentation: Good

Architecture: Sufficient

API: Sufficient

Implementation: Sufficient

Integration Complexity: Low

Issues: No issue reported

Lessons Learned and next steps

Evaluation of the feature comes from B5 and B6 use case. According to them, the feature is sufficiently well implemented and documented.

To completely fulfill B6 evaluation, changes may be foreseen along the 3\textsuperscript{rd} year to set a special kind of binding rules enabling an invoke activity to call, in one step, several partners according to a multi-party conversational model instead that a bi-party one model as assumed by WS-BPEL specification.
**Virtual Hardware Infrastructure Provisioning**

**Traceability**

**Component/s:** Infrastructure Service Manager.

The Infrastructure Service Manager is the lowest level component responsible for actual provisioning of virtual infrastructure. It works very closely with the Infrastructure SLA Manager, which manages all Infrastructure SLA-level concerns. The implement Infrastructure Service Manager implements the OCCI interface. It is described in deliverable D.A4a SLA Aware Infrastructure Management.

**Related to innovation:** SLA-enabling Infrastructure Providers, Harmonized interface to access heterogeneous infrastructure resources.

Regarding the state of the art, there are no providers that offer machine readable SLAs, and there is no standard for harmonised access to heterogeneous infrastructure resources. Hence, the consortium’s attention and co-chairing of the recently drafted OCCI standard.

**A-Line evaluation**

**Associated tests:** 25

**Passed tests:** 100%

**Status:** Implemented

An initial implementation of the Infrastructure Service Manager is completed. It is built on top of the Apache TASHI provisioning system.

**B-Line evaluation**

**Documentation:** Good

**Architecture:** Good

**API:** Good

**Implementation:** Good

**Integration Complexity:** Low

**Issues:** Using this already;

**Lessons Learned and next steps**

Overall the evaluation shows that this is a much needed and important feature to many use cases. The feature itself is implemented and available. It is accessible using either a Java client library or using the OCCI standard. A repeated request for this particular feature was guidance on how to setup the infrastructure provisioning services on users' own hardware and infrastructure. Although there already exists documentation on the wiki, further clear setup documentation will
be provided to aid in such future requests. An eventual goal could also include the automated installation of the infrastructure provisioning service on a single server and/or cluster of servers.

**Software Provisioning**

**Traceability**

*Component/s:* SoftwareServiceManager

SoftwareServiceManager provides a default implementation for ORC based Software provisioning. However, to accommodate use case specific custom provisioning logic, an interface is defined which can be implemented by used cases to plug-in use case specific software provisioning logic.

*Related to innovation:* N/A

**A-Line evaluation**

*Associated tests:* 6

*Passed tests:* 100%

*Status:* Designed

**B-Line evaluation**

*Documentation:* Sufficient

*Architecture:* Sufficient

*API:* Sufficient

*Implementation:* Nearly Sufficient

*Integration Complexity:* Medium

*Issues:* Poor guidelines for customization.

**Lessons Learned and next steps**

Software Provisioning evaluation has shown that the overall architecture was sufficient to support a generic provisioning methodology which can be customized for domain specific provisioning processes. However, it was reported that guidelines to enable customization can be improved to make it a convenient process. In the future this shall be addressed.
Software Landscape

Traceability
Component/s: SoftwareLandscape

Software Landscape is implemented in the SoftwareLandscape component and is accessible through the interface exposed by the SoftwareLandscapeManager entity. SoftwareLandscape is available as part of the SoftwareServiceManager

Related to innovation: Software Landscape Modelling

The main innovation of the activity is the capability to capture model elements beyond the core entities i.e. software, service and their relationships. Landscape metamodel designed in the task enables the service providers to represent monitoring configurations, service packaging and service dependencies.

A-Line evaluation
Associated tests: 3
Passed tests: 100%
Status: Implemented

Software Landscape has been implemented and integrated into the SoftwareServiceManager component.

B-Line evaluation
Documentation: Good
Architecture: Good
API: Good
Implementation: Sufficient
Integration Complexity: Low
Issues: No issue reported

Lessons Learned and next steps

SoftwareLandscape provides consistent information in the form of landscape models which can be used during various phases of the lifecycle. Having landscapes models stored in SoftwareLandscape ensures that the service operations can be automated and performed with marginalized error. In the future, SoftwareLandscape should be aligned with the infrastructure landscape to create a federated information repository which can provide a holistic view of the entire system landscape.
6.1.7 SLA Enforcement

Automated SLA Enforcement Coordination

Traceability

Component/s: Provisioning/Adjustment and Planning/Optimization Components

The enforcement of a SLA, following its establishment and provisioning, is of great importance both to providers and to customers. Naturally, during runtime there may be circumstances that affect the service, and therefore the SLA with the agreed service parameters. Therefore, it is important to have mechanisms that can react if a monitoring event arrives that indicates a violation, or even a warning that a violation is imminent. The Provisioning/Adjustment Component (PAC) does exactly that. When something is found to be wrong, it tries to locate the exact problem, and take a corrective action e.g. by reconfiguring the service. When this is not possible, but there are additional actions that must be taken, the PAC may consult with the Planning/Optimization Component (POC), which is mostly used during the SLA negotiation. The POC will try to create a new plan and submit it to the PAC for provisioning (an original plan has been created and submitted after the SLA was established). If it is not possible to construct a new plan, re-negotiation may take place, initiated by the POC.

Related to innovation: N/A

A-Line evaluation

Associated tests: 10
Passed tests: 10
Status: Partially integrated

A generic implementation of the Provisioning and Adjustment Component has been provided, where the adjustment decisions rely on a rule engine. This facilitates the adaptation needed by the use cases. Furthermore, concrete software and infrastructure Adjustment components have been implemented and integrated into the Open Reference Case basic scenarios.

B-Line evaluation

Documentation: Sufficient
Architecture: Good
API: Sufficient
Implementation: Sufficient
Integration Complexity: Low
Issues: No issue reported
Lessons Learned and next steps

The evaluation shows that the developed Adjustment component has been useful for all the use cases. In particular, the easy integration of this component has been well evaluated. Nevertheless, several aspects receive a not so good evaluation: on one hand, the API definition and the implementation, which will be refined in order to better meet the needs of the use cases. On the other hand, the documentation, which at the moment is scattered in different formats and documents (deliverables, wiki page), will be centralized and improved with more detailed descriptions and examples.

Virtual Hardware Infrastructure Adjustment

Traceability

Component/s: Infrastructure SLA Manager - Infrastructure PAC, Infrastructure Service Manager

The Infrastructure PAC is responsible for making SLA-aware adjustment decisions based on the results from infrastructure monitoring. The Infrastructure Service Manager includes internal resource-aware optimisation. For more information please see deliverable D.A4a SLA Aware Infrastructure Management.

Related to innovation: SLA-enabling Infrastructure Providers, Predictive Compression and Decompression of Infrastructure

Although other providers allow infrastructure adjustment to be made, adjustments are typically manually driven. VMWare has recently demonstrated runtime compression and decompression of virtual infrastructure based on current load - the ability of it to respond to predictions has not yet been established. This functionality has not yet been implemented within SLA@SOI.

A-Line evaluation

Associated tests: 11

Passed tests: 100%

Status: Implemented

An initial version of the Infrastructure PAC has been implemented. It drives the SLA-aware re-provisioning of infrastructure based on alerts from the monitoring system. Resource-aware adjustment is currently being implemented. Prediction has been integrated, however adjustment based on prediction is still a work in progress.

B-Line evaluation

Documentation: Sufficient

Architecture: Good
API: Sufficient
Implementation: Sufficient
Integration Complexity: Low
Issues: Basic functions already being used. More complex ones will probably not be available till Y3;

Lessons Learned and next steps
The evaluation shows that this feature meets the requirements of the use cases. However, the integration with the use cases will get more attention in the year 3. The documentation will be extended and enhanced.

Software Adjustment

Traceability
Component/s: SoftwarePAC
it will be used in the Provisioning and Adjustment Components inside the SLA Managers. Being the PAC domain specific, there will be several implementations: reference implementations focused on the Open Reference Case (ORC), both software and infrastructure, as well as implementations for the industrial use cases.

Related to innovation: N/A

A-Line evaluation
Associated tests: 5
Passed tests: 100%
Status: Partially integrated

B-Line evaluation
Documentation: Sufficient
Architecture: Good
API: Good
Implementation: Good
Integration Complexity: Low
Issues: No issue reported
Lessons Learned and next steps

Overall, the evaluation shows that the Software Adjustment component developed within A3 workpackage has been well considered for the use cases. In particular, the architecture, API, implementation and the easy integration of this component has been well evaluated. Nevertheless the documentation is considered to be the weakest point of the feature. Significant effort will be invested in specific documentation of the component, which can be addressed as part of the open sourcing of the framework.

Automatic Binding

Traceability

Component/s: Dynamic Orchestration Engine (DOE)

This features is implemented within the Dynamic Orchestration Engine which is a specialized form of ManageabilityAgent.

Related to innovation: Development of a New Orchestration Engine: to execute abstract processes described by using standard WS-BPEL language for process description and a decoupled SLAT based representation for describing the component services

A-Line evaluation

Associated tests: 0
Passed tests: 0%
Status: Designed

B-Line evaluation

Documentation: Good
Architecture: Sufficient
API: Sufficient
Implementation: Sufficient
Integration Complexity: Low
Issues: No issue reported

Lessons Learned and next steps
The evaluation shows that the designed feature meet the requirements of B5 and B6 use cases. Implementation is planned for 3rd year.

**Business Adjustment**

**Traceability**

**Component/s:** BusinessPAC

This feature is implemented as part of the Business Manager component, and takes care of the economical impact of the service performance. It receives information of the violations produced at lower levels (softwared and infrastructure), calculates the penalties to be paid based on the SLA, and triggers actions aim to reduce the cost due to the malfunctioning of the service.


**Related to innovation:** Business SLA post-sale management

**A-Line evaluation**

**Associated tests:** 6

**Passed tests:** 100%

**Status:** Implemented

Business adjustment feature is implemented in the violationPenalty component within the Business Manager. It uses the information coming from the lower level PACs through the <<control/track>> interaction, and using the information in the database. This component is used by the ORC scenario and some of the use cases.

**B-Line evaluation**

**Documentation:** Sufficient

**Architecture:** Good

**API:** Good

**Implementation:** Sufficient

**Integration Complexity:** Low

**Issues:** No issue reported

**Lessons Learned and next steps**
The evaluation shows that the Business Adjustment component has been useful for all the use cases. In particular, the easy integration of this component has been well evaluated. Nevertheless, two aspects need to be improved: on one hand, the implementation, which will be refined in order to better meet the needs of the use cases. On the other hand the documentation, where significant effort will be invested, adding examples and more detailed descriptions.

### 6.1.8 Runtime Prediction

#### Runtime Infrastructure Metrics Prediction

**Traceability**

*Component/s:* PredictionClient, PredictionService

This feature is implemented as a couple independent services that can be deployed according to the specific requirements of the use case. PredictionClient gathers local instrumentation metrics and provide computed predictive values over those metrics. The computation of prediction is done using different techniques following approaches such as heuristics and/or machine learning, according to requirements. As prediction is a resource intensive operation, the computational part of prediction can be offloaded to a dedicated PredictionService(s) nodes.

PredictionClient and PredictionService implement a message passing actor model to exchange data and provision prediction computations. PredictionService is also configurable to be able to offer interfaces for the consumption of prediction data.

*Related to innovation:* Predictable Systems Engineering

**A-Line evaluation**

*Associated tests:* 4

*Passed tests:* 100%
*Status:* Implemented

**B-Line evaluation**

*Documentation:* Sufficient

*Architecture:* Good

*API:* Sufficient

*Implementation:* Good

*Integration Complexity:* Medium
**Issues:** Gathering prediction data in year 2 in ganglia/collectl. Already used in for demo purposes.;

*Lessons Learned and next steps*

Infrastructure Runtime Prediction can be improved in several ways. First of all, the current implementation does not allow the dynamic configuration of prediction algorithms which should be chosen at deployment time. Also for Machine Learning based prediction implementations the window of time and data re-sampling that is used for training the prediction models is fixed at deployment time and it could be made dynamic and configurable also during runtime. One area also of improvement for runtime prediction is to integrate the the underlying implementation to standardised scalable backends such as Hadoop/Mahout.

*Runtime Software SLA Violations Prediction*

**Traceability**

*Component/s:* Software Level Prediction System (SLPS)

The SLPS is part of the Software Level Monitoring System (SLMS). Building element of the SLMS is the Reasoning Component. A reasoning component (aka Reasoning Component Gateway (RCG)) can perform monitoring, prediction, or it can combine both monitoring and prediction.

A RCG receives a reasoning component configuration (that is part of a monitoring system configuration) and translate it into monitoring and/or prediction specification. The SLMS is described in deliverable D.A3a whilst the prediction system is described in deliverable D.A6a.

**Related to innovation:**

Three limitations of existing techniques that make them falling short of providing adequate support for run-time prediction of SLA violations are:

- they tend to focus on system infrastructure properties (e.g., network and server properties) rather than service level application based properties (e.g., service throughput, mean time to failure).
- they tend to focus on the prediction of specific types of properties without providing a more generic framework for building predictors that can cover a wide or even the whole spectrum of service properties that can be part of an SLA.
- They are not integrated with environments for monitoring SLAs for service-based systems.

We addresses the above key issues by realising a framework that provides an integrated architecture for SLA monitoring and prediction that is capable of supporting the latter activity through the deployment of a built-in set of model-based predictors. The framework doesn't focus specific QoS properties; on contrary it provides the infrastructure for creating used-defined QoS predictors.
A-Line evaluation

Associated tests: 0
Passed tests: N/A
Status: Implemented

B-Line evaluation

Documentation: Sufficient
Architecture: Sufficient
API: Sufficient
Implementation: Sufficient
Integration Complexity: Medium

Issues: Currently some KPIs used by B6 (e.g. "number of calls with a certain response time", cannot be predicted).

Lessons Learned and next steps

We plan to extend the set of available “built-in” QoS predictors and automatically inferred models to create a robust and flexible support for QoS monitoring and prediction.

6.1.9 Service Monitoring

Monitoring Coordination

Traceability

Component/s: MonitoringManager

The MonitoringManager (MM) coordinates the generation of a monitoring configuration of the system, given a set of Agreement and Guarantee Terms in an SLA model. A monitoring configuration describes which components to configure and how their configurations can be used to obtain events from service monitoring. See deliverable D.A3a (SLA Aware Service Management) for more information. The monitoring configuration supports the monitoring agents in the monitoring architecture by expressing output ids of events for reasoning about guaranteed states.

Related to innovation: Advanced SOA Management

A-Line evaluation
**Associated tests:** 3 (incremental versions of the A4 SLA Template for testing)

**Passed tests:** 100%

**Status:** Implemented

**B-Line evaluation**

**Documentation:** Sufficient

**Architecture:** Sufficient

**API:** Sufficient

**Implementation:** Sufficient

**Integration Complexity:** Medium

**Issues:** No issue reported

**Lessons Learned and next steps**

The MonitoringManager component has been successfully tested to extract, parse and configure monitoring components from terms expressed as Guaranteed States in the current examples of SLA templates and models. As part of this test we have generated example Monitoring System Configurations which are used by monitoring agents to instantiate and respond to monitoring events in the monitoring system. We will seek to develop more complex configurations representing coordinating guaranteed actions.

**Virtual Hardware Infrastructure Observation and Violation Detection**

**Traceability**

**Component/s:** Low Level Monitoring System (LLMS).

The Low Level Monitoring System receives monitoring configuration information, deploys the appropriate monitors and gathers, analyses and propagates violations and warnings. The LLMS is described in Deliverable D.A4a SLA Aware Infrastructure Management.

**Related to innovation:** Standardisation of Infrastructure Monitoring Interfaces, Self-Provisioned Distributed Infrastructure Monitoring.

**A-Line evaluation**

**Associated tests:** 11

**Passed tests:** 100%

**Status:** Implemented.
An initial Low Level Monitoring System has been implemented and deployed on top of Apache Tashi. It can send monitoring events to the Infrastructure Provisioning Adjustment Component (IPAC).

**B-Line evaluation**

**Documentation:** Nearly Sufficient  
**Architecture:** Nearly Sufficient  
**API:** Sufficient  
**Implementation:** Nearly Sufficient  
**Integration Complexity:** High  
**Issues:** Some work is complete on this, but it requires pre-configured values rather than dynamic ones. Ownership of this component within the overall project is not clear to B4.

**Lessons Learned and next steps**

This feature is essential not only for infrastructure based use cases, but always when service is deployed as a set of virtual appliances. We have learned that a instrumentation should focus on available virtual hardware resources (cpu speed, memory size, disk size, net throughput) would be sufficient for most purposes, as these are the parameters that service user understands and should be guaranteed in SLA.

**Software Violation Detection**

**Traceability**

**Component/s:** RCG, EVEREST  

**Related to innovation:** Supporting detection of violations of SLAs expressed in SLA(T) at the software level.

The developed RCG supports the transformation of SLA(T) onto EC-Assertion. Hence, it provides a possible formal foundation for SLA(T) itself since EC-Assertion is a formal version of Event Calculus (first order temporal logic language).

The implementation of EVEREST has also been extended to provide support for more complex data aggregation functions and basic relational conditions, as well as the use of time expressions in fluent related predicates.

**A-Line evaluation**

**Associated tests:** 6
Passed tests: 50%

Status: implemented

75 percent completion of implementation/integration. The components have been compiled with the rest of the SLA@SOI framework in Maven but its availability as an OSCGI service has not been fully tested yet.

B-Line evaluation

Documentation: Nearly Sufficient

Architecture: Nearly Sufficient

API: Nearly Sufficient

Implementation: Nearly Sufficient

Integration Complexity: High

Issues: No issue reported

Lessons Learned and next steps

The implementation of EVEREST has also been extended to provide support for more complex data aggregation functions. Future steps are to further extend RCG EVEREST functionalities to support, besides guaranteed states already supported, also guaranteed action.

Software monitoring rules extraction from SLA

Traceability

Component/s: MonitoringManager

The MonitoringManager (MM) coordinates the generation of a monitoring configuration of the system, given a set of Agreement and Guarantee Terms in an SLA model. It decides, for an SLA model instance it receives, which is the most appropriate monitoring configuration according to a monitoring component selection criteria. A monitoring configuration describes which components to configure and how their configurations can be used to obtain events from service monitoring. See deliverable D.A3a (SLA Aware Service Management) for more information.

Related to innovation: Advanced SOA Management

A-Line evaluation

Associated tests: 3 (incremental versions of the A4 SLA Template for testing)

Passed tests: 100%
**Status:** Implemented.

**B-Line evaluation**

**Documentation:** Nearly Sufficient

**Architecture:** Nearly Sufficient

**API:** Nearly Sufficient

**Implementation:** Nearly Sufficient

**Integration Complexity:** Medium

**Issues:** No issue reported

**Lessons Learned and next steps**

The MonitoringManager component has been successfully tested to extract, parse and configure monitoring components from terms expressed as Guarantee States in the current examples of SLA templates and models. The selection process of these components is currently limited to a simple "first in list" basis (if more than one type of component is selected as a candidate for selection). This will be improved to perform some complex selection based upon preferential criteria (potentially expressed in the SLA itself, e.g. cost optimisation, provider-based optimisation etc).

**Virtual Hardware Monitoring Rules Extraction from SLA**

**Traceability**

**Component/s:** SLA Level Monitoring System

The SLA Level Monitoring System receives SLAs and extracts monitoring configuration information to evaluate which QoS terms should be analysed. See Deliverable D.A4a SLA Aware Infrastructure Management for more information.

**Related to innovation:** Self-Provisioned Distributed Infrastructure Monitoring.

**A-Line evaluation**

**Associated tests:** 11

**Passed tests:** 100%

**Status:** Implemented

**B-Line evaluation**

**Documentation:** Nearly Sufficient
Lessons Learned and next steps

The Infrastructure SLA in Y2 is based on a predefined SLA Template defining just the parameters of virtual infrastructure. Rule extraction is therefore pretty basic, however still necessary.

6.1.10 Service Reporting

SLA Management

Traceability

Component/s: Business Manager

This feature makes available all the administrative information related with the lifecycle of a given SLA. In particular, the detailed information of the content of the Agreement, any extension and revision to it, and the history of its performance should be persistently stored and accessible to both the service provider and the customer.

It is possible to look up in A2 official deliverable: “DA2a Business SLA Management” M26

Related to innovation: Business SLA post-sale management.

A-Line evaluation

Associated tests: 12
Passed tests: 100%
Status: Implemented

B-Line evaluation

Documentation: Sufficient
Architecture: Good
API: Sufficient

Implementation: Sufficient

Integration Complexity: Low

Issues: Unclear how to access the history of SLAs

**Lessons Learned and next steps**

The mean evaluation results show that the feature is adequate to be used by the use cases. Nevertheless, the results from one of the use cases are significantly lower than the average values. Therefore, a closer interaction with the leaders of the B-line workpackages is needed, in particular with this use case, to understand the specific requirements they may have regarding the specification, implementation and documentation of this functionality.

**Push/Pull Business SLA Reporting**

**Traceability**

Component/s: Reporting

This feature allows the service provider to generate reports about the administrative information of an SLA, including the description of the purchased service, validity dates and the conditions of the agreement, as well as information about the performance of the service, in terms of violations and penalties occurred. The generated report will be periodically sent to the customer, and also an interface will be offered so the customer can request the business information on demand.

The Reporting component of the Business Manager is the one that implements this functionality, and offers its functionality through different interfaces: an internal API, web services and a GUI inside Business Web Tools (part of the Business Manager). It is possible to look up in A2 official deliverable: “DA2a Business SLA Management” M26.

**Related to innovation:** Business SLA post-sale management

**A-Line evaluation**

Associated tests: 9

Passed tests: 100%

Status: Implemented

**B-Line evaluation**

Documentation: Sufficient

Architecture: Good
**API:** Sufficient

**Implementation:** Sufficient

**Integration Complexity:** Low

**Issues:** Reporting at the level of single guarantee term should be supported.

---

**Lessons Learned and next steps**

SLA Reporting fulfils the more basic requirements of the use cases for the Y2. Significant improvements should be included in this feature, in order to allow the dynamic configuration of the content of the reports. Also different methods to deliver the reports should be considered, apart from the already implemented webservices (for the push interface) and the email (for the pull one).

---

**SLA Status Reporting**

**Traceability**

**Component/s:** PAC, SLA Registry and SLA Management Console

During runtime, as SLAs are being established, provisioned, monitored, renegotiated and eventually expiring or terminated, their status is persistently stored in the SLA Registry, after the PAC receives it from the monitoring system and forwards it accordingly. The information on each SLA includes how exactly it relates to other SLAs, i.e. the dependencies among them. Based on this information, it is possible e.g. for the PAC to address violations taking into account dependent SLAs. Status information is also necessary to see how SLAs are performing, and extract conclusions about the resources/infrastructure used – which may be very useful later on for reasons of capacity management.

The information within the SLA Registry is also used by the SLA Management Console, for visualization purposes. It is therefore possible for a framework manager to see how many SLAs are active, their content, their status, etc, and filter them based on various criteria to get an overview of the whole SLA ecosystem within the framework he/she manages.

**Related to innovation:** N/A

N/A

---

**A-Line evaluation**

**Associated tests:** 15

**Passed tests:** 14]

**Status:** Completed
Respective PAC and SLA registry tests complete; one test failing in the SLA Management Console due to recent changes in the SLA registry. It is expected to successfully complete this test before the end of Y2.

**B-Line evaluation**
*
**Documentation:** Sufficient

**Architecture:** Good

**API:** Good

**Implementation:** Good

**Integration Complexity:** Low

**Issues:** No issue reported

**Lessons Learned and next steps**
The feature has been evaluated by use cases B3 and B6. The feedback was that overall implementation is sufficient; while documentation of the feature would benefit from being extended. This is something scheduled for Y3, in parallel with extensive modifications to the console so that it can handle arbitrary SLAs as well as possible, and display them intuitively. To that extent, a number of options from disciplines such as Human-Computer Interaction (HCI) are currently evaluated.

**Virtual Hardware Infrastructure Reporting**

**Traceability**

**Component/s:** Low Level Monitoring System (LLMS).

The Low Level Monitoring System receives monitoring configuration information, deploys the appropriate monitors, and gathers, analyses and propagates violations and warnings. It also stores monitoring information in an historical database. The LLMS is described in Deliverable D.A4a SLA Aware Infrastructure Management.

**Related to innovation:** Standardisation of Infrastructure Monitoring Interfaces, Self-Provisioned Distributed Infrastructure Monitoring.

**A-Line evaluation**

**Associated tests:** 11

**Passed tests:** 100%

**Status:** Implemented.
The historical database stores monitoring data in a structured way, and an API is provided to allow this data be queried as needed by the use case.

**B-Line evaluation**

**Documentation:** Nearly Sufficient  
**Architecture:** Sufficient  
**API:** Sufficient  
**Implementation:** Sufficient  
**Integration Complexity:** Low  
**Issues:** Historical DB is not fully integrated yet.

**Lessons Learned and next steps**

Infrastructure Reporting is important for all use cases were the service user is aware of the infrastructure. The reporting should offer in insight into history in trend of the metrics used in SLA guarantees. As the GUI is use case specific, this feature provides just a common data access to observed metrics.
7 Conclusions

7.1 Summary

During Y1 we collected SLA@SOI framework requirements. The result of such a process consists of 42 features organized in 10 categories.

In Y2 we performed a detailed evaluation both at level of single features and at level of feature categories. The evaluation has been performed by both the A-Line and B-Line. We found that while there is some significant difference in the perception of the quality of the feature implementation between the A-Line and B-Line and not all features have been the same level of experimentation, the judgments of B-Line are generally aligned with the level of completeness of the implementation reported by the A-Line.

Not all the features have reached the required level of maturity, therefore the use cases have used some ad-hoc solution for features not yet completed and that was needed in Y2, for example to avoid complications due to incomplete support of OSGi. A part technical issued to be solved at design level, a more clear and complete documentation, possible accompanied by tutorials, is required not only for the development of use cases, but also to release the Framework as open source.

7.2 Outlook on Future Work

There are important issues already identified that needs to be solved or mitigated during Y3, for example on the monitoring features, but a better analysis is still required in order to identify the most important issues to be solved with the limited resources still available in Y3. Each use case have done a concrete experience of features required for Y2, while other features have been analysed just at design level or through tests code. Some more issue is expected to be raised in the next months and a deeper analysis of the currently reported issues is needed to plan the Y3 development.

Some of the identified issue will be converted in formal requirements to be implemented in Y3. While the quality is generally sufficient, there is room for several improvements of the provided APIs, so it is expected to express most requirements directly at the level of new API characteristics.

Some hint for the prioritization of Y3 requirements is expected to come from the SLA@SOI ICT event at the end of September 2010. As described in section 3, a dedicated survey will be distributed to the participants to the event to help to identify more important issues from the point of view of both the academic and industrial communities.

The formalized Y3 requirements and the evaluation of their implementation will be reported in the third version of this deliverable.
References


[21] SLA@SOI project: *Deliverable D.A1a Framework architecture – full lifecycle.* July 2010

[22] SLA@SOI project: *Deliverable D.A1b Integrated Framework – full lifecycle.* July 2010

[23] SLA@SOI project: *Deliverable D.A2a Business SLA management.* July 2010

[24] SLA@SOI project: *Deliverable D.A3a SLA-aware service management.* July 2010

[25] SLA@SOI project: *Deliverable D.A4a SLA-aware infrastructure management.* July 2010

[26] SLA@SOI project: *Deliverable D.A5a Foundations for SLA-management.* July 2010

[27] SLA@SOI project: *Deliverable D.A6a Predictable/Manageable Service Engineering Methodology and Prediction Services.* July 2010

[28] Deliverable D.B1b - Industrial Assessment Report
Appendix A: Glossary

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

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreement Initiator</td>
<td>An agreement initiator is a party to a service level agreement. The initiator creates and manages an agreement on the availability of a service on behalf of either the service customer or service provider, depending on the domain-specific signalling requirements.</td>
</tr>
<tr>
<td>Agreement Offer</td>
<td>An offer is the description of the agreement relationship that is sent from agreement initiator to agreement responder during agreement creation, indicating the relationship which the initiator would like to form.</td>
</tr>
<tr>
<td>Agreement Responder</td>
<td>The agreement responder is a party to a service level agreement. The responder implements and exposes an agreement on behalf of either the service provider or service customer, depending on the domain-specific signalling requirements.</td>
</tr>
<tr>
<td>Agreement Template</td>
<td>An agreement template is an XML document used by the agreement responder to advertise the types of offers it is willing to accept.</td>
</tr>
<tr>
<td>Agreement Term</td>
<td>Agreement terms define the content of a service level agreement.</td>
</tr>
<tr>
<td>Business Service</td>
<td>A business service is exposed/invoked via at least some non IT elements.</td>
</tr>
<tr>
<td>Business Manager</td>
<td>A specialization of service provider: person that defines the SLATs of products and joins available services in a product.</td>
</tr>
<tr>
<td>External Service</td>
<td>External services are exposed across the boundaries of an organization, i.e. across at least two administrative domains.</td>
</tr>
<tr>
<td>Feature*</td>
<td>Functionality that a specific software component of the SLA@SOI framework should implement to satisfy one or more requirements.</td>
</tr>
<tr>
<td>Framework Administrator</td>
<td>A specialization of service provider: person that configures/adapts the SLA@SOI framework for a specific application.</td>
</tr>
<tr>
<td>Guarantee Term</td>
<td>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.</td>
</tr>
<tr>
<td>Hybrid Service</td>
<td>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).</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Infrastructure Manager</td>
<td>A specialization of infrastructure provider: person/system that is interested to measure and control infrastructure properties.</td>
</tr>
<tr>
<td>Infrastructure Provider</td>
<td>A specific kind of service provider that focuses on the provisioning of infrastructure services.</td>
</tr>
<tr>
<td>Infrastructure Service</td>
<td>An infrastructure service is a specific IT service which exposes resource/hardware-centric capabilities.</td>
</tr>
<tr>
<td>Internal Service</td>
<td>Internal services are exposed within the boundaries of an organization, i.e. within one administrative domain.</td>
</tr>
<tr>
<td>IT Service</td>
<td>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.</td>
</tr>
<tr>
<td>NGOSS Shared Information and Data model*</td>
<td>A set of comprehensive standardized information definitions, developed by TMF, acting as the common language for building easy to integrate OSS (Operational Support System) and BSS (Business Support System) solutions.</td>
</tr>
<tr>
<td>Offered Service</td>
<td>An abstract service (more precisely: service type) which is offered by a specific Service Provider to its Service Customers.</td>
</tr>
<tr>
<td>Operation Level Agreements</td>
<td>A specification of the conditions under which an internal service or a component is to be used by its “customer”.</td>
</tr>
<tr>
<td>Requirement*</td>
<td>Functional or not functional characteristics of the SLA@SOI Framework desired from potential adopter (service providers).</td>
</tr>
<tr>
<td>Research Topic*</td>
<td>Subject of investigation of a scientific work package (action line A) that should produce specific project results.</td>
</tr>
<tr>
<td>Service</td>
<td>A means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks. See also service interface type, service concreteness, service exposure</td>
</tr>
<tr>
<td>Service Concreteness</td>
<td>The stage a service reaches over time from a fully abstract type to actually instantiated. See also service type, offered service, service implementation, service instance</td>
</tr>
<tr>
<td>Service Consumer</td>
<td>Person(s) who actually consume/use the provided services. Typically they belong to the service customer.</td>
</tr>
<tr>
<td>Service Customer</td>
<td>Someone (person or group) who orders/buys services and defines and agrees the service level targets.</td>
</tr>
<tr>
<td>Service Description Term</td>
<td>Service Description Terms describe the functionality that will be delivered under the service level agreement. The agreement description may include also other non-functional items referring to the service description terms.</td>
</tr>
<tr>
<td>Service Exposure</td>
<td>Services can be exposed either internally (within the same administrative domain) or externally. See also internal service, external service</td>
</tr>
<tr>
<td>Service Implementation</td>
<td>A service implementation is a possible concrete realization of a given service type.</td>
</tr>
<tr>
<td>Service Instance</td>
<td>A concrete realization of an offered service which is ready for consumption by service users. It relies on the instantiations of all the resources required for a given service implementation.</td>
</tr>
<tr>
<td><strong>Service Interface Type</strong></td>
<td>Describes the nature of an actually exposed service, i.e. about the nature of his invocation interface.</td>
</tr>
<tr>
<td><strong>Service Level Consequence</strong></td>
<td>An action that takes place in the event that a service level objective is not met.</td>
</tr>
<tr>
<td><strong>Service Level Agreement</strong></td>
<td>An agreement defines a dynamically-established and dynamically managed relationship between parties. The object of this relationship is the delivery of a service by one of the parties within the context of the agreement. The management of this delivery is achieved by agreeing on the respective roles, rights and obligations of the parties. The agreement may specify not only functional properties for identification or creation of the service, but also non-functional properties of the service such as performance or availability. Entities can dynamically establish and manage agreements via Web service interfaces.</td>
</tr>
<tr>
<td><strong>Service Level Objective</strong></td>
<td>Service Level Objective represents the quality of service aspect of the agreement. Syntactically, it is an assertion over the agreement terms of the agreement as well as such qualities as date and time.</td>
</tr>
<tr>
<td><strong>Service Provider</strong></td>
<td>An organization supplying services to one or more internal customers or external customers.</td>
</tr>
<tr>
<td><strong>SLA Manager</strong></td>
<td>A specialization of service provider: person/system that is responsible for managing SLATs and SLA relationships.</td>
</tr>
<tr>
<td><strong>Software Designer</strong></td>
<td>A specialization of software provider: person that designs/develops the architecture and components of a specific SLA based application.</td>
</tr>
<tr>
<td><strong>Software Manager</strong></td>
<td>A specialization of service provider: person that defines software-based services, takes care of their management and supports the SLA manager in creating appropriate SLA templates.</td>
</tr>
<tr>
<td><strong>Software Provider</strong></td>
<td>An organization producing software components which might be used by a service provider to assemble actual services.</td>
</tr>
<tr>
<td><strong>Software Service</strong></td>
<td>A software service is a specific IT service which is exposed/invoked by means of software entities such as Web services, user interfaces, or software-based business processes.</td>
</tr>
<tr>
<td><strong>Software Component</strong></td>
<td>Software components are the entities produced at design-time by a software provider.</td>
</tr>
<tr>
<td><strong>Service Type</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>TeleManagement Forum</strong></td>
<td>An international non-profit industry association focused on improving business effectiveness for service providers and their suppliers. The forum provides practical solutions, guidance and leadership to improve the way that digital services are created, delivered and charged.</td>
</tr>
<tr>
<td><strong>WS-Agreement</strong></td>
<td>A standardised specification for the establishment of SLAs between initiators and responders.</td>
</tr>
</tbody>
</table>
Appendix B: Abbreviations

AOP  Aspect Oriented Programming
BM   Business Manager
B-SLAM Business SLA Manager
EMF  Eclipse Modelling Framework
ERP  Enterprise Resource Planning
IE   Interaction Event
FCR  Finite capacity regions
Infr-SLAM Infrastructure SLA Manager
Infr-SM Infrastructure Service Manager
IoC  Inversion of Control
KPI  Key Performance Indicator
LLMS Low Level Monitoring System
LQN  Layered Queueing Networks
MA   Manageability Agent
MRE  Monitoring Result Event
MVC  Model View Controller
NFP  Non-functional property
ORC  Open Reference Case
OVF  Open Virtualization Format
QoS  Quality of Service
QPN  Queueing Petri Nets
PAC  Provisioning and Adjustment Component
POC  Planning and Optimization Component
POJO Plain Old Java Objects
SaaS Software as a Service
SE   Service Evaluation
SID* NGOSS Shared Information and Data Model
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
SotA State of the Art
SW-SLAM Software SLA Manager
SW-SM Software Service Manager
TCO  Total Cost of Ownership
TMF* TeleManagement Forum
TOGAF The Open Group Architecture Framework
Appendix C: Status Definitions used in TRAC

The following status values have been used in Trac, which captures the current state of each requirement:

- **proposed**: In this state are all requirements, when they are put into the TRAC-system at the first time or when a change to them is proposed. (→ initial status set by the responsible B-line WP partner if a requirement is new or changed).

- **issues**: In this state there is more information needed for A-line to continue the validation process. If an extensive reconciliation between A-line and B-line will be necessary this should be done via wiki. In this case a remark in the TRAC (e.g. comments field) must note that there are additional statements in wiki. This remark in the TRAC have to be done by the responsible A-line. (→ status set by A-line).

- **rejected**: After evaluation from A-line the requirement is set to “rejected” if an implementation is not possible or the realisation makes no sense from the perspective of the A-line. It could be also set to rejected if the addressed A-line WP is not responsible for it (the requirement has no influence to the addressed WP). (→ set by A-line).

- **accepted**: Status “accepted”: fill A-Line accepted the requirement proposed by B-Line. (→ set by A-line).

- **checked**: After examination of the requirement from B-Line the priorities of it have to be set and the status has to be changed to “checked”. The status “checked” indicates that the requirement is incorporated in a feature and ready to be analysed by A-Line. This status will be set by B1 (TA) based on the TRAC report results. (→ set by B1 TA).

- **designed**: fill A-Line has defined components & interfaces that implement the requirement. After acceptance of the proposed architecture by B-line, the status of the requirement is set to designed. (→ set by A-line)

- **implemented**: fill A-Line has completed the implementation of the requirement (→ set by A-line).

- **cancelled**: This could be, if a requirement has been rejected from A-line and the action lines agree not to implement it, or B-line renounced to ask the requirement (all priorities set to zero) (→ set by B1 TA after check with B1 leader)

- **tested**: fill A-Line has completed the tests of the feature associated to the requirement (→ set by A-line).

- **approved**: After finalized tests of the implementation of a requirement from A-line the result has to be proved from B-line. If the result is accepted from B-line the status will be set to “approved” (→ set by B-line).

- **failed**: After finalized tests of the implementation of a requirement from A-line the result has to be proved from B-line. If the result is not accepted from B-line the status will be set to “failed” (→ set by B-line).
Appendix D: Technical evaluation of Use Cases

The following tables report the technical evaluation of each Framework feature provided by each use case. Each use case was requested to perform a complete evaluation of at least the features used in its scenario implemented in Y2. An un-complete evaluation was allowed for the other features. No evaluation was required for features used by the use case neither in Y2 nor in Y3.

Each feature was evaluated from 5 points of view (Documentation, Architecture, API, Implementation, Integration Complexity). In a separate table for each use case the issues (problems to be solved or possible improvements to be applied in Y3) identified by each use case are reported.

Technical evaluation of Use Case B3

<table>
<thead>
<tr>
<th>Feature</th>
<th>Documentation</th>
<th>Architecture</th>
<th>API</th>
<th>Implementation</th>
<th>Integration Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework Configuration &amp; Setup</td>
<td>Nearly Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Framework Model Configuration</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Framework Operation</td>
<td>Nearly Sufficient</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Low</td>
</tr>
<tr>
<td>Framework Access</td>
<td>Nearly Sufficient</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Service properties dependencies coding</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Low</td>
</tr>
<tr>
<td>Designing of QoS predictable systems</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Nearly Sufficient</td>
<td>Excellent</td>
<td>Medium</td>
</tr>
<tr>
<td>Discovery of Serv. and SLAT</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Feature</td>
<td>Documentation</td>
<td>Architecture</td>
<td>API</td>
<td>Implementation</td>
<td>Integration Complexity</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------</td>
<td>--------------</td>
<td>-----------</td>
<td>---------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Product Discovery</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Product Definition</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>SLAT Definition</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
<td>High</td>
</tr>
<tr>
<td>Business Negotiation</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>Automatic orchestration of customisable (re)negotiation protocol</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Coordination of Provisioning</td>
<td>Good</td>
<td>Good</td>
<td>Sufficient</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>On the fly deploy of monitoring</td>
<td>Sufficient</td>
<td>Excellent</td>
<td>Good</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Virtual hw. infrastructure provisioning</td>
<td>Good</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Software provisioning</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Software landscape</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Automated SLA enforcement coordination</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Nearly Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Virtual hw. infrastructure Adjustment</td>
<td>Sufficient</td>
<td>Excellent</td>
<td>Good</td>
<td>Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>Software Adjustment</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Runtime Software SLAs violations prediction</td>
<td>Good</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>High</td>
</tr>
<tr>
<td>Monitoring coordination</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Monitoring rules extraction from SLA</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
</tbody>
</table>
### Table 4: Technical evaluation of Use Case B3

<table>
<thead>
<tr>
<th>Feature</th>
<th>Documentation</th>
<th>Architecture</th>
<th>API</th>
<th>Implementation</th>
<th>Integration Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLA Management</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Push/Pull Business SLA Reporting</td>
<td>Nearly Sufficient</td>
<td>Good</td>
<td>Sufficient</td>
<td>Nearly Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>SLA Status Reporting</td>
<td>Nearly Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Virtual hw Infrastructure Reporting</td>
<td>Nearly Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
</tbody>
</table>

### Issues reported from Use Case B3

<table>
<thead>
<tr>
<th>Feature</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designing of QoS predictable systems</td>
<td>Model-driven performance evaluation turned out to be insufficient for use case B3.</td>
</tr>
<tr>
<td>SLAT Definition</td>
<td>There is no editor in place that supports creation of properly validated SLATs.</td>
</tr>
</tbody>
</table>

### Table 5: Issues reported by Use Case B3
## Technical evaluation of Use Case B4

<table>
<thead>
<tr>
<th>Feature</th>
<th>Documentation</th>
<th>Architecture</th>
<th>API</th>
<th>Implementation</th>
<th>Integration Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework Configuration &amp; Setup</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Insufficient</td>
<td>Sufficient</td>
<td>Very High</td>
</tr>
<tr>
<td>Framework Model Configuration</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Very High</td>
</tr>
<tr>
<td>Framework Operation</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Framework Access</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Customer Registration</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Discovery of Serv. and SLAT</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Product Management</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Product Discovery</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Product Definition</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>SLAT Definition</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Coordination of Provisioning</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Low</td>
</tr>
<tr>
<td>On the fly deploy of monitoring</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Virtual hw. infrastructure provisioning</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Low</td>
</tr>
</tbody>
</table>
### Feature Evaluation

<table>
<thead>
<tr>
<th>Feature</th>
<th>Documentation</th>
<th>Architecture</th>
<th>API</th>
<th>Implementation</th>
<th>Integration Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated SLA enforcement coordination</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>Virtual hw. infrastructure Adjustment</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Runtime infrastructure metrics prediction</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
<td>Medium</td>
</tr>
<tr>
<td>Virtual hardware infrastructure</td>
<td>Insufficient</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Insufficient</td>
<td>High</td>
</tr>
<tr>
<td>monitoring rules extraction</td>
<td>Insufficient</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Insufficient</td>
<td>High</td>
</tr>
<tr>
<td>Monitoring rules extraction from SLA</td>
<td>Insufficient</td>
<td>Insufficient</td>
<td>Insufficient</td>
<td>Insufficient</td>
<td></td>
</tr>
<tr>
<td>Virtual hw Infrastructure Reporting</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Table 6: Technical evaluation of Use Case B4**

### Issues reported from Use Case B4

<table>
<thead>
<tr>
<th>Feature</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework Configuration &amp; Setup</td>
<td>Runs on non supported Java Version (version 5)</td>
</tr>
<tr>
<td>Framework Model Configuration</td>
<td>No editor for SLA models. Extensions are not supported in year 2</td>
</tr>
<tr>
<td>Feature</td>
<td>Issues</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Framework Operation</td>
<td>Default deployment with PAX Runner is very complex &amp; requires internet access</td>
</tr>
<tr>
<td>Framework Access</td>
<td>Would be nice to have REST and messaging based interfaces</td>
</tr>
<tr>
<td>Designing of QoS predictable systems</td>
<td>Usage profiles are important with regards to the provisioning of services</td>
</tr>
<tr>
<td>Customer Registration</td>
<td>In B4 we haven’t seen or used the feature. It’s a B4 requirement but we're assuming it's ok from A5.</td>
</tr>
<tr>
<td>Discovery of Serv. and SLAT</td>
<td>Searching of SLATs is a future feature</td>
</tr>
<tr>
<td>Product Discovery</td>
<td>Cannot search SLAs by their internal agreement terms, only by ID's and/or other generated metadata, which is not as useful</td>
</tr>
<tr>
<td>Product Definition</td>
<td>We haven’t seen or used the feature in year 2. It’s a B4 requirement. We're assuming as A2 were able to demo it in year 1 that it will be ok in y2.</td>
</tr>
<tr>
<td>SLAT Definition</td>
<td>Yes SLATs can be defined, but requires expert level knowledge of the SLA model. Perhaps an editor would help with this, as per B3's suggestion</td>
</tr>
<tr>
<td>Business Negotiation</td>
<td>B4 does this themselves by provisioning through the UI and Apache Tashi. We need the feature but we are not using the A-line component so have not rated it.</td>
</tr>
<tr>
<td>SLAT Customization</td>
<td>B4 does this themselves through the UI. We need the feature but we are not using the A-line component so have not rated it.</td>
</tr>
<tr>
<td>Coordination of Provisioning</td>
<td>Have used it already</td>
</tr>
<tr>
<td>On the fly deploy of monitoring</td>
<td>At the moment, it's static, as A4 needs to resolve issues as A5 has not received these monitoring features to date</td>
</tr>
<tr>
<td>Virtual hw. infrastructure provisioning</td>
<td>Using this already</td>
</tr>
<tr>
<td>Software provisioning</td>
<td>B4 is PaaS only</td>
</tr>
<tr>
<td>Feature</td>
<td>Issues</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Software landscape</td>
<td>B4 is PaaS only</td>
</tr>
<tr>
<td>Automated SLA enforcement coordination</td>
<td>Aware of implementation but not integrated or deployed currently in B4</td>
</tr>
<tr>
<td>Virtual hw. infrastructure Adjustment</td>
<td>Basic functions already being used. More complex ones will probably not be available till Y3</td>
</tr>
<tr>
<td>Runtime infrastructure metrics prediction</td>
<td>Gathering prediction data in year 2 in ganglia/collectl. Already used in for demo purposes.</td>
</tr>
<tr>
<td>Virtual hardware infrastructure monitoring rules extraction</td>
<td>Some work is complete on this, but it requires pre-configured values rather than dynamic ones. Ownership of this component within the overall project is not clear to B4.</td>
</tr>
<tr>
<td>Virtual hardware infrastructure observation and violation detection</td>
<td>Some work is complete on this, but it requires pre-configured values rather than dynamic ones.</td>
</tr>
<tr>
<td>Monitoring rules extraction from SLA</td>
<td>No rating on complexity because we have not seen anything on it.</td>
</tr>
<tr>
<td>Virtual hw Infrastructure Reporting</td>
<td>Historical DB is not fully integrated yet. Prediction data is available</td>
</tr>
</tbody>
</table>

**Table 7: Issues reported on Use Case B4**
## Technical evaluation of Use Case B5

<table>
<thead>
<tr>
<th>Feature</th>
<th>Documentation</th>
<th>Architecture</th>
<th>API</th>
<th>Implementation</th>
<th>Integration Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework Configuration &amp; Setup</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td></td>
<td>Sufficient</td>
<td>Very High</td>
</tr>
<tr>
<td>Framework Model Configuration</td>
<td>Sufficient</td>
<td>Nearly Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>High</td>
</tr>
<tr>
<td>Framework Operation</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Framework Access</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Reference Software Manageability Components and Configuration</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Service properties dependencies coding</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Manageability Design</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Good</td>
<td>High</td>
</tr>
<tr>
<td>Customer Registration</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Negligible</td>
</tr>
<tr>
<td>3rd Parties Configuration</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Negligible</td>
</tr>
<tr>
<td>Discovery of Serv. and SLAT</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>Product Management</td>
<td>Good</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Product Discovery</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Feature</td>
<td>Documentation</td>
<td>Architecture</td>
<td>API</td>
<td>Implementation</td>
<td>Integration Complexity</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------</td>
<td>--------------</td>
<td>------------</td>
<td>----------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Product Definition</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>SLAT Definition</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Very High</td>
</tr>
<tr>
<td>Business Negotiation</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>SLAT Customization</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Low</td>
</tr>
<tr>
<td>Business SLA planning for negotiation</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Out of band SLA Registration</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Coordination of Provisioning</td>
<td>Sufficient</td>
<td>Good</td>
<td>Sufficient</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>On the fly deploy of monitoring</td>
<td>Sufficient</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Dynamic binding setting</td>
<td>Good</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Virtual hw. infrastructure provisioning</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Software provisioning</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Software landscape</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Automated SLA enforcement coordination</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
<td>Low</td>
</tr>
<tr>
<td>Virtual hw. infrastructure</td>
<td>Sufficient</td>
<td>Good</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Feature</td>
<td>Documentation</td>
<td>Architecture</td>
<td>API</td>
<td>Implementation</td>
<td>Integration Complexity</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------</td>
<td>--------------</td>
<td>---------</td>
<td>----------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Adjustment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Adjustment</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
<td>Low</td>
</tr>
<tr>
<td>Automatic Binding</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Business adjustment</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Runtime infrastructure metrics prediction</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Runtime Software SLAs violations prediction</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Monitoring coordination</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Virtual hardware infrastructure monitoring rules extraction</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Virtual hardware infrastructure observation and violation detection</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Software violation detection</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Monitoring rules extraction from SLA</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>SLA Management</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
<td>Low</td>
</tr>
</tbody>
</table>
### Table 8: Technical evaluation of Use Case B5

<table>
<thead>
<tr>
<th>Feature</th>
<th>Documentation</th>
<th>Architecture</th>
<th>API</th>
<th>Implementation</th>
<th>Integration Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push/Pull Business SLA Reporting</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Excellent</td>
<td>Low</td>
</tr>
<tr>
<td>Virtual hw Infrastructure Reporting</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
</tbody>
</table>

### Issues reported from Use Case B5

<table>
<thead>
<tr>
<th>Feature</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework Configuration &amp; Setup</td>
<td>Not finished yet. OSGi integration very complex.</td>
</tr>
<tr>
<td>Framework Model Configuration</td>
<td>Must be updated in Y3</td>
</tr>
<tr>
<td>SLAT Definition</td>
<td>Very complex and difficult to define and validate SLATs</td>
</tr>
</tbody>
</table>

**Table 9: Issues reported by Use Case B5**
## Technical evaluation of Use Case B6

<table>
<thead>
<tr>
<th>Feature</th>
<th>Documentation</th>
<th>Architecture</th>
<th>API</th>
<th>Implementation</th>
<th>Integration Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework Configuration &amp; Setup</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Framework Model Configuration</td>
<td>Sufficient</td>
<td>Good</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Framework Operation</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Framework Access</td>
<td>Sufficient</td>
<td>Good</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Reference Software Manageability Components and Configuration</td>
<td>Sufficient</td>
<td>Insufficient</td>
<td>Insufficient</td>
<td>Insufficient</td>
<td>Negligible</td>
</tr>
<tr>
<td>Service properties dependencies coding</td>
<td>Good</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Designing of QoS predictable systems</td>
<td>Good</td>
<td>Good</td>
<td>Nearly Sufficient</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Customer Registration</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>3rd Parties Configuration</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>Discovery of Serv. and SLAT</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Product Management</td>
<td>Nearly Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Product Definition</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Feature</td>
<td>Documentation</td>
<td>Architecture</td>
<td>API</td>
<td>Implementation</td>
<td>Integration Complexity</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------</td>
<td>--------------</td>
<td>---------</td>
<td>----------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>SLAT Definition</td>
<td>Good</td>
<td>Good</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Business Negotiation</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Business SLA planning for negotiation</td>
<td>Insufficient</td>
<td>Good</td>
<td>Sufficient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automatic orchestration of customisable (re)negotiation protocol</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Out of band SLA Registration</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Coordination of Provisioning</td>
<td>Sufficient</td>
<td>Good</td>
<td>Sufficient</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>On the fly deploy of monitoring</td>
<td>Good</td>
<td>Good</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Medium</td>
</tr>
<tr>
<td>Dynamic binding setting</td>
<td>Good</td>
<td>Good</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>Software provisioning</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Sufficient</td>
<td>Insufficient</td>
<td>High</td>
</tr>
<tr>
<td>Automated SLA enforcement coordination</td>
<td>Good</td>
<td>Good</td>
<td>Sufficient</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Software Adjustment</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Automatic Binding</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
<tr>
<td>Business adjustment</td>
<td>Sufficient</td>
<td>Good</td>
<td>Good</td>
<td>Nearly Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>Runtime Software SLAs violations prediction</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Feature</td>
<td>Documentation</td>
<td>Architecture</td>
<td>API</td>
<td>Implementation</td>
<td>Integration Complexity</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>---------------</td>
<td>--------------</td>
<td>--------------</td>
<td>----------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Monitoring coordination</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Software violation detection</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>Monitoring rules extraction from SLA</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>High</td>
</tr>
<tr>
<td>SLA Management</td>
<td>Nearly Sufficient</td>
<td>Sufficient</td>
<td>Nearly Sufficient</td>
<td>Nearly Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>Push/Pull Business SLA Reporting</td>
<td>Sufficient</td>
<td>Good</td>
<td>Sufficient</td>
<td>Nearly Sufficient</td>
<td>Low</td>
</tr>
<tr>
<td>SLA Status Reporting</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 10: Technical evaluation of Use Case B6

Issues reported from Use Case B6

<table>
<thead>
<tr>
<th>Feature</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework Configuration &amp; Setup</td>
<td>The adoption of OSGi delays the adoption of the framework.</td>
</tr>
<tr>
<td>Framework Model Configuration</td>
<td>Documentation sparse in different documents.</td>
</tr>
<tr>
<td>Reference Software Manageability Components and Configuration</td>
<td>No facility for monitoring composed services is offered.</td>
</tr>
<tr>
<td>Service properties dependencies coding</td>
<td>It’s not clear how to treat dependency from third party services.</td>
</tr>
<tr>
<td>Feature</td>
<td>Issues</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SLAT Definition</td>
<td>The SLA Model does not support explicit dependency between SLAs.</td>
</tr>
<tr>
<td>Software provisioning</td>
<td>Poor guidelines for customization.</td>
</tr>
<tr>
<td>Runtime Software SLAs violations prediction</td>
<td>Currently some KPIs used by B6 (e.g. &quot;number of calls with a certain response time&quot;, cannot be predicted).</td>
</tr>
<tr>
<td>SLA Management</td>
<td>Unclear how to access the history of SLAs</td>
</tr>
<tr>
<td>Push/Pull Business SLA Reporting</td>
<td>Reporting at the level of single guarantee term should be supported.</td>
</tr>
</tbody>
</table>

**Table 11: Issues reported by Use Case B6**