Deliverable DA3.a

SLA-Aware Service Management

Keywords:
Service Level Agreement, Service Management, Service Monitoring, Service composition

Due date of deliverable: 31st of July 2011
Actual submission to EC date: 31st of July 2011

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

Lead contractor for this deliverable: SAP
Revision: V.1.0 (July 28th, 2011)
The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 216556.

### Document Status

<table>
<thead>
<tr>
<th>Deliverable Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliverable Lead</td>
<td>Wasif Gilani (SAP)</td>
</tr>
<tr>
<td>Reviewer 1</td>
<td>Christoph Rathfelder (FZI)</td>
</tr>
<tr>
<td>Reviewer 2</td>
<td>Miha Stopar (XLAB)</td>
</tr>
<tr>
<td>PMT Reviewer</td>
<td>Joe Butler (Intel)</td>
</tr>
<tr>
<td>Complete version submitted to reviewers</td>
<td>17th of June, 2011</td>
</tr>
<tr>
<td>Comments of reviewer 1 received</td>
<td>26th of June, 2011</td>
</tr>
<tr>
<td>Comments of reviewer 2 received</td>
<td>20th of July, 2011</td>
</tr>
<tr>
<td>Revised deliverable submitted to PMT</td>
<td>24th of July, 2011</td>
</tr>
<tr>
<td>PMT Approval</td>
<td>27th of July 2011</td>
</tr>
</tbody>
</table>

### Contributors

<table>
<thead>
<tr>
<th>Partner</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP</td>
<td>Wasif Gilani</td>
</tr>
<tr>
<td>Engineering</td>
<td>Paolo Zampognaro</td>
</tr>
<tr>
<td>CITY</td>
<td>Howard Foster</td>
</tr>
<tr>
<td>PMI</td>
<td>Sam Guinea</td>
</tr>
<tr>
<td>FBK</td>
<td>Natallia Rasadka</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.

### Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>27/05/2011</td>
<td>Wasif Gilani</td>
<td>Initial Structure and Contents</td>
</tr>
<tr>
<td>0.2</td>
<td></td>
<td>Howard Foster, Wasif Gilani, Natallia Rasadka, Sam Guinea, Paolo Zampognaro</td>
<td>Sections 2, 3,4 and 5 added</td>
</tr>
<tr>
<td>0.3</td>
<td>17/06/2011</td>
<td>Wasif Gilani</td>
<td>First consolidated version ready</td>
</tr>
<tr>
<td>0.5</td>
<td>26/06/2011</td>
<td>Wasif Gilani</td>
<td>Comments and suggestions from first internal review</td>
</tr>
<tr>
<td>0.8</td>
<td>20/07/2011</td>
<td>Wasif Gilani</td>
<td>Comments and suggestions from second internal review</td>
</tr>
<tr>
<td>0.9</td>
<td>27/07/2011</td>
<td>Howard Foster, Wasif Gilani</td>
<td>Comments and</td>
</tr>
<tr>
<td>1.0</td>
<td>Gilani, Natallia Rasadka, Sam Guinea, Paolo Zampognaro</td>
<td>recommendations from reviewers addressed</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-------------------------------------------------</td>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>29/07/2011</td>
<td>Wasif Gilani</td>
<td>Final version ready</td>
<td></td>
</tr>
</tbody>
</table>
Executive Summary

SLA@SOI aims at addressing the SLA management problem from a holistic perspective. The objective is to design a multi-layer SLA management framework for SOA based application landscapes. The focus of work package A3 is the software service layer and aims at designing SLA aware service management capabilities for the overall SLA management framework. In this regard, WP A3 has divided the service management into a number of distinct yet complementary and interconnected working tracks. This deliverable document provides description of the activities, progress and achievements in different tracks of A3 during the course of Y3. The overall direction of Y3 activities within SLA@SOI have been:

- In the Service Manageability area, significant progress has been made regarding the runtime management of software services, with a special focus on BPEL processes.
- In the Dynamic Service Discovery & Composition area, the mechanism supported by the DOE (Dynamic Orchestration Engine) has been extended for the automatic SLAT based (i.e. the mechanism for execution of abstract invoke activities) dynamic binding.
- In the Service Monitoring area, significant contributions have been made in the context of SLA coverage testing, planning and optimization validation, monitoring engine validation and monitoring instrumentation for the SLA@SOI eclipse studio.
- In the SOA Modelling area, SLA management area has been broadened to cover the area of business process continuity management. The ERP B3 framework has been extended with a process-centric Business Continuity Analysis toolkit.
# Table of Contents

1 Introduction .............................................................................................................. 9  
1.1 Context and scope ............................................................................................... 9  
1.2 Main achievement ............................................................................................... 9  
1.3 Document Overview ........................................................................................... 10  
2 Enhanced SOA Modelling ...................................................................................... 13  
2.1 Introduction ......................................................................................................... 13  
2.2 Business Continuity Management ....................................................................... 15  
2.3 Related Work ...................................................................................................... 15  
2.4 Model-Driven and Process-Centric BCM Framework ......................................... 17  
2.4.1 Requirements ............................................................................................... 17  
2.4.2 Architecture .................................................................................................. 19  
2.4.3 Stakeholders .................................................................................................. 20  
2.4.4 Environment .................................................................................................. 21  
2.4.5 Workflow and Methodology .......................................................................... 21  
2.4.6 IT BCM Model Derivation ............................................................................. 23  
2.4.7 BEAM Model Derivation .............................................................................. 24  
2.4.8 Alpha BEAM ................................................................................................ 25  
2.4.9 Beta BEAM ................................................................................................... 27  
2.4.10 Gamma BEAM ............................................................................................. 28  
2.4.11 Business Continuity Analysis ....................................................................... 28  
2.4.12 Analysis Result Presentation ....................................................................... 28  
2.4.13 Tracing .......................................................................................................... 30  
2.4.14 Context Sensitive Presentation Mode ............................................................. 30  
2.5 Conclusions and Outlook .................................................................................... 32  
3 Service Manageability ............................................................................................ 34  
3.1 Overview ............................................................................................................. 34  
3.2 Support for Event Correlation ............................................................................ 34  
3.3 Process restructuring .......................................................................................... 36  
3.3.1 BPEL Graph .................................................................................................. 36  
3.3.2 Translation of BPEL to BPEL Graph .............................................................. 37  
3.3.3 BPEL Graph Analysis .................................................................................... 41  
3.3.4 Restructuring Algorithm ............................................................................... 43  
3.3.5 Example of Parallelization ............................................................................ 44  
3.3.6 Parallelization Restriction ............................................................................. 46  
3.3.7 Converting BPEL Graph to BPEL ................................................................. 47  
3.4 Runtime Adaptation of Process Flows .................................................................. 47  
3.5 Using the Unified Manageability Interface .......................................................... 48  
3.5.1 Managing event correlations ....................................................................... 49  
3.5.2 Managing Adaptations .................................................................................. 50  
3.6 Related Work ...................................................................................................... 50  
3.7 Conclusions and Outlook .................................................................................... 53  
4 Dynamic Service Discovery & Composition .......................................................... 54  
4.1 Architecture & packaging .................................................................................. 54  
4.2 DOE: multi-party conversational model .............................................................. 55  
4.2.1 Concepts ....................................................................................................... 55  
4.2.2 Extension of DOE management interface .................................................... 57  
4.2.3 Run time environment ................................................................................... 63  
4.3 DOE: installation notes ...................................................................................... 65  
4.4 Related Work ..................................................................................................... 65  
4.5 Conclusions & Outlook ..................................................................................... 66  
5 Service Monitoring .................................................................................................. 67  
5.1 Overview ............................................................................................................ 67
5.2 Monitoring Architecture................................................................. 67
5.3 Monitorability Assessment............................................................ 69
5.4 Packages for Deployment............................................................... 70
5.5 SMaRT Workbench ......................................................................... 70
  5.5.1 Architecture ............................................................................. 71
  5.5.2 Monitorability Reporting............................................................ 72
  5.5.3 Workbench................................................................................. 72
5.6 Testing and Validation ................................................................... 75
  5.6.1 SLA Coverage Testing............................................................... 75
  5.6.2 Validation ................................................................................ 76
5.7 Related Work.................................................................................. 76
5.8 Conclusions and Outlook............................................................... 77
6 Conclusions....................................................................................... 78
7 References......................................................................................... 79
Table of Figures

Figure 1 BCM framework architecture .......................................................... 19
Figure 2: BCM positioning in SLA@SOI framework ....................................... 20
Figure 3 Business Process Requirements Annotation ..................................... 23
Figure 4 Generation of IT BC Model ............................................................... 24
Figure 5 IT BC Library Models ....................................................................... 25
Figure 6 Business Process Requirement Annotation Editor ............................ 26
Figure 7 Alpha-Beam generation .................................................................... 27
Figure 8 Beta MEAM model generation ......................................................... 28
Figure 9 Gamma BEAM ................................................................................. 29
Figure 10 Business Continuity Analysis .......................................................... 30
Figure 11 Post-Analysis filtering and reasoning .............................................. 31
Figure 12 Trace models in the architecture ...................................................... 32
Figure 13 Architecture of the Context Sensitive Analysis Result ..................... 33
Figure 14 Document-oriented result presentation .......................................... 34
Figure 15 The Manageability Agent in the context of the framework. ............ 35
Figure 16 Information that can be obtained through correlation ....................... 36
Figure 17 Healthcare and mobility booking: original workflow ....................... 37
Figure 18 Restructured workflow .................................................................... 38
Figure 19 Architecture for runtime process modifications ............................... 39
Figure 20 The unified manageability interface. ................................................. 40
Figure 21 Managing a process’ bindings .......................................................... 41
Figure 22 Adapting a process’ structure .......................................................... 42
Figure 23 Architecture and packaging ............................................................. 43
Figure 24 SCA multi valued reference variable .............................................. 44
Figure 25 Syntax of multiReference extension .............................................. 45
Figure 26 sca:serviceReferenceList declaration .......................................... 46
Figure 27 Example of multi-valued reference application ............................... 47
Figure 28 addRuleForSingleBinding: details of input data ............................. 48
Figure 29 addRuleForExtendedSingleBinding: details of input data ............... 49
Figure 30 addRuleForMultipleBinding: details of input data ......................... 50
Figure 31 addRuleForExtendedMultipleBinding: details of input data .......... 51
Figure 32 addAbstractRuleForSingleBinding: details of input data .............. 52
Figure 33 addAbstractRuleForExtendedSingleBinding: details of input data 53
Figure 34 addAbstractRuleForMultipleBinding: details of input data .......... 54
Figure 35 addAbstractRuleForExtendedMultipleBinding: details of input data 55
Figure 36 Class diagram: Binder internal relationships ................................. 56
Figure 37 Run time environment detail ........................................................... 57
Figure 38 – Extended architecture ................................................................. 58
Figure 39 The Service Monitoring Architecture ............................................. 59
Figure 40 Monitorability Assessment Activity Diagram ................................ 60
Figure 41 MonitoringManager Module Packages .......................................... 61
Figure 42 The SMaRT Architecture ............................................................... 62
Figure 43 Tree Structure for Monitorability Reporting Items .......................... 63
Figure 44 SMaRT: Monitorability Report View ............................................. 64
Figure 45 SMaRT: SLA Term Decomposition Results View ......................... 65
Figure 46 SMaRT: Monitoring Components Specification View .................... 66
Figure 47 Partial Results of Testing with Use-case Elements ........................ 67
1 Introduction

1.1 Context and scope

This deliver provides an overview of the progress made in various building blocks which enable service management. The work described in this deliverable contributes to the overall SLA Management framework being developed within the SLA@SOI project. More specifically, work described in this document addresses the SLA management related aspects at the software / service layer of the IT stack.

This deliverable is the final document from the A3 workpackage that presents the work carried out within work package A3 till month 38. This deliverable presents progress on particular architectural building blocks (Service Management) of the overall SLA@SOI’s SLA Management framework. The overall architecture is described in the deliverable D.A1a and other architectural building blocks are presented in deliverables D.A2a (Business Management), D.A4a (Infrastructure Management), D.A5a (SLA Foundations) and D.A6a (Predictable Systems Engineering). The work carried out within this workpackage is driven by the requirements from various use cases. These use cases employ the tools and technology being developed within this and other workpackages.

SLA-aware Service Management addresses a number of aspects. First, modelling related activities focus on specifying metamodels for representing software and service landscapes for effective and efficient service management for large and complex landscapes. Secondly, a unified manageability interface is required for facilitating streamlined interactions with heterogeneous elements of the SOA landscapes. Thirdly, a dynamic orchestration engine enables composite service execution with the capability to perform dynamic service bindings to ensure compliance with the service level agreements established with the customer. Fourth, a large part of service management focuses on monitoring and runtime adjustment. Monitoring is essential to comprehend the execution status of the services and software elements.

1.2 Main achievement

The A3 work package has made important conceptual and technical contributions to the project’s SLA management framework and its open-source distribution. Examples are the modelling related activities for software and service landscapes, the unified manageability interface, and the monitoring tools.

Moreover, it has also created a second open-source project called BPEL Management Toolkit (http://sourceforge.net/projects/bpelmt/) that complements the main framework’s distribution. In particular, there are three modules, the Dynamic Orchestration Engine (a.k.a. DOE), its instrumentation for configuring runtime sensors, and an approach for BPEL adjustment. These contributions are not part of the platform, but they can be used to demonstrate the capabilities of the platform in a number of situations. These modules enable the dynamic configuration of process bindings by means of rules, the execution of abstract processes. The instrumentation allows for the runtime configuration of sensors within a deployed and running BPEL process. BPEL adjustment permits the structural modification of the BPEL process that implements the service in those cases when this may be needed.
1.3 Document Overview

The remainder of this document is structured as follows. Section 2 focuses on an elaborate description of the modelling activities, and presents an overview of the contributions in the area of SOA Modelling. Section 3 describes the contributions made in the Service Manageability area, and concentrates on the uniform manageability interface and autonomic service management activities. The progress made in the area of Dynamic Service Discovery & Composition is provided in Section 4. Section 5 provides a detailed description of the instrumentation, monitoring and analysis aspects. The deliverable ends with a summary and conclusion section.

The following table lists all the changes and new elements of this deliverable compared to the previous deliverable version.

Table 1: Document changes against previous version

<table>
<thead>
<tr>
<th>Section</th>
<th>Change overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 3</td>
<td><strong>Enhanced SOA Modelling:</strong> The IT ecosystem has been undergoing a transformation in the recent years. This transformation is leading the IT ecosystem from a product based landscape to a service based IT landscape. This means that the IT elements which were previously being bought by businesses are now offered by service providers in a service based fashion. For example, infrastructure providers offer infrastructure services aka Infrastructure as a Service (IaaS), software is delivered as software services aka Software as a Service (SaaS). The complete solution used by business and enterprises is hence composed of pieces from various service providers. As businesses adopt this new IT landscape and procure and consume services to power their business operations and processes, it becomes critical for the service providers to continuously improve the services being delivered to the customers. Business continuity is the new area that is addressed in this chapter which comes as an enhancement of the SOA modelling. Business Continuity is the foremost objective of the organizations and service disruptions can compromise this objective leading to financial loss as well as business reputation and customer base. To effectively cope with this, service customers would like to have some quality of service guarantees from the service providers agreed in the form of service level agreements (SLA). Service providers, on the other hand should continuously strive to ensure that the objectives stipulated in the SLAs must be sustained. An effective, agile and accurate SLA management framework becomes a top most requirement in the upcoming service based economy. This chapter describes in detail the model driven BCM framework implemented in Y3 which reuses various modelling concepts from the SLA @SOI framework.</td>
</tr>
<tr>
<td>Chapter 4</td>
<td><strong>Service Manageability:</strong> Progress has been made regarding the runtime management of software services during the project’s third year, with a special focus on BPEL processes. Regarding behavioural monitoring, the ManageabilityAgents have been extended with event</td>
</tr>
</tbody>
</table>
general and domain-specific event correlation capabilities. Regarding process adaptation an algorithm has been implemented for structure re-writing. An extension is further provided to the ActiveBPEL BPEL execution engine that will allow changes to the process’ structure to be made on-the-fly, both at the process definition level and at the process instance level. Finally, the DOE manageability façade has been extended to be able to deal with all the above novelties, as well as with the new multi-party binding feature provided by the DOE.

### Chapter 5

**Dynamic Service Discovery & Composition:** Investigation of dynamic service discovery and composition of services is one major activity within WP A3. The task aims at identifying methodologies for integrating dynamic service discovery and composition into the holistic SLA management approach undertaken by SLA@SOI consortium.

In Year 3, the mechanism, supported by the DOE (Dynamic Orchestration Engine) for automatic dynamic binding (i.e. the mechanism for execution of abstract invoke activities), has been extended. In the previous implementation, it was only possible to automatically bind the partner role of an abstract invoke with one (and only one) web service, according to the WS-BPEL specification, which refers exclusively to a bi-party conversational model [15]. This limitation has been overcome by the implementation of a multi-party conversational model which allows invoking, in one step, several partners exposing the same operation. Such improvement has represented, for A3, both an innovation challenge and a specific requirements coming from B line (see B6 use case). Moreover, the automatic binding mechanism has been extended in order to take also the monitored data into account. The matching SLAs can now be ordered on the base of the monitored QoS and the binding algorithm may choose the best one with respect to specified criteria.

### Chapter 6

**Service Monitoring:** In the previous SLA@SOI A3 period (as reported in [18]) a mechanism was implemented to support dynamic configuration of monitoring infrastructures by selecting service providers from their matching capabilities of providing service monitors for consumer configuration preferences. Mechanical support is provided for this monitoring configuration for the wider service level agreement driven architecture. In this period, specific contributions have been made to the SLA@SOI monitoring architecture solution with the following innovations and achievements:

- **Enhanced Support for SLA Model Reasoning.** Additional support for Agreement Term and Guaranteed State Pre-condition reasoning and configuration, as described in the SLA model specification are now implemented.
- **SLA Coverage Testing.** An extensive SLA term and expression coverage activity was undertaken to
<table>
<thead>
<tr>
<th>Planning and Optimization Validation. Further integration testing with the Planning and Optimization (POC) component as part of ORC scenarios.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitoring Engine Validation (as with the POC).</td>
</tr>
<tr>
<td>Monitorability Instrumentation for the SLA@SOI Eclipse Studio.</td>
</tr>
</tbody>
</table>

In this deliverable, an approach is reported to advanced configuration of service systems, in particular, ones in which an SLA agreement has been established and has services requiring monitoring. The work aims to provide a generic module applicable not only to the architecture illustrated but also to other architectures (although still based upon SLAs and Monitoring Component Features). A tool suite is provided to visualise the assessment of monitoring such systems.
2 Enhanced SOA Modelling

SOA modelling and management is more than pure management of a service based IT landscape. Analysis and discussions with numerous stakeholders suggested that there is a huge additional potential to leverage the approach of the SLA@SOI framework for supporting IT environments leveraging services.

Business continuity supports management of such extended environments and therefore goes beyond the pure management of IT stacks but also includes the operations at a data centre including cooling, support staff and many other aspects. Business Continuity Management (BCM) provides a Business Continuity Plan against various expected threats and disruptions. In order to conduct a sound Business Impact Analysis, BCM manager requires an integrated and coherent view of all involved business processes and IT related resources. However, this integrated and coherent view is hard to establish using conventional analysis methods, such as questionnaires and Visio diagrams. Hence, BCM managers struggle to create comprehensive, up-to-date and thorough BCM analyses.

Enterprise resource planning and related resources belong to the most critical parts of a customer’s IT stack. Interruptions of the ERP operations have immediate and severe effects on the continuity of a business leading to financial losses, legal consequences, losses in reputation and may cause bankruptcies.

ERP B3 is a SLA@SOI framework based solution for Service Level Agreements driven, on demand and dynamic provisioned business applications. We have extended the ERP B3 framework with a process-centric Business Continuity Analysis toolkit. The BCM framework is based on our previous work carried out in another European project called MODELPLEX. The BCM framework is coupled to SLA@SOI framework to consume arbitrary management models and concepts.

The BCM framework is highly model-driven and utilises model-driven technologies, such as model-to-model transformation. The BCM framework is able to consume models and components from SLA@SOI framework, such as infrastructure models from A4, software models from A3, and general service models from A1. The BCM framework conversly can be plugged in to the SLA@SOI framework to consume SLA definitions, translate BCM related SLAs, and perform service evaluation for service management.

Our model-driven, non-intrusive approach extends ERP business process management tools with an integrated IT BCM analysis support. Our non-intrusive approach permits automatic consolidation of information from various stakeholders, service providers and multiple data sources. Furthermore, different kinds of analysis tools, such as simulation tools and analytical tools, can be utilized in an automatic fashion.

Our BCM framework helps Business Continuity Manager and IT Service Managers to (a) estimate business impact of failed IT services (b) to validate SLAs and verify if selected SLAs are appropriated and (c) to determine an optimal set of service-level SLAs to minimise business disruptions.

2.1 Introduction

Nearly 1 in 5 businesses suffer a major disruption every year. These disruptions
in businesses often immediately affect thousands of business customers and consumers. A disruption could result in financial and legal losses as well as damage to reputation. For example, on January, 4th 2010, SALES FORCE, a company offering online enterprise support services, experienced an outage for over an hour which effected 68'000 business customers [6]. Another example would be PAYPAL, a service to process online payments. PAYPAL was down for 4.5 hours worldwide on August, 4th 2009. PAYPAL usually processes 2'000 USD per second for its customers. The key industrial sectors, such as energy, gas, oil, pharmacy or finance, have to particularly demonstrate business continuity competence, which is required by regulations and laws. A study to quantify Information and Communication Technologies (ICT) business continuity risks at ESSENT NETWERK, a Dutch electricity and gas distributor, revealed that a four hour outage of an IT service might result in a withdrawal of the licenses to operate and would eventually take ESSENT NETWERK out of business [11]. IT Business Continuity Management aims to:

- identify potential threats to an IT system, services and operations,
- assess the business impact of a threat, estimate probabilities and compute risk exposures,
- determine strategies and responses to these threats, and model an IT business continuity plan to overcome or mitigate a possible business disruption.

IT Business Continuity Managers struggle to conduct a comprehensive, thorough and valid analysis. The reasons for this are manifold:

- IT landscapes are complex and it is hard to formulate a complete picture. A BCM manager needs to comprehend and incorporate the complete end-to-end picture of resource dependencies of the whole IT stack in order to conduct a thorough dependency analysis.
- Different methodologies are used to model and document different aspects of complex IT landscapes. For example, business processes are modeled in workflow charts, the behaviour of software artifacts is expressed in UML activity diagrams and the IT infrastructure deployment layout is documented in topology models. Business Continuity Managers find themselves lost within this variety of heterogeneous, but related models, and they have limited visibility.
- IT landscapes are adaptive systems and evolve over time. New technologies, such as virtualization and cloud computing, allow to alter a landscape deployment layout within minutes.
- Current tool support and methodologies are insufficient to comprehend the impacts and consequences of failed elements of the entire landscape and dependent business processes.
- Visualization and reporting of analyses results is not automated and does not meet expectations of involved stakeholders.
- Last and most important, already available data sources, such as business process knowledge, UML or topology models are not utilized / reused for business continuity management.

All issues mentioned above may lead to inaccurate decisions made to circumvent disruptions.

In this section we present a novel framework, which addresses the needs of IT Business Continuity Management by:

- using model-driven engineering techniques to tap into business process knowledge and IT landscape models
- providing a Business Continuity (BC) model to simplify work for Business Continuity Managers and
- a two level model refinement process that increases the quality of the resulting
analysis model in terms of accuracy and precision utilizing automated model transformation chains to connect to a variety of data sources and analysis tools utilizing automated model transformation chains to feed analyses results into various reporting tools in order to visualize analyses results.

The remainder of this Section is organized as follows: Section 2.2 provides an overview on Business Continuity Management and Section 2.3 discusses related work. Section 2.4 introduce our framework. First motivation and challenges are discussed followed by an architecture description. Finally, Section 2.5 concludes our part with a summary and an outlook on future work.

## 2.2 Business Continuity Management

Business Continuity Management (BCM) is standardized by the British Standards Institution (BSI) and is formally defined as:

A holistic management process that identifies potential threats to an organization and the impacts to business operations that those threats, if realized, might cause, and which provides a framework for building organizational resilience with the capability for an effective response that safeguards the interests of its key stakeholders, reputation, brand and value-creating activities [4].

Business Continuity Management comprises four groups of activities, which are: understanding the organization, determining Business Continuity Strategies, developing and implementing a BCM response, and exercising, maintaining and reviewing BCM arrangements. All four activities are organized by a fifth activity, the BCM Program Management, which initiates business continuity related projects, assigns responsibilities, observes and manages activities, conducts training, and provides documentation.

To understand the organization, the Business Continuity Manager has to understand the effects of an adverse incident on a business and the dependencies among business processes, dependent resources and possible root-causes of an adverse incident. Business Continuity Manager uses two different, but complementary analyses, the Dependency and Risk Analysis (DA) and Business Impact Analysis (BIA). BIA aims to distinguish between mission critical processes, and non-critical business processes and functions. The Business Continuity Manager has to consider that a disruption of business processes may have a financial impact, legal consequences or may cause effects on other business values and indicators, such as reputation or customer satisfaction. For each business process and function, various different business continuity metrics are assigned. Return Time Objective (RTO) and Recovery Point Objective (RPO) are examples of BCM metrics. RPO defines “the maximum amount of data loss an organization can sustain during an event”. RTO defines the “target time for resumption of product, service or activity delivery after an incident” [4].

A Dependency Analysis is conducted to identify dependent resources, involved stakeholders, assets and internal/external service dependent to a critical business process. Also identified are possible failure modes and disruption causes. A Business Continuity Manager must be enabled to analyse how failures propagate through the system and layers. For example, the Manager needs to understand how a broken air-conditioning unit may affect the data centre and servers deployed in that data centre and eventually if and when a business process will be disrupted. Finally, a Recovery Plan details the steps to be taken to restore business operations to defined levels of operations within given timeframes.

## 2.3 Related Work
BCM is close to reliability engineering. Fault Tree Analysis (FTA) is a common technique used in reliability engineering to determine combinations of failures in a system that could lead to undesired events at the system level [5]. The modelling process starts with the undesired event and is broken down into a fault tree. Each fault is analysed in more detail and if necessary broken down again, until a reasonable level of understanding is achieved. The logical relationship between faults is defined by logical “gates”, such as AND, OR, XOR or NOT. Probabilities are assigned to basic events and the overall likelihood of an undesired event can be calculated. Although a fault tree analysis is able to provide a better estimation of the probability of adverse events to occur, such an analysis is not able to model the dynamic behaviour of systems since Boolean gates are not able to capture the order in which events occur, nor is it possible to model time constraints, such as dead-lines. This limits the application of FTA in IT BCM to very simple analyses.

The Tropos Goal-Risk (GR) Framework is used for requirement analysis and risk assessment for critical socio-technical systems, such as Air Traffic Management [1]. First, it provides means to model combinations of failures similar to FTA. Second, it provides semantics to model other aspects, such as time dependencies, treatments and assets, which are useful for BCM. Asnar and Giorgini demonstrated that the GR framework could be used to analyse and compare the effectiveness and cost-efficiency of different treatment strategies [2]. However, the analysis does not provide means to determine business impact nor does it provide means to determine BCM metrics, such as RTO. Moreover, TROPOS lacks strong business process integration and it does not provide means to generate TROPOS graphs from existing models or to integrate analysis techniques other than the ones provided by TROPOS itself [2,3]. From the model-driven engineering point of view performance models have been extracted from development models [3]. Most approaches focus on software development models, e.g. UML models, as input models whereas our approach aims to take various different kinds of models from different levels of IT stack into account, for example, business process models and IT topology models. Moreover, we aim to provide a cross-layer performance and dependency view. The following section below provides a description of our idea of consuming available models from different levels of technology stack and providing a systematic and business process-centric BCM solution.

ROPE is another related work that provides a Risk-Aware Business Process Modeling and Simulation Methodology [8, 9]. ROPE provides an approach to analyse the business impact of threats and the effect of counter measurements. ROPE uses a three-layer model approach. The first layer is the business process layer, the second layer refines business process layers with resource requirements. The third layer models threats impacts and related recovery actions. ROPE utilizes simulations to estimate the expected downtime of a business process and to evaluate recovery actions. Similar to our work ROPE analysis business impact on the business process activity level, provides a dependency model and considers threats and responses in a process centric way. However, ROPE does not provide any means to generate models for each ROPE layer from existing models, and hence is limited to manual created scenarios. Furthermore ROPE restricts itself to three layers as this simplifies modelling activities. However, due to this simplification ROPE is not able to analyse various important scenarios. For example, all recovery processes depend on resources themselves. If a resource, needed by a recovery plan is not available, the recovery activity fails. Due to the three-layer approach in ROPEs modelling methodologies, it is not possible to assign resources to recovery activities. Hence ROPE is not capable to analyse the important feasibility property of recovery plans. Our BCM framework is not restricted to three model layers and therefore permits more comprehensive analyses, for example feasibility analysis of recovery actions. To minimize and simplify modelling activities our approach
utilizes automated model transformations in combination with pre-defined libraries.

### 2.4 Model-Driven and Process-Centric BCM Framework

Business processes are the main subject or the starting point of a Business Impact Analysis and Dependency Analysis respectively. However, existing approaches are primarily focused on IT layers only, and do not take detailed business process knowledge into account. The business processes are generally abstracted as single black boxes in the top layers of dependency graphs. An increasing number of businesses are now employing Business Process Management (BPM) tools to model and execute their business processes. An incorporation of detailed business process information available via business process models (for example, BPMN [7], YAWL [10], etc.) can help to identify the dependencies, threats, etc., at the process step level, and provide greater context for BCM. Furthermore, tools used to conduct BCM analyses are not tailored according to the needs of Business Continuity Managers; often Business Continuity Managers use drawing tools like Visio with no analyses support at all to model a dependency graph. Without sufficient analysis tool support, Business Continuity Manager struggles to cope with a vast number of business processes and rapidly changing IT landscapes.

Our approach relies on model-driven engineering techniques. Model-driven engineering simplifies the development of tools for Domain Specific Languages (DSL). This increases adoption of domain specific models as software artefacts and permits contribution of domain experts, such as business process analysts, to system design and problem analysis. Models as software artifacts are therefore not uncommon in industrial applications nowadays. For instance, business process knowledge is not hard-coded into business software anymore, rather exists as well defined business process model artefacts. These business process models can be accessed by auxiliary software to open up new ways to improve business management in related areas, such as BCM.

#### 2.4.1 Requirements

We have identified the following two important requirements for a model-driven, business-process centric framework for Business Continuity Management:

**Heterogeneous Meta-Models and Tools:** Business Continuity Management covers the whole stack of an enterprise. This stack comprises various different domains and layers, such as business process domain, service composition and execution domain, IT infrastructure domain (software hardware, network, etc.), as well as facility items and human resources. Each domain is modelled by domain experts, e.g. the business process analyst, or the software architect, etc. Every domain expert potentially utilizes various different meta-models and tools to model and express domain specific needs. For example, business processes are documented with BPMN or YAWL, whereas software artefacts are depicted in UML. Of course, this separation of concerns is useful and desirable, since every model covers specific aspects of the respective domain. In most cases there exists no need to cross domains. However, BCM needs a cross-domain viewpoint on all domains to conduct a comprehensive and thorough business impact analysis and dependency analysis. To cope with heterogeneous, but complementary meta-models in a multi-tool environment is a major design and implementation challenge.
Multi-Paradigm Model Analyses, Reasoning and Model Optimisation: In the context of the Business Impact Analysis and Dependency Analysis, Business Continuity Manager employs various kind of analysis to quantify risks, validate recovery plans, etc. In essence BCM manager must be able to answer the following types of example questions given in Table 2.

Table 2 BCM related questions

<table>
<thead>
<tr>
<th>Issues</th>
<th>Example Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporal failure propagation</td>
<td>What is the expected time delay until a broken air conditioner in a data centre causes business disruptions?</td>
</tr>
<tr>
<td>Performance analysis</td>
<td>How many servers are needed to guarantee a sales order processing time of six hours even if one server breaks down for two hours?</td>
</tr>
<tr>
<td>Dead-lock detection</td>
<td>Is the recovery plan to repair a broken air-conditioner, to reboot all servers and to replay database transaction logs sufficient to guarantee a Business Process RTO of less than four hours?</td>
</tr>
<tr>
<td>Worst-case / Residual risk analysis</td>
<td>What is the likelihood that a recovery plan fails?</td>
</tr>
<tr>
<td>Sensitivity analysis</td>
<td>Which is the single point of failure?</td>
</tr>
<tr>
<td>Estimated value analysis</td>
<td>Is the organization willing to spend 5,000 USD per month for additional air-conditioning units to remove the 10% risk of a broken airconditioning unit, which would cost the organization 70,000 USD?</td>
</tr>
</tbody>
</table>

Business Continuity Manager should be able to run any kind of analysis across all layers of the enterprise stack by “pushing a button”. This is a challenging task, since it requires integrating knowledge captured from different levels of enterprise stack in different domain specific models, and data sources with existing multi-paradigm analyses tools, i.e. simulation engines, optimization tools, model checkers, etc.

Role based Analyses Results Presentations: Business Continuity Analyses Results have to be presented to various stakeholders. Stakeholders are (but not limited to) Business Continuity Manager, Business Process Analyst, Line-of-Business Manager, IT Architects and external reviewers. Every stakeholder has distinctive interests and hence has distinctive requirements how Business Continuity analyses results should be evaluated and presented. For example a Business Process Analyst is interested in finding out how well certain business process activities perform in case of disruptions whereas the Line-of-Business manager is more interested in the overall process resilience capability and the potential impacts on strategic business values of adverse events. The IT architect interests are focused on IT deployment requirement and the external reviewer needs to verify that a conducted Business Continuity Analyses is complete, coherent, current and covers all critical business functions. Every stakeholder has a preferred environment in which he or she wants to review analyses results. Business Process Analyst and IT architect prefer to work with analyses results in their respective modelling and editing environment, whereas Line-of-Business manager and external reviewers prefer office documents, such as Microsoft Excel.
2.4.2 Architecture

In this section and subsequent section we want to give a more detailed description of our architecture. First, we identify involved stakeholders and we introduce the tooling environment our architecture is targeting. Then, we discuss the overall workflow followed by a detailed description of major building blocks in our architecture.

As mentioned earlier, the BCM framework utilizes various models including the IT landscape models, the business process models, etc., and generates a BEAM model, which is consumed by the simulation engine to perform the analysis. Figure 1 gives a high-level overview on our architecture and serves as reference.

![BCM framework architecture](image)

**Figure 1 BCM framework architecture**

The BCM framework uses particular aspects of the SLA@SOI framework: the SLA translation, validation and optimisation package. Figure 2 presents the BCM positioning in the SLA@SOI framework. We used the package to implement an important aspect of Business Continuity Management, the Business Impact Analysis (BIA).
2.4.3 Stakeholders

Business continuity analysis project involves various different stakeholders. Every stakeholder contributes respected domain knowledge, domain business requirements, constraints and objectives to the overall business continuity analysis. Our solution supports Multi-Stakeholder analyses by providing user interfaces and tooling, modelling and analysis support suited for every stakeholder. We anticipate five stakeholders which are involved in an IT Business Continuity Analysis and who will use our tool. These roles are the Business Process Expert, IT Architect, the Business Continuity Analyst, the Line of Business Manager and the External Reviewer. The Business Process Expert designs and model business processes. He/she is responsible to make sure that all business process activities are executable and perform according to the stated requirements. The Line of Business Manager/Process Owner is interested in the overall business process performance. He/she defines process performance indicators, which are aligned with the overall business strategy of an organisation. He/she has to be assured that a business is able to deliver products and services even in case of disruptions.

The task of an External Reviewer is to validate that an organisation has sufficient business continuity competency. The External Reviewer has to make sure that a Business Continuity Analysis is coherent, complete, current and accurate. The analysis should contain all critical and important business processes, functions, and dependent resources and potential risk. The External Reviewer also has to validate that planned responses are appropriated and are sufficient to circumvent or to mitigate adverse effects.

The primary concerns of the IT Architect are IT landscape models. IT landscape models are a set of hardware, software and network elements, arranged in a specific configuration, which serve as a fabric to support the business operations.
of an organization. The IT Architect has to make sure that the IT deployment layout is suitable to support an organizations business and it is capable to meet business requirements, such as business continuity requirements.

The Business Continuity Analyst manages and drives the business continuity analysis work. He/she is responsible for determining critical business processes, critical resources, risk to IT elements and business processes and developing appropriate risk response strategies. The Business Continuity Analyst has to communicate analysis results to all involved stakeholders and to convey change request to the Business Process Expert and IT Architect.

2.4.4 Environment

Our BCM toolkit is completely a generic solution and can be plugged to any process modelling environment. Here in this deliverable, we will explain the integration of the BCM toolkit within NetWeaver Business Process Modelling Environment (NW BPM), which is SAP solution to design, model, document, and design business processes.

NetWeaver BPM supports process modelling based on the BPMN standard from OMG. NW BPM provides two editor tools, namely the NW BPM Process Composer (Process Composer) and the NW Business Process Scenario Editor (Scenario Editor). These tools are used by the Business Process Expert to define business process scenarios and to model business processes. The Process Composer tool provides graphical modelling of activity flows using Business Process Modelling Notation (BPMN). Scenario Models are used to organize a collection of business processes and provide an end-to-end view on business processes. Both tools operate on a set of models (NW BPMN and NW Scenario Model), which are grouped into the NW Common Process Layer model set. Figure 3 depicts the relationship of Process Composer, Scenario Editor, Business Process Expert and the related model artefacts.

Besides the aforementioned modelling tools NW BPM provides tooling support to create other business process related artefacts, such as business rules tables. However, as stated previously, NW BPM provides does not provide means to add business continuity related information nor does it offer any BCM related analytics.

2.4.5 Workflow and Methodology

A Business Continuity Analysis Project is managed by the Business Continuity Analyst and is conducted in five phases; that are the Business Process requirement Analysis, IT BC Model Derivation, BEAM Derivation, Business Continuity Analysis, and Analysis Result Presentation.

Business Process Requirement Analysis: The first phase is the Business Process Requirement Analysis phase, in which the Business Process Expert, in cooperation with the Line of Business Manager, determines business process relevant requirements. The output of this phase is the Business Process Annotation Model (BPAM).

IT BC Model Derivation: In the second phase, the Business Continuity Manager determines resources and depended resource needed to support a business process. This phase involves the IT Architect, which provides the needed domain knowledge and IT models. As stated earlier, it is crucial to generate a consolidated and coherent view of all involved business processes and IT related resources to conduct an appropriate dependency and risk analysis. Our approach utilizes Model-to-Model (M2M) transformations to generate a BC model from
existing models, i.e. Process Models and IT Topology Models. Thus existing models are utilized to provide a profound modelling base to the Business Continuity Manager. However, these models are incomplete and often disconnected from the BCM point of view. Hence our framework provides graphical tools to refine these BC models and to connect model elements. For instance, the user is able to add business continuity related information and connect different resource dependency graphs. The result of this phase is an IT BC Model.

**BEAM Derivation:** In the third phase, the Business Continuity Analyst connects IT BC Models with business process models and adds behaviour information (either manual or in an automated fashion). M2M transformation is executed to transform the IT BC model into a Behaviour and Analysis Model (BEAM) model. Here again the Business Continuity Manager is provided with an editor, the so-called BEAM-Editor. This editor permits the user to further refine the BEAM model, alter the behaviour models, add measurement models and recovery plans if needed. The BEAM model is transformed into input format for the analysis tools with the help of Model-to-Text (M2T) transformations.

**Business Continuity Analysis:** Once the BEAM model is complete, the BCM Manager can trigger an analyses run just by pushing a button. Our solution supports various multi-paradigm analysis tools, such as analytical or simulation tools, to yield accurate and complete results. The analyses results are stored in Analysis Results Models.

**Analysis Result Presentation:** We provide two analysis results presentation mode, the document oriented presentation mode and the context sensitive presentation mode. The document oriented presentation mode generates various types of office documents for the LoB-Manager or External Reviewer. The context sensitive presentation mode is an essential additional tool which provide analysis results to the IT Architect or the Business Process Expert embedded in their respective working (modelling and tooling) environment. The result of the analyses helps stakeholders to understand how a broken resource affects depending resources, how a failure propagates through the system and when eventually a failure impacts the business process. They are able to validate recovery plans as it can be decided if the average response time of a recovery plan is sufficient enough to meet the business level KPIs or if the residual risk is still too high and the recovery plan needs to be revised to meet the business level KPIs.
Figure 3 Business Process Requirements Annotation

To capture business requirements our solution provides a non-intrusive model-driven mechanism to annotate existing business process models with supplementary information. This enables the Business Process Analysts to capture business process requirements, constraints and objectives needed to conduct a comprehensive, coherent and thorough Business Impact Analysis. Figure 3 depicts the major building blocks needed for this activity, which are Business Process Annotation Editor (View) and Business Process Annotation Models.

The Business Process Annotation Editor uses NetWeaver Developer Studio Platforms extension point mechanism to observe selection changes events emitted from Process Composer. Selection change events are emitted when the Business Process Expert selects a business process activity in the Process Composer. These events are carried as payload information from the business process model. This means, if the Business Process Expert selects a single process element, for example a process activity, the Business Process Requirement Annotation Editor becomes aware of the selected element. The Business Process Expert enters requirements related to this selected process element using the Business Process Requirement Editor. Business requirements and references to process elements are stored in the Business Process Annotation Model (BPAM).

2.4.6 IT BCM Model Derivation

IT Business Continuity (BC) Models: These are a set of domain-specific “front-end” models in our model ecosystem. IT BC models lay the foundation to extend existing business process management tools with business continuity
management support. IT BC models cover the resource dependency and risk modelling aspects of Business Continuity.

The Business Continuity Analyst has two options to create IT BC Models. The first option is to use the IT BCM Model Editor to create such a model completely manually. This requires that the Business Continuity Manager has very detailed knowledge of IT landscape elements, configuration semantics, and deployment options. Usually a Business Continuity Manager lacks this detailed knowledge. The second option is to generate IT BC Models from existing IT deployment topology models. This second option has several advantages. First, such an automated generation releases the Business Continuity Manager from any manual effort, which is time consuming. Second, the resulting IT BCM models are complete, coherent and of better quality as human errors are eliminated.

To enable automated derivations of IT BC models additionally a Mapping Model is required. A Mapping Model maps - on meta-model level - IT topology model classes to IT BC model classes. The tool that the Business Continuity Manager uses for this definition is called Mapping Model Editor.

IT deployment topologies are modelled in different model languages. For example, SAP uses the Common Information Model. IBM utilizes a proprietary modelling language to model deployment layouts of WebSphere application servers. Due to our meta-model based mapping approach our solution is able to process all model based IT deployment topologies designs. All it takes is to create a Mapping Model for the respective IT topology model language.

Automated M2M transformation agent transforms the IT Topology Model into an IT BC model using the mappings defined in the mapping model. This process is depicted in Figure 4.

Once the transformation agent creates an IT BC Model instance, the Business Continuity Analyst can examine, refine and finalize this model instance by using the IT BC Model Editor. He/she can add supplementary information to the IT BC model, which is not part of an IT landscape deployment topology. Threats, failure modes, and responses to risk are examples of such complementary information.

![Figure 4 Generation of IT BC Model](image)

**2.4.7 BEAM Model Derivation**

A BEAM focuses on the behaviour, performance and analysis modelling aspects solely. This meta-model refines the BC model with behavioural information of
resources and dependencies. Behaviour modelling is also used to detail recovery plans and measurement models. Measurement Models are meant for defining BCM KPIs for resources and business processes alike, such as Return Time Objective.

BEAM models are derived by the merging of IT BCM models and business process models into a single, unified analysis model. It takes three steps to derive a complete BEAM model. In a first step business process model is transformed into a (inchoate) BEAM model. We call this Alpha BEAM. The second step transforms IT BC Model into a BEAM model that we named Beta BEAM. The last and final step merges Alpha and Beta BEAM into the final BEAM model. The final BEAM model is called Gamma BEAM. The whole process is orchestrated by an agent (not depicted), which co-ordinates all transformations.

2.4.8 Alpha BEAM

As stated previously, IT BC models solely define dependencies between various resources. They do not define how these resources behave or influence each other in case of disruptions. This behavioural information needs to be added.

The Business Continuity Manager could add behavioural information to IT BC model elements manually. As a matter of fact this is supported in our solution. However, for large IT BC models this would require a lot of manual work which is cumbersome and error prone. Moreover, if the IT architecture changes, the Business Continuity Manager has no other option than to discard his previous work and to re-start the whole BEAM construction process again.. This is not an acceptable solution in environments where IT deployments are subject to frequent changes, such as Cloud Computing based deployments. In order to enrich IT BC models with behavioural information in an automated fashion our solution employs IT BC Library Models. The Business Continuity Manager defines IT BC Library models using a dedicated IT BC Library Editor. These library models map IT BC model elements types to pre-defined BEAM behaviour model elements. For example the IT BC model element type Server would be mapped to a set of behaviour states such as Off, Booting, Running or Shutting Down and related state transitions.

![Figure 5 IT BC Library Models](image)

Our approach reduces the manual effort to model behaviour information to a minimum in two ways. First, the Business Continuity Manager needs to model behaviour information only once for every IT BC model element type, e.g. for the type Server. This behavioural information is then applied by the Alpha-BEAM M2M agent automatically to all occurrences of the same model element type in the IT BC model. Second, the Business Continuity Manager has to define IT BC Library Model only once. Once such library models are available the whole process can be repeated as many times as needed, with no human interaction required. This
enables automated Alpha-BEAM model creations, which is required for frequently and dynamically changing IT landscapes.

As we aim to analyse business continuity on business process activity level, it is very important to identify resources needed by business process activities and establish a link among resources and activities. This is done by the Business Process Expert using the Business Process Requirement Annotation Editor (see Figure 6). The Business Process Expert establishes references from business process model elements to IT BC resources. These mappings are stored in the BPAM model.

Once all required models are prepared we are able to derive the Alpha-BEAM model. The Alpha-BEAM M2M transformation agent takes as input IT BC Models, IT BC Library Model and BPAM model and transforms all models into an Alpha BEAM model. Furthermore the transformation agent produces a tracing model. The need for the tracing model is explained later. The automated Alpha-Beam creation process is depicted in Figure 7.

**Figure 6 Business Process Requirement Annotation Editor**
This Alpha-BEAM model now contains all resources, dependencies among resources, default behaviour models and references to business process activities. However, other important details of business processes, such as connections, gateways, etc are still missing.

2.4.9 Beta BEAM

The purpose of a Beta-BEAM model is to consolidate different parts of a business process model spanning multiple process modelling and execution environments into one single end-to-end business process model. For instance, NetWeaver BPM models do not cover the whole business process rather they generally represent only the extended/customized part of the process, which is more specifically the front-end extension of the back-end ERP processes. In order to enable end-to-end Business Continuity Analysis our solution consolidate all available process models that belong to an end-to-end business process scenario in a single BEAM Model, the Beta BEAM Model. This model comprises the complete business process with all process activities, gateways, path connections, etc. Moreover, all business requirements, documented in the BPAM, are merged into the Beta-BEAM as well. As depicted in Figure 8, a model-to-model transformation agent, the Process to Beta BEAM agent, takes the business scenario model, all related business process models, the BPAM, and produces a single Beta-BEAM model. This model contains all business process elements of the complete end-to-end business process and related business requirements.
2.4.10 Gamma BEAM

The Gamma BEAM M2M transformation agent merges Alpha and Beta BEAM Models into a combined BEAM model, the Gamma BEAM model. The agent also resolves missing references and sets missing default values if appropriated. The Gamma BEAM is the final BEAM model. It contains all necessary elements for Business Continuity Analysis from both modelling domains, the business process modelling domain as well as from IT landscape modelling domain. This process is shown in Figure 8. The Business Continuity Manager may want to refine the Gamma BEAM model further. For example, the manager may want to detail recovery plans or modify resource behaviour. For altering BEAM models we provide the BEAM Editor.

![Figure 9 Gamma BEAM](image)

2.4.11 Business Continuity Analysis

The final Gamma BEAM model serves as an input for various analyses and evaluation tools. As depicted in Figure 10, the Business Continuity Manager uses the Business Continuity Analysis Controller to select the type of tool based on the type of analysis to be executed. This information is written in tool specific configuration storage, for example a property file. The Controller also commands the tool to execute the analysis run. Every analysis tool involved in an analysis run writes its result into analysis result storage.

2.4.12 Analysis Result Presentation

Analysis results are usually not in a presentable format and generally contain some irrelevant information. Therefore we provide BC Reasoner. BC Reasoners act as filter and pre-processor which transforms analysis results into Business Continuity Analysis result Model (BCAM). Moreover, some results from heterogeneous analysis tools are only meaningful if they are correlated and interpreted in one context. For example, an analytic tool is able to compute the worst case and the best-case execution time for a recovery plan. On the other hand, a simulation tool is able to predict the average execution time of the same recovery plan. To decide, if the recovery plan needs to be improved, all of these execution time values are required.
The task of a BC Reasoner is to bring together results from heterogeneous, multi-paradigm analysis tools and to compute sound and relevant Business Continuity results. These results are stored in BCAM storage as well. This post-analysis filtering and reasoning process for a simulation tool is shown in Figure 11.

Our solution provides two modes to present analysis results to all stakeholders, that is (1) an interactive and context sensitive presentation mode and a (2) document oriented presentation mode.
2.4.13 Tracing

In order to provide context sensitive results and to relate analyses results with source model elements (such as the business process activity model element) links are preserved via trace models across the whole transformation chain. Every transformation agent is tracing enabled, thus produces a tracing model. This tracing model preserves a mapping from source model elements (the input to the agent) to target model elements (the output of the transformation agent).

![Figure 12 Trace models in the architecture](image)

Trace models enable to compute the transformation path from analysis results back to the originating source model element (as shown in Figure 12). However, to simplify this computation we consolidate all trace models first. A trace model consolidation agent merges Alpha, Beta and Gamma Trace models and produces a consolidated BEAM Trace model (Figure 12).

2.4.14 Context Sensitive Presentation Mode

One major objective of our solution is to present analysis results to stakeholders in their respective modelling environments. We call this the Context Sensitive Analysis Result Presentation (or interactive presentation for short). For example, the expected Recovery Time Objective of a business process activity should be displayed to the Business Process Expert if he/she selects a process activity in his/her modelling environment. This view is provided by the Context Sensitive Analysis Result View. The Context Sensitive Analysis Result View uses the same mechanism as the Business Process Requirement Editor to detect model element selection changes in NW BPM Process Editor. On selection change events, the Context Sensitive Analysis Result View displays analysis results related to the selected model element. For example, if the Business Process Expert selects a certain business process activity, the view will display analysis results for this specific process activity only.
The interactive presentation mode has various advantages. First, it allows stakeholders to visualize analysis results in their modelling domain. They are able to change the models accordingly (e.g. alter a business process or an IT landscape model) and get immediate feedback about how changes affect business continuity aspects of a business process or IT landscape. The architectural details of the Context Sensitive Analysis Result View for the Business Process Expert are shown in Figure 12. The Context Sensitive Analysis Result View for the IT Architect works similar.

Not all stakeholders have a dedicated modelling domain with sufficient tooling support, such as NW BPM. Moreover, in some cases it is required to disseminate analyses results to external stakeholders with no access to appropriate modelling tools. This is particularly true for the BCM auditors. Therefore our solution provides an additional presentation mode, the document-oriented presentation mode. In this mode, Report Generators transform analysis results into various office documents. A Report generator takes as input BCAM models and other information sources, such as business process models, and generates BC analysis documents in various formats, e.g. Microsoft Excel documents.
Our solution also supports generators chaining. For example, the xCelsius Report Generator re-uses reports generated by the Microsoft Excel document generator and produces SAP xCelsius Dashboards. The Business Continuity Manager controls the document generation process via the BIA Analysis Controller. The BIA Analysis Controller orchestrates and configures Report Generators and Report Generator chains by means of configuration files.

2.5 Conclusions and Outlook

As mentioned in the start of the section, SLA management is more than pure management of stack. There is a huge additional potential to leverage the approach of the SLA@SOI framework for supporting environments where services are actually delivered. Business continuity supports management of such extended environments and therefore goes beyond the pure management of IT stack but also includes the operations at a data centre including cooling, support staff and many other aspects. We have extended the ERP B3 framework with a process-centric Business Continuity Analysis toolkit, which can be used in SLA@SOI, for example, for SLA definitions and translations, service evaluation and for service management. We have described in this section how this BCM toolkit consumes already available business process models and IT landscape models to automatically uncover dependencies among the resources in different layers of the enterprise stack. Within the transformation chain, the Business Continuity Manager is further provided with two editors to further refine the models, the BCM-Editor and the BEAM-Editor. The availability of these editors helps enhance the quality of the resulting analysis model in terms of completeness, accuracy and precision. The BCEditor allows the manager to assign resource dependency graphs to business process activities, and to further enhance the knowledge captured about the resource dependencies within the automatically generated resource dependency graphs by adding his/her expert knowledge. Once the BC model is transformed into the BEAM model, with the help of BEAM-Editor the user is able to further refine the BEAM model, alter the behavioural models, and add measurement models and recovery plans. The proposed architecture allows connecting to various different types of analysis tools.

In the future SAP plans to work together with Intel in another EU project, named TIMBUS, to further enhance the existing BCM framework. Intel has introduced the Enterprise Capability (EC) Framework in Deliverable D.B4c. The EC Framework provides a structured, high-level view on organisation functions, for example
Sales and Marketing. The BCM framework also provides a high-level view on an organisation, but on operations in a process-centric way. Business processes span multiple organisational functions. For example, a Sales and Distribution Process cross-cuts various functional units including Sales, Marketing, Production and so forth. We plan to converge both approaches into a unified BCM/EC framework. We believe that a converged approach is capable of offering a better insight of business requirements and will further enhance SLA@SOI framework’s SLA translation functionality.
3  Service Manageability

3.1 Overview

During SLA@SOI's second year we developed a unified manageability interface that the project's main platform could use to manage service instances so that their behaviours were aligned to the terms negotiated in an SLA. Through this manageability interface it became possible to configure service instance sensors, for gathering runtime information that were useful for analysis, and service instance effectors, for performing some sort of adaptation on the running instance. During the project's second year we concentrated on developing sensors and effectors for manageable BPEL processes.

In the project's third year we have continued to develop manageability techniques for BPEL processes. The main contributions are: (i) the support for event correlation through a plug-in for the Manageability Agent, (ii) an algorithm for process restructuring that attempts to maximize the parallelization of a BPEL process' set of internal activities, (iii) an extension to the open-source ActiveBPEL engine that allows for the runtime adaptation of a process' control flow, and for enacting any new parallelized versions of the process the restructuring algorithm may have produced, and (iv) an extension of the DOE's manageability agent façade with support for setting up event correlation, multi-party binding rules and the dynamic adaptation of a process' flow.

The rest of this section is organized as follows. First we will present the support for event correlation in the Manageability Agent. Second, we will show the process-restructuring algorithm. Third, we will illustrate how we extended ActiveBPEL to support the dynamic alteration of a process' control flow, and how the unified manageability interface from Y2 can be used to call upon all these features. Finally, we will conclude with a look at the state of the art.

3.2 Support for Event Correlation

Figure 15 shows the Manageability Agent and how it fits into the overall SLA@SOI framework. The Manageability Agent allows the SLA@SOI platform to interact with and manage service instances that have been implemented using different domain-specific technologies. In order to manage a service it is required that the service allows the configuration of appropriate sensors, for capturing runtime data that can be of interest for monitoring, and of appropriate effectors, for adapting the service so that it remains aligned with the SLA. The cardinality of the manageability agent with respect to its managed service instances is 1 to n, meaning that a single manageability agent can take care of many specific service instances at a time.

The figure has been adapted, with respect to a similar figure that can be found in the project's second year deliverable, to support event correlation and manipulation. This is used to correlate and manipulate sensor data to produce higher-level data that can be useful for monitoring the system's behaviour with respect to the SLA.
The main novelty is that now the ManageabilityAgent can subscribe to events that go through the bus, and publish new ones. This allows the correlated data to be produced in a pipe-and-filter fashion. The correlations are implemented as correlators that are plugged into the ManageabilityAgent and implemented using Esper Complex Event Processing technology. They are added to the Manageability Agent so that the sensors, usually deployed in strict contact with the actual service instances being managed, can concentrate on obtaining the simple runtime data that are requested from them.

Figure 16 illustrates the different kind of correlators we currently provide. We support three kinds of Key Performance Indicator Aggregates out of the box: Avg Response Time, Reliability, and Rate. Avg Response Time calculates how long it takes a given service call to complete. To do so it subscribes to two different kinds of events on the bus: the time-stamped requests to a given service and its time-stamped responses. It then correlates the requests and the responses, calculates the time elapsed between corresponding couples, and adds the results to the bus for further consumption by actual reasoners. Reliability calculates the number of correct interactions with a service over the total number of interactions attempted (more concrete details on how this is actually achieved can be found in the A6 deliverable, in which we describe what Reliability means in the context of...
you manageability models). When using a Reliability KPI we must also state (i) over how much time it has to be calculated, and (ii) how often its output value needs to be made available. For example, we might want to calculate a service’s reliability considering the last 12 hours, and output a new value every 5 minutes. This is achieved through its periodUnit and periodValue, and outputUnit and outputValue attributes respectively. Finally we have the Rate of request arrivals. It also requires that we specify periodUnit, periodValue, outputUnit, and outputValue attributes. We also support domain-specific correlators, provided they have a corresponding definition given in Esper’s event correlation language.

3.3 Process restructuring

Our approach is based on a static analysis of a structure of a process encoded in BPEL. Namely, the BPEL syntax is parsed and its activities are enveloped in objects. The latter are organized in parent-child manner w.r.t. control flow in the original BPEL file. This structure is a directed acyclic graph that may be seen as a model of the business process independent on BPEL syntax.

The process restructuring algorithm is applied to this graph taking into an account a set of dependencies which are identified at the stage of graph analysis: data-flow, partner protocol, correlation sets. The dependencies cannot be broken. Otherwise, the resulting process will be incorrect, inconsistent and faulty. On the other hand, the absence of any dependencies between a pair of parent-child nodes gives us a possibility to place them in parallel or rearrange in some other way. Thus, the algorithm attempts to maximize the parallelization of BPEL activities by rearranging the edges in the graph model.

First, we will discuss the directed graph model adopted by the restructuring algorithm. Second, we will show how to produce the corresponding graph model of a BPEL process. Third, we will discuss how data flow dependencies and partner protocol dependencies influence the restructuring algorithm. Fourth, we shall give a formal description of the restructuring algorithm, and illustrate on a simple healthcare and mobility booking example. Finally, we give some hints how the resulting graph can be transformed back into a BPEL process.

3.3.1 BPEL Graph

We model a business process encoded in BPEL as a BPEL graph, which is a directed graph, according to the following definition.

**Definition 1.** A **BPEL graph** is a pair \( G = (N, E) \) where \( N \) is a set of nodes corresponding to BPEL activities and \( E \subseteq \{N \times N\} \) is a set of directed edges.

Each node \( n \in N \) is a tuple \( <\pi, \chi, I, O> \), where \( \pi \) is a partner link, \( \chi \) is a correlation set name (if exists), \( I \) is set of inputs, and \( O \) is set of outputs. Nodes may be either simple or complex, corresponding to primitive or structured BPEL activities respectively. A complex node \( \bar{N} = (N', E') \) is a BPEL graph with a set of nodes \( N' \), and a set of edges \( E' \subseteq \{N' \times N'\} \) such that \( N \cap N' = \emptyset \) and \( E \cap E' = \emptyset \).

A partner link of a simple activity is defined in its attribute partnerLink.
A correlation set is defined as a child tag \(<\text{correlation set="}\chi\text{"/>\).

A directed edge \( e \in E \) from node \( n \) to node \( m \) (denoted as \( e = (n, m) \)) exists in the BPEL graph \( G \) if an execution of an activity corresponding to \( n \) is followed by an execution of an activity corresponding to \( m \). In other words, edges encode
those control flow requirements that are specified with sequence construct. The node \( n \) is called a parent of a node \( m \), and \( m \) is called a child of \( n \).

If the node \( a \) has several children \( b_1, b_2, ..., b_m \) it means that \( b_i, i = 1, ..., m \) can be executed in parallel after the execution of \( a \). In other words, multiple children encode those control flow requirements that are specified with flow construct.

If the node \( b \) has several parents \( a_1, a_2, ..., a_m \) it means that \( b \) can be executed only when execution of all \( a_i, i = 1, ..., m \) is concluded.

A node that has no parents is called root and is an entry point of the business process. A graph \( G \) may have one or more roots.

Note that the specification of constructs sequence and flow is embedded directly into the graph structure in an unfolding manner. In other words, if we have a sequence containing several sequence constructs, they will be represented as a single chain of graph edges. A flow of several flows (without any nested sequences) will be a union of the content of the latter. A sequence of flows will build a node with multiple children from the first flow having multiple children from the second flow, etc.

### 3.3.2 Translation of BPEL to BPEL Graph

In order to distinguish among various components of each particular node \( n \), we use the bracket-notation. Namely, \( \pi(n) \), \( I(n) \), \( O(n) \) denote correspondingly the partner link, the inputs and the outputs of node \( n \).

In Table 3, we show how each particular BPEL activity can be converted to a fragment of BPEL graph of the whole business process according to Def. 1. In the first column we report a fragment of activity specification which is important for our formalization, i.e. what should be taken into consideration during parsing of activities in the original BPEL file. In the middle column a graphical representation is provided. Finally, a formal BPEL graph translation is given in the last column according to the Def. 1.

If the node is primitive, its input-output arguments are taken from corresponding attributes of the activity according to Table 3. If the node is complex, its input arguments \( I(\overline{n}) = \bigcup_{n \in N} \{I(n)\} \) and output arguments \( O(\overline{n}) = \bigcup_{n \in N'} \{O(n)\}, \) where \( N' \) is a set of nodes of the complex node \( \overline{n} \).

To be more precise, all primitive BPEL activities, i.e., invoke, receive, reply, assign correspond to a BPEL graph consisting of a single BPEL node with partner link and arguments extracted from the original BPEL in a trivial way (see Table 3). A set of edges in such graphs is empty.

We would like to draw a particular attention to activity assign which may have input and/or output arguments populated from some expression (e.g., XPath). Dealing with expression is a bit tricky, since the expression may be too complex. For the moment, we limit ourselves to detecting expression tokens of the form \$\text{variable} \text{or } \$\text{variable.part}, i.e. \text{variable} must follow symbol ‘$’ and may be followed by part after ‘.’ symbol.

<table>
<thead>
<tr>
<th>Table 3. Mapping BPEL constructs to BPEL graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragment of BPEL</td>
</tr>
<tr>
<td>Fragment of BPEL</td>
</tr>
<tr>
<td>construct</td>
</tr>
<tr>
<td>-----------</td>
</tr>
</tbody>
</table>
| `<invoke` partnerLink="pName" portType="QName"? operation="NCName" inputVariable="iName"? outputVariable="oName"? />` | ![Invoke Diagram](image1) | $G = (N, E)$  
$N = \{n\}, E = \emptyset$  
n $\rightarrow pName, iName, oName >$ |
| `<receive` partnerLink="pName" portType="QName"? operation="NCName" variable="oName"? createInstance="yes\|no"? messageExchange="NCName"? />` | ![Receive Diagram](image2) | $G = (N, E)$  
$N = \{n\}, E = \emptyset$  
n $\rightarrow pName, \emptyset, oName >$ |
| `<reply` partnerLink="pName" portType="QName"? operation="NCName" variable="iName"? faultName="QName"? messageExchange="NCName"? />` | ![Reply Diagram](image3) | $G = (N, E)$  
$N = \{n\}, E = \emptyset$  
n $\rightarrow pName, iName, \emptyset >$ |
| `<assign>` `( <copy>`  
`<from` variable="iName" part="partName"?> `</from>` | ![Assign Diagram](image4) | $G = (N, E)$  
$N = \{n\}, E = \emptyset$  
n $\rightarrow \emptyset, iName, oName >$  
$n \leftarrow \emptyset,$  
`getVariables(from-exp),`  
`getVariables(to-exp)` |
The arguments detected in the way described above are tackled as inputs or outputs depending on their declaration, i.e. if they are specified in either element from or element to respectively. For extracting variables from expressions, we introduce a function getVariables which is applied expression declared in a corresponding element of BPEL file. The result of this function is a set of arguments. Thus, the second variant of assign construct is depicted with multiple inputs and multiple outputs.

Expressions are also used in complex activities if and while. Namely, they are declared as a bool-exp of condition element in the original BPEL graph. We apply to bool-exp the function getVariables explained above in order to extract arguments that will be consumed by activities in the following graph. Thus, these activities are considered to be input arguments for graph.

Complex activities pick and if are constituted of several subgraphs with mutually exclusive execution. In other words, these subgraphs correspond to execution branches defined by conditions which may be evaluated only during run-time. According to the evaluation, only one branch will be executed, while the execution of the other branches will be excluded. Thus the subgraphs corresponding to
branches are mutually exclusive. In Table 3, we denote a graph with mutually exclusive subgraphs via symbol ‘X’. In such a graph, the subgraphs cannot be considered as children of some node, since, as we said above, several children with the same parent are considered to be all executed in parallel. However, we place nodes corresponding to these subgraphs as items of a container corresponding to complex activity (pick and if). Since mutually exclusive subgraphs may be one-many, we denote them via ‘+’ symbol.

Element onMessage in BPEL specification has only one attribute variable. However, we use it both for input and output argument of onMessage node in the BPEL graph formalization. The reason is that variable is populated from one of the previously executed activities invoke. Thus it may be considered as an input for onMessage. Further, this input parameter may be consumed by the activities in the following graph. Therefore, variable can be considered as an output parameter as well. Thus in the BPEL graph corresponding to onMessage element, a set of nodes include the node n_onMessage and nodes of the graph. The node n_onMessage is a parent node for the roots of the graph. In other words, a set of edges include edges (n_onMessage, root(graph)) and the edges of graph, where root(graph) is a set of roots of graph.

Finally, the complex node corresponding to the activity while consists of the nodes and edges of graph with input arguments extracted from bool-expr by means of previously introduced function getVariables.

Thus, each particular BPEL activity is transformed in a corresponding node of the BPEL graph. Afterwards, the nodes are connected via BPEL edges. The overall algorithm of edge construction is show formally below.

\begin{algorithm}
\textbf{Algorithm} BPEL PARSING
\begin{algorithmic}[1]
\Input{A set of BPEL activities $A$ both simple and complex, a BPEL graph $G = (N,E)$}
\Output{An BPEL graph $G = (N,E)$ with additional elements}
\ForAll{activities $\alpha \in A$}
\If{$\alpha$ is a simple activity}
\State identify $\langle \pi(\alpha), I(\alpha), O(\alpha) \rangle$;
\Else /// $\alpha$ is a complex activity
\State BPEL PARSING($\alpha.getChildren()$, $G$);
\State let $c_1, c_2, \ldots, c_n$ be $\alpha.getChildren()$;
\If{$\alpha$ is sequence}
\State $E = E \cup \{ (\alpha, c_i) \} \cup \{ (c_i, c_{i+1}) \}$ for $i = 1, \ldots, n-1$
\ElseIf{$\alpha$ is flow}
\State $E = E \cup \{ (\alpha, c_i) \}$ for $i = 1, \ldots, n$
\Else
\State create a node for a complex activity $C$;
\State $I(C) := I(\alpha.getChildren())$
\State $O(C) := O(\alpha.getChildren())$
\EndIf
\EndIf
\State $N = N \cup \{ \alpha \}$
\EndFor
\State return $G = (N,E)$
\end{algorithmic}
\end{algorithm}

The algorithm BPEL PARSING also includes parsing of BPEL file. Since BPEL is a dialect of XML, we apply a simple DOM-parser which constructs a DOM-tree. We navigate that tree in DFS order starting from the most outer complex node, e.g., sequence, flow, pick etc. Elements of this node may be both simple activities
and other complex nodes. If they are complex nodes, among their child nodes (in terms of XML-tree) there are sub-activities which can be also simple or complex. Thus we use an expression `node.getChildren()` to obtain such subactivities from the DOM-tree of the BPEL-file and the parsing proceeds recursively for the children. Finally, the nodes that do not have parents are marked as the roots of the constructed graph $G$.

### 3.3.3 BPEL Graph Analysis

In our approach for structural adjustment, we pursue the principle of maximum parallelization of the original workflow, i.e. our restructuring algorithm is intended to parallelize as many operations as possible. However, the parallelization should take into account several conditions that will render the resulting graph correct and consistent with respect to data flow and conversation protocols. Namely, the BPEL graph, before it undergoes a restructuring process, should be analyzed and the following dependencies should be identified:

- data-flow dependency;
- partner protocol dependency;
- correlation dependency.

We encode dependencies in dependency rules.

**Definition 2.** Dependency rule is a binary relation $a < b$, called dependency relation, between two graph nodes $a$ and $b$. Dependency relation $a < b$ establishes that activity $a$ must be executed before $b$. The left part of the rule is called a precondition for the right part of the rule.

Dependency relations establish a strict partial ordering over the nodes of the graph $G=(N, E)$. In other words, dependency relation is a binary relation that is irreflexive and transitive, and therefore asymmetric. More formally, if $a, b, c \in N$, we have the following properties to be held:

- **irreflexivity:** $\neg(a < a)$;
- **assymetry:** if $a < b$ then $\neg(b < a)$; and
- **transitivity:** if $a < b$ and $b < c$ then $a < c$.

**Data-flow Dependency**

**Definition 3. [Basic data-flow dependency]** We say that a node $n \in N$ is data-dependent on a node $n' \in N$ if $\exists o' \in O(n')$ and $\exists i \in I(n)$ such that $o' = i$. In other words, some output of $n'$ is an input for $n$.

Since arguments may have parts, the dependability by data flow can be split and merged among several activities.

Consequently, the data-independent nodes do not have any dependencies between their input-output arguments. Therefore such operations can be executed in parallel. This principle of parallel execution of data-independent activities is the basis for our restructuring algorithm.

We would like to note, that assign specification is more extended w.r.t. Table 3. Namely, from and to may be populated also from a literal. It is clear that literal (or in other words constant) does not create any dependency and thus can be skipped as an argument in our model.
We distinguish arguments (with parts) by their names. If the BPEL file is constructed in such a way that no argument rewriting is allowed, i.e., the argument can be output only once, we don't have any ambiguity. In other word, with this assumption we can uniquely identify an activity that produces some value and an activity that further consumes that value. However, the real BPEL processes usually contain a lot of argument rewriting. To deal with them, we introduce the notion of data segments.

**Definition 4.** Data segment is a tuple \( DS = < aName, n_0, \{ n_1, \ldots, n_s \}, \{ DS_1, \ldots, DS_q \}> \), where \( aName \) is an argument name, \( n_0 \) is a graph node where argument is initialized, set \( \{ n_1, \ldots, n_s \} \) consists of graph nodes where argument \( aName \) is consumed without modifications, \( \{ DS_1, \ldots, DS_q \} \) are data segments where the argument \( aName \) is rewritten.

In the following, the data dependency is checked not w.r.t. arguments but w.r.t. data segments. In other words, data segments specify a finer granularity of the original argument set. And the dependency rules over data segments are managed as follows.

**Definition 5.** [Data segment flow dependency] Given a data segment \( DS = < aName, n_0, \{ n_1, \ldots, n_s \}, \{ DS_1, \ldots, DS_q \}> \), define data-flow dependencies among \( n_0 \) and \( \{ n_1, \ldots, n_s \} \) according to the basic data-flow dependency definition. Then for each \( DS_i, i=1, \ldots, q \), \( n_0 \) is dependent from \( n_{0i} \).

**Partner Protocol Dependency**

The communication with business-process stakeholders is held via partnerLink attribute of some simple activities, e.g., invoke, receive, reply, and a complex activity onMessage. In the original BPEL file, all such activities are strictly partially ordered w.r.t. each particular partner link. The total order takes place when the business process is sequential with no parallel and conditional branches.

**Definition 6.** Suppose that \( A = \{ a_1, a_2, \ldots, a_n \} \subseteq N \) is a set of all activities of graph \( G = (N,E) \) that have the same partner link \( \pi \). A partner protocol for a partner with a partner link \( \pi \) is a strict partial order over set \( A \), where the binary relation of ordering \( a < b \) means that activity \( a \) must be executed before \( b \).

The partner protocol dependency is extracted from the original BPEL file during its parsing in the DFS order.

We assume that the process party protocols should be preserved also in the absence of data-dependency.

**Correlation dependency**

In our approach, to some extent we also consider dependencies among elements of correlation sets. Namely, correlation sets are used among messaging activities, i.e., receive, reply, invoke, onMessage with the purpose to maintain a correct message routing during a conversation among stakeholder for each particular process instance.

According to the BPEL specification, a correlation set can be initiated only once during the BPEL process. The initializing activity must precede all the other
activities with the same correlation set. Otherwise, the correlation consistency is violated. This means that the BPEL-file incorrectly implements a business process.

The BPEL specification includes the third mode for correlation set initialization which is declared with the key word “join”. This mode means that “the related activity MUST attempt to initiate the correlation set, if the correlation set is not yet initiated.” (citation from BPEL specification).

Thus we can conclude, that the correlation set specifies a sort of correlation protocol that should be respected in each particular instance of BPEL process. Since our approach is based on analysis of structural properties of BPEL process, we define the notion of correlation protocol at the level of BPEL syntax rather than process execution and process instance.

**Definition 6.** Suppose that \( A = \{a_1, a_2, ..., a_n\} \), \( A \subseteq \mathcal{N} \) is a set of all activities of graph \( \mathcal{G}=(\mathcal{N}, \mathcal{E}) \) that have the same correlation set \( \chi \). A correlation protocol for a correlation set \( \chi \) is a strict partial order over set \( \mathcal{A} \), such that there exists only one activity \( a_i \in \mathcal{A} \) that initializes correlation set, while the other activities \( a_j \in \mathcal{A}, j=1, \ldots, n, j \neq i \) must follow activity \( a_i \), which is denoted as \( a_i \prec a_j \). If the activity \( a_k, k \neq i \) initializes a correlation set with "join" mode, its execution must precede the execution of those \( \{b_1, b_2, \ldots, b_m\} \subseteq \mathcal{A} \) that are descendant nodes of the \( a_k \) in the graph \( \mathcal{G} \).

### 3.3.4 Restructuring Algorithm

Given a BPEL graph and a set of dependencies in the form \( a \prec b \), we can run our restructuring algorithm which is depicted below. In the algorithm, we use functions \( \text{child}(a) \), \( \text{parent}(b) \), \( \text{ancestor}(c) \), and \( \text{root}(G) \) which return, correspondingly, a set of children for node \( a \), a set of parents of node \( b \), a set of ancestors of node \( c \), and a set of roots of graph \( G \). Here children, parents, and ancestors are calculated w.r.t. the edges \( \mathcal{E} \). Recall, if \( (a, b) \) and \( (b, c) \in \mathcal{E} \), then \( a \) is a parent for \( b \), \( c \) is a child for \( b \), and \( a, b \) are ancestors for \( c \).

The algorithm is launched for each graph root \( r \in \text{roots}(G) \) and proceeds in DFS as described below.

---

**Algorithm** MAXIMUM PARALLELIZATION

**Input:** A BPEL activity \( \alpha \) either simple or complex, a BPEL graph \( \mathcal{G} = (\mathcal{N}, \mathcal{E}) \)

**Output:** A restructured BPEL graph \( \mathcal{G} = (\mathcal{N}, \mathcal{E}) \)

1: \( \Delta = \text{children}(\alpha) \);
2: **for all** \( \delta \in \Delta \) **do**
3: **if** does not exist any dependency such that \( \alpha \prec \delta \) **then**
4: calculate set \( \Phi = \{\phi \in \mathcal{N} : \phi \prec \delta\} \);
5: calculate set \( \Phi_\alpha = \{\phi \in \mathcal{N} : \phi \prec \delta \text{ and } \phi \in \text{ ancestor}(\alpha)\} \);
6: **if** \( \Phi_\alpha = \emptyset \) **then**
7: **if** \( \Phi_\alpha = \emptyset \) **then**
8: \( R := \text{roots}(\mathcal{G}) \);
9: \( R := R \cup \{\delta\} \);
10: **end if**
11: **else**
12: calculate \( \Phi_\alpha := \Phi_\alpha \setminus (\text{ ancestor}(\Phi_\alpha) \cap \Phi_\alpha) \);
13: **for all** \( \phi \in \Phi \) **do**
14: \( E := E \cup \{(\phi, \delta)\} \);
15: **end for**
16: \( E := E \setminus ((\alpha, \delta)) \);
17: \textbf{end if}
18: \textbf{end if}
19: \textbf{if } \delta \text{ is a complex node then}
20: \quad R_{\delta} := \text{roots}(\delta);
21: \quad \text{for all } \rho \in \ R_{\delta} \text{ do}
22: \quad \quad \text{\textsc{maximumParallelization}(}\rho,G);\text{\textsc{end for}}
23: \quad \text{\textsc{end if}}
24: \text{\textsc{end for}}
25: \text{\textsc{maximumParallelization}(}\delta,G);\text{\textsc{end for}}
26: \text{return } G = (N,E);

For a \textit{currently considered} node \(\alpha\), we may assume that a graph consisting of \(\text{ancestor}(\alpha)\) is already parallelized. It means that all dependencies among the nodes from the set \(\text{ancestor}(\alpha)\) are respected and for all edges \((x,y)\), where \(x,y \in \text{ancestor}(\alpha)\) there exist some dependency \(x < y\) that cannot be broken. Thus, we have to parallelize the graph that consists of nodes \(\text{descendant}(\alpha)\) and corresponding edges from the graph \(G\).

For this purpose, we start from the children of \(\alpha\) (line 1) and analyze if we can parallelize them with \(\alpha\). The parallelization is impossible if a graph dependency rule set contains a dependency \(\alpha < \delta\), where \(\delta\) is a child of \(\alpha\). Otherwise, the edge among \(\alpha\) and \(\delta\) can be eliminated and we have to decide what nodes of the graph will become parents of \(\delta\). For this purpose, in line 5 of the algorithm \textsc{maximumParallelization}, we calculate a set \(\Phi_{\alpha}\) that consists of graph nodes that are preconditions for \(\delta\) and ancestors of \(\alpha\) at the same time. The nodes of the set \(\Phi_{\alpha}\) are considered to be potential parents for \(\delta\). On the other hand, if \(\Phi_{\alpha}\) is an empty set, it means that \(\delta\) does not need any precondition to be executed, and it does not depend on the current parent \(\alpha\). Hence \(\delta\) is a potential graph root. Indeed, it becomes one of the graph roots in lines 7–9, if no other dependency exists.

From the set of potential parents \(\Phi_{\alpha}\) we have to exclude those nodes that are ancestors of potential parents (i.e., among \(\text{ancestor}(\Phi_{\alpha})\) there may be some elements of \(\Phi_{\alpha}\)). In other words, we leave only the nearest potential parents to \(\delta\) which are calculated in line 12. For the nodes from \(\Phi_{\alpha}\) calculated according to the formula in line 12, we create an edge with \(\delta\). Consequently, the edge \((\alpha,\delta)\) is deleted.

If \(\delta\) is a complex node, i.e. it is a graph per se, we have to parallelize its inner content. This is done in lines 19–24 in a recursive manner starting from the roots of the complex node. Finally, the algorithm proceeds recursively for \(\delta\) making it a currently considered node.

\subsection{Example of Parallelization}

To demonstrate the outcome of the restructuring algorithm, let us introduce an intuitive example of health treatment and mobility booking borrowed from B6 e-Government scenario \cite{25}. It’s schematic representation is shown in Figure 17. Namely, patient calls to the healthcare system to assign an appointment for a ticket. Healthcare system operator checks an agenda (\textit{getTreatmentOptions}) for possible appointment options and invites a citizen to select a desired one (\textit{selectTreatmentOptions}). After that, an agenda is updated (\textit{updateAgenda}) and treatment call back is issued (\textit{bookTreatmentCallback}).
Then, the user may decide that she needs a mobility service as well to bring her for the treatment. If this happens, a mobility booking sequence is executed analogously to the treatment booking. First, trip options are requested from mobility providers ($getTripOptions$) which may be multiple (e.g., bus, train, taxi, ambulance, etc.). Then, the citizen selects desired trip options among proposed ones ($selectTripOptions$). Finally, trip options are fixed in an agenda of a mobility provider ($bookTripOptions$).

We may need to parallelize this workflow because of the following reasons:

1. Multiple mobility services are external to the healthcare system. Therefore, it may require a lot of time to collect trip options from them.
2. Let us assume that the patient has mobility issues and that mobility services can only be guaranteed in the afternoon. Then the mobility sequence will fail since no mobility service is available. To recover the failure, the process might try to search for a new appointment. However, such a redundant behaviour could be avoided by simply requesting the
mobility service search first or at least in parallel with the search for the appointment.

![Restructured workflow diagram]

**Figure 18 Restructured workflow**

A workflow shown in Figure 18 resolved the mentioned issues. Namely, treatment and mobility booking branches are placed in parallel. Moreover, there is additional parallelization inside each branch. Activity Assign₇ is in parallel with the treatment booking sequence, because there are no any dependencies. Instead, bookTreatmentCallBack follows selectTreatmentOptions due to partner protocol dependency, where partner is the citizen.

### 3.3.6 Parallelization Restriction

In some cases, the maximum parallelization is not needed or even undesirable. It happens when we parallelize activities belonging to different partner protocols. The classical example is related to hotel-flight booking scenario: we cannot book the hotel until the flight is definitely booked. On one hand these activities are independent and can be booked in parallel. On the other hand, if flight booking fails, the successful booking of a hotel should be found and cancelled. The redundant operations could have been avoided if we had a restriction of parallelization.
Definition 7. Parallelization restriction rule is an expression of the form \( n \prec m \), where \( n, m \in N \), which means that an operation carried by node \( n \) should be executed before an operation carried by node \( m \).

If we recall an example from Sec. 3.3.5, we may require parallelization restriction \( updateAgenda \prec bookTripOptions \), which guarantees that the treatment is booked before the mobility. Thus, in the case of treatment booking no additional recovery steps will be needed. The parallelization restriction rules are applied in a post-processing step and introduced via a link from the former to the latter activity. The link is not shown in Figure 18 due to peculiarities of BPEL visualization plug-in for Eclipse.

### 3.3.7 Converting BPEL Graph to BPEL

To this point, we constructed a BPEL graph and restructured it according to the principle of maximum parallelization along with the rules of parallelization restriction. Now, the BPEL graph represents an invariant structure of a parallelized process. Yet, it is not an executable process. We have to convert the BPEL graph to a BPEL file. The basic idea of such a conversion lies in consideration of a graph node \( n \) along with its parents \( P = \{p_1, ..., p_q\} \) and children \( C = \{c_1, ..., c_s\} \):

- \( q = 1 \) and \( s = 1 \), then \( p_1, n, c_1 \) are placed in sequence;
- \( q=1 \) and \( s>1 \), then sequence contains elements \( p_1, n, \) and a flow of elements \( \{c_{i_1}, ..., c_{i_s}\} \);
- \( q > 1 \) and \( s = 1 \), then sequence contains flow of \( \{p_{i_1}, ..., p_{i_q}\} \), followed by elements \( n \) and \( c_1 \);
- \( q > 1 \) and \( s > 1 \), then sequence contains flow of \( \{p_{i_1}, ..., p_{i_q}\} \), followed by \( n \) and flow of \( \{c_{j_1}, ..., c_{j_s}\} \).

Boundary cases, e.g., \( n \) is a root or a graph leaf, are trivial.

However, the graph structure may have more complex fragment. For example, several elements from \( P \) may have multiple children. In other words, there exists \( R = \{r_1, ..., r_t\} \) such that \( n = r_j \) for some \( j = 1, ..., t \), and for any \( p_i \in P, i = 1, ..., q \) there exists \( R_i \subseteq R \). This structure should be converted to flow of \( \{p_{i_1}, ..., p_{i_q}\} \) followed by flow of \( \{r_{j_1}, ..., r_{j_t}\} \).

A precise formalization of converting BPEL graph to BPEL is left for the future work, since we have discovered that such a conversion may be not unique. Moreover, we may not need a unique conversion due to performance reasons when the workload of some web-service may change with time. This issue require a more profound analysis.

For the moment, we refer an interested reader to the papers [12], [13], [14] that deal with the issue of converting graph structures to block structures.

### 3.4 Runtime Adaptation of Process Flows

Building upon our previous experience in providing instrumentation techniques for BPEL processes we have extended ActiveBPEL (an open-source BPEL execution engine) with four main components (see Figure 19): a Process Runtime Modifier, a Static BPEL Modifier, a SOAP Admin Interface, and a Changes Repository.
The runtime modifier makes use of Aspect-Oriented techniques to intercept a running process and modify it in one of three ways: by intervening on its BPEL activities, on its set of partnerlinks, or on its internal state. The runtime modifier takes three parameters. The first is an XPath expression that uniquely identifies the point in the process execution in which the restructuring has to be activated. The second is an XPath expression that uniquely identifies the point in the process in which restructuring needs to be achieved; it can be different than the point in which the restructuring is activated. The third is a list of restructuring actions. Supported actions consist of the addition, removal, or modification of BPEL activities, partnerlinks, and data values. When dealing with BPEL activities we must provide the BPEL snippet that needs to be added to the process, or used to modify one of the process' existing activities. When dealing with partnerlinks we must provide the new partnerlink that needs to be added to the process, or use to modify an existing one. When dealing with the process' state we must uniquely identify a BPEL variable within the process that needs to be added or modified, and the XML snippet that will consist of its new value.

We can use the static BPEL modifier when the process restructuring needs to be more extensive. It supports the same kinds of modifications to the process' activities, partner links, and internal variables, except that the modifications are performed on the process' XML definition. This means the process needs to be redeployed. However, this operation is completely transparent to the process users. First, already running process instances are not modified; the changes are only applied to new instances. Second, using the same endpoint, all new process requests will be forwarded to the newly deployed version of the process.

The Changes Repository is a database that is used to enable the correct behaviour of the Process Runtime Modifier and the Static BPEL Modifier. It contains the current model of the process that each process instance is using, as well as the XML snippets that represent the changes being requested. These snippets are extracted from the database when the AOP weaving detects that the process has reached the correct place in which the snippet needs to be considered.

The Admin interface is an extended version of the ActiveBPEL administration interface that exposes the web methods needed to request changes on the running processes.

3.5 Using the Unified Manageability Interface

Figure 20 shows the unified manageability interface. In order to manage a service instance the overall SLA@SOI platform interacts with a specific
ManageabilityAgent through the unified IManageabilityAgent interface. The cardinality of the manageability agent with respect to its managed service instances is 1 to n, meaning that a single manageability agent can take care of many specific service instances at a time. Moreover, the service instances can be implemented using different domain-specific technologies. This is achieved through the notion of domain-specific manageability agent facades. Each specific service instance has a domain-specific façade through which the platform can get information about the service, manage sensor configurations, and execute specific corrective actions on the service instance.

Figure 20 The unified manageability interface.

3.5.1 Managing event correlations

In order to setup correlators we have added three new methods to the ManageabilityAgent's IManageabilityAgent interface.

```java
void addCorrelator(CorrelatorConfiguration correlator)
```

The addCorrelator method takes a CorrelatorConfiguration object and constructs a corresponding Correlator to be added to the ManageabilityAgent. This causes the ManageabilityAgent to subscribe to the event bus to obtain the data it needs, as well as open a channel to the event bus for publishing the correlated information. The CorrelatorConfiguration object can be instantiated as a ResponseTimeConfiguration, a ReliabilityConfiguration, an AvgRateConfiguration, or a DomainSpecificConfiguration. All these configurations contain the unique topic identifiers used by the input data on the event bus, and the unique topic identifier that should be used when publishing the correlated data. On top of that, in the second and third case the methods also need the values for the periodUnit, periodValue, outputUnit, and outputValue attributes, while in the last case the method needs a domain-specific Esper rule. If the method receives a CorrelatorConfiguration that specifies a unique outgoing topic id that already exists within the ManageabilityAgent, the configuration substitutes the previously existing one.

```java
List<CorrelatorConfiguration> getCorrelatorData()
```
The getCorrelator data can be used to ask the ManageabilityAgent what correlators are currently active, together with all the information that were initially required to configure them.

```java
void deconfigureCorrelator(CorrelatorConfiguration correlator)
```

Finally, deconfigureCorrelator takes a specific correlator and de-activates within the ManageabilityAgent.

### 3.5.2 Managing Adaptations

Adaptation actions are achieved through specific effectors that are triggered through method `executeAction`. This method receives an instance of an `IEffectorAction` and produces an `IEffectorResult`. In the project’s second year the `DOEManageabilityAgentFacade`’s `executeAction` method only supported dynamic binding. In the project’s third year we have extended its implementation to support the DOE’s multi-party conversational model (see Section 4.2) and the runtime adaptation of BPEL process.

![Figure 21 Managing a process' bindings.](image)

Figure 21 illustrates how the DOE’s manageability agent façade has been extended to support the definition, at run time, of process binding rules. The `executeAction` of the DOE’s façade receives an `UpdateAction` object, which is an implementation of the `IEffectorAction` interface. It contains a `ServiceBuilder`, a data object containing the details of the process’ setup. In particular, ServiceBuilders that are intended for BPEL processes that are to be run within the DOE must contain the binding rules that need to be adopted. These binding rules can be of four kinds. Besides BindingRules and AbstractBindingRules, which were already present at the end of the project’s second year, the manageability agent façade now supports MultipleBindingRules and MultipleAbstractBindingRules. Since the DOE façade is aware of the system’s current setup, i.e., of its current ServiceBuilder, it can easily understand whether any changes to the current bindings are being requested. If there are changes, it creates a SOAP stub
towards the DOE’s administration interface and issues the changes using the exposed remote methods.

**Figure 22 Adapting a process' structure**

Figure 22 illustrates how the DOE’s manageability agent façade has been extended to support the adaptation, at run time, of a process definition’s or of a process instance’s structure, i.e. control- and data-flows. In this case the executeAction method of the DOE façade receives an UpdateStructure object, which is an implementation of the IEffectors interface. It contains an indication of whether the ServiceManager is requesting a structural adaptation to be applied to a single process instance, or if the adaptation involves an entire process definition. If the ServiceManager is requesting a modification of a single instance, the DOE façade uses a SOAP stub to call method changeInstance on the DOE’s administration interface. It passes the service instance’s unique id, an XPATH indicating at what point in the execution the structure change will need to be activated, and XPATH indicating at what point in the execution the structure change will need to be enacted, and a BPEL snippet representing the part of process that will need to be added. If the ServiceManager is requesting a modification to an entire family of business processes, the DOE façade calls method changeProcess. The method’s parameters are the same ad before, except that instead of an instance’s unique id, the method takes the name of the process that needs to be adapted. This ensures that the modification will take place on the process’ actual definition, and not on a running instance.

### 3.6 Related Work

There are several works dedicated to process adjustment in general, and to process restructuring in particular. In some cases, restructuring means partitioning a process with respect to some security properties or regulation rules [39], [46], [42], [43], [53]. However, such a fragmentation is useful in scenarios in which we need to outsource some process fragments for execution by an external partner. The main focus of the mentioned papers is to provide an adequate coordination of the distributed process, preserving all initial data- and control- flow dependencies. In such cases, there is no parallelization added. In other cases, restructuring means the substitution or extension of faulty fragments with recovery procedures [38], [41], [49]. The healing procedures may be introduced either in parallel or within the sequence of the considered process, but the original structure of the process does not undergo significant changes. Another class of restructuring proposals intend to provide some generic structure optimization, e.g., loops [44], topology [47], [51].

Our approach is different from the mentioned above. Namely, we “recycle” the fragments of the original process so that no activities are deleted or added but rather rearranged. This approach has similarities with process composition in the presence of QoS requirements [36], [37], [40], [54]. However, QoS-approaches are based on run-time performance evaluation. Instead, our adjustment algorithm operates statically (off-line) and takes into a consideration structural properties of the BPEL-file.
A lot of proposals deal with dependencies in business process and their optimization. For example, Zhou et al. [56] construct a dependency graph which is based on inputs/outputs of activities (data-flow) and precondition/effect of the structured activities. The graph is minimized so that redundant dependencies are eliminated, where redundant dependencies may arise from possible business conditions. This is similar to our algorithm step where connections are created with the nearest potential parents. The disadvantage of [56] is the absence of analysis of dependency minimization impact the original process.

Wu et al. [52] consider not only data- and control- dependencies (the latter is specified by conditional nodes) but also service and cooperation dependencies. The service dependencies are the same as our partner protocol, while the cooperation dependencies reflect synchronization links. The set of dependencies defines partial ordering over the process activities. It is minimized by eliminating any ordering while a transitive closure of the remaining orderings is left the same. Authors claim that in such a way they determine the least sequencing that preserve correct execution of the process. The approach is designed for monitoring workload minimization. However, it is not discussed if the remained dependencies can be used to restructure the process.

An approach that is the most similar to ours was reported in [50]. The goal of the proposal is to maximizing the throughput of the process. However, the goal is not achieved via parallelization of the composite process but rather by partitioning the process into fragments that can be outsourced and run with minimal communications among decentralized parties. The similarity to our approach lies in data- and control- dependency analysis where authors propose to rearrange original sequences of activities on the basis of their data inputs/outputs and the presence of links. However, it is not explained what happens with complex activities and how partner protocols are respected.

Another process partitioning solution is presented in [55]. The authors propose to start from a graph where all activities are placed in parallel. Data-flow analysis helps defining sequential dependencies among graph nodes. And additional optimization eliminates redundant connections among nodes. In the constructed graph, the clusters of message exchange are identified and corresponding partitioning is conducted.

Several approaches consider from-sequence-to-parallel restructuring from the view point of data flow which is modelled and analyzed as Petri-nets [35], [45], [48]. The drawback of these proposals is that control-flow requirements among process stakeholders are not considered and therefore cannot be exploited in the area of BPEL restructuring.

Regarding possible technical implementations of process restructuring, Leitner et al. [57] propose a technique for preventing SLA violations in Service Compositions using Aspect-Oriented Fragment Substitution. The approach is similar to ours yet the authors do not explicitly support class-based process evolution. Instead they concentrate entirely on instance-based modification. Their prototype implementation is entirely achieved in .NET technology, meaning it could not be easily used or extended in the context of SLA@SOI. By adopting WF Sequential Workflows, the authors were able to directly exploit Microsoft’s WorkflowChanges API. This API allows the authors to suspend, modify, and resume any running instance.

Aspects have also been used for runtime adaptation in the context of the BPEL ‘n’ Aspects framework [58]. With respect to our work they present a much more
limited approach to process modification in the sense that they only support single Web service invocations as aspects.

On the level of atomic services Kongdenfha et al. [59] have also presented related work. They adopt AOP techniques to change the implementation of atomic services. Song et al. [60] also present an approach in which they use AOP to introduce cross-cutting concerns into their Web service compositions at runtime. In this case they concentrate on quality attributes such as security. Narendra et al. [61] also follow a similar. They use an AOP-based solution to propagate changes in the non-functional properties through the composition.

3.7 Conclusions and Outlook

We have presented the advances made during the project’s third year regarding the runtime management of software services, with a special focus on BPEL processes. Regarding behavioural monitoring we have extended our ManageabilityAgents with event general and domain-specific event correlation capabilities. Regarding process adaptation we have studied and implemented an algorithm for structure re-writing. We have also provided an extension to the ActiveBPEL BPEL execution engine that will allow changes to the process’ structure to be made on-the-fly, both at the process definition level and at the process instance level. Finally, we have extended our DOE manageability façade to be able to deal with all the above novelties, as well as with the new multi-party binding feature provided by the DOE.
4 Dynamic Service Discovery & Composition

In Year 3 we have extended the mechanism supported by the DOE (Dynamic Orchestration Engine) for automatic dynamic binding (i.e. the mechanism for execution of abstract invoke activities). In the previous implementation, it was possible to automatically bind the partner role of an abstract invoke with one (and only one) web service, according to the WS-BPEL specification which refers exclusively to a bi-party conversational model [15]. We have overcame such limitation by implementing a multi-party conversational model which allows to invoke, in one step, several partners exposing the same operation. Such improvement has represented, for A3, both an innovation challenge and a specific requirements coming from B line (see B6 use case). Moreover in Y3 the automatic binding mechanism has been extended in order to take also the monitored data into account. Now the matching SLAs can be ordered on the base of the monitored QoS and the binding algorithm may chose the best one with respect to specified criteria.

4.1 Architecture & packaging

The architecture of the DOE, as realized in Y2, is illustrated in Figure 23. ENG-DOE modules are components pluggable to WS-BPEL compliant orchestration engines in order to enable: the dynamic configuration of process bindings and the execution of abstract processes.

- The dynamic configuration of bindings is enabled by the management interface IBinder (see Figure 23), offered by Binder module, which allows to associate several binding information to a specific partner role of a WS-BPEL process, in any moment after its deployment (i.e. dynamically). This interface allows associating to a partner role a fixed binding or a conditional binding, dependent from the status of the process. More precisely, it is possible to associate to each partner role a set of couples <condition, binding information> each couple representing a binding rule. The binding information identifies the service provider to invoke, while the condition represents the process status that will trigger the adoption of the associated binding. The process status on turn is represented by a set of couples <process variable, value>. The same interface is used also for the description of an abstract process (i.e. a process with some invoke activity not yet bound to a service provider), by extending the concept of binding rule with that one of abstract binding rule. An abstract binding rule associates a SLA Template to a specific condition in place of a binding information. This template provides a description of a service to be searched, negotiated and bound when the invoke node must be executed.

- The execution of an abstract process is enabled by the invocation interface IBinderInvocation (see Figure 23), offered by the Binder module, to which the engine forward service invocations. Such module extracts the abstract binding rule associated to invoke under execution and delegates to an external component, called BindingListener, the responsibility to select an appropriate web service to be invoked.
From an implementation point of view the needed modules have been grouped in three distinct packages: `doe.binding`, `doe.binding.listener`, `doe.binding.event`. The first one offers to the engine the rules management functionalities (see IBinder interface) and the invoke capability (see IBinderInvocation interface). The second one offers to the Binder the appropriate selection strategy, while the third one defines the type of event adopted for the communication.

In SLASOI the adopted engine is ActiveBPEL 5.0 (open source realise) extended to plug the DOE modules. Further details about architecture and implementation are available in [16].

4.2 DOE: multi-party conversational model

In the sections below we describe the main concepts about multi-party conversational model and limitations of current implementations (sec. 4.2.1) and the needed extensions to the architecture described above, to implement our solution to the problem (sec 4.2.2 and 4.2.3).

4.2.1 Concepts

According to the Ws-BPEL standard, an invoke activity may be linked to a single partner; in other words the partner role EPR may be wired to a single target endpoint service. Anyway, it is of interest, in some cases, that a partner role has a multi-valued reference i.e. a reference to a set of target services, all offering the same interface, to be invoked simultaneously. This is very useful when a process needs to perform the same interaction over a set of similar partners. An example of this capability is a purchasing activity wired to a list of accepted vendors.

Such issue has been taken into account by the standard SCA (Service Component Architecture). To this end SCA introduces an extension [17] of Ws-BPEL language that allows a process variable to include an `sca:multiReference` element meaning that the variable represents a multi-valued reference and specifies other than the partner link type and partner role of the reference also its multiplicity (see Figure 24, Figure 25 ).
The `sca:serviceReferenceList` element holds the set of references (Figure 26).

A typical usage of a variable that holds a multi-valued reference is to have a `<forEach>` activity with an iteration for each element in the list. The body of the `<forEach>` activity would declare a local partner link and assign one of the list elements to the local partner link (see Figure 27).

We think the SCA approach has some disadvantages:

- Such a multi-reference variable is a bit uncomfortable to manage in Ws-BPEL, since you have to explicitly cycle and make a copy of each reference in the partner link before of the invocation.
Moreover the invocations results would be available to the process one by one, forcing to add other BPEL coding to gather them.

Last, this approach implies an extension of Ws-BPEL language.

We propose an alternative approach by adopting the same architecture showed in the previous section without affecting Ws-BPEL but simply by extending the concept of binding rule. Instead of including a foreach step in the process description, we can delegate to the Binder component (see Figure 23) the invocation of several web services (all exposing the same interface) associated to just one invoke activity. To this end we allow the binding rule, associated to the partner role of the invoke activity, to contain a list of bindings. This can be formalized as a multiple binding rule:

\[
\text{<condition, bindings, variable>}
\]

where bindings is now a set of endpoints, and variable is an optional parameter that specifies the name of a Ws-BPEL process variable in which the Binder will copy the bindings so that they will be available inside the process similarly to SCA approach.

The possibility of a multiple binding applies also to the abstract binding rules that are now generalized with the following form:

\[
\text{<condition, SLAT, mergingMethod, rankingCriteria, variable>}
\]

At run time the Binder will extract from the multiple abstract binding rule, the SLAT, the name of the variable and rankingCriteria, before delegating to the BindingListener the responsibility of selecting the matching web service(s); afterwards, the BindingListener will specify a multiple binding rule containing the list of bindings (i.e. endpoints), ordered on the base of the rankingCriteria, of all services which SLA matches the specified SLAT. Finally the Binder will continue the process execution by invoking all the selected web services and merging (following the specified mergingMethod) the single results of each invocation before returning control to the engine.

### 4.2.2 Extension of DOE management interface

We have proceeded to a refactoring of DOE management interface adding missing methods to support multi-reference and renaming existing ones to better clarify, to the final user, which are the automatic binding types he can use by means of this interface.

Supported Automatic Binding Categories:

- **SLAT based single (simple/extended)**
  - Simple: it means the user wants configure an abstract invoke activity to be bound, at run time, with a single web service.
  - Extended: same as the previous one but now the user would be able to specify also a process variable where the engine will hold endpoints of alternative partners. This opportunity could be useful for a manual change of the binding defined at design time.

- **SLAT based multiple (simple/extended)**
  - Simple: it means the user wants configure an abstract invoke activity to be bound, at run time, with several services.
Extended: same as the previous one but now the user would be able to specify also a process variable where the engine will hold endpoints of specified web services. This opportunity could be useful for a manual change of the binding defined at design time (as in SCA approach).

**BINDING RULES MANAGEMENT METHODS:**

Registration of a new binding rule for a single binding is affected through the administration method:

- `addRuleForSingleBinding(String processID, String partnerRole, IEnBindingRule bindingRule);`

With such a method the user specifies, in any moment, the single web service endpoint to use (`bindingInfo` in the figure), during process execution, for all invoke activities having the specified partner role. Naturally the adoption of such endpoint is always dependent from the fulfilment of condition expressed in the binding rule.

**Figure 28 addRuleForSingleBinding: details of input data**

The registration of a binding rule for an extended single binding is effected through the administration method:

- `addRuleForExtendedSingleBinding(String processID, String partnerRole, IEnExtendedBindingRule bindingRule);`

With such method the user specifies, as in the previous case, a single web service endpoint (`bindingInfo`) and in addition a `processVariable` in which the engine has to maintain the list of alternative endpoints that can be used for a manual change of the partner role endpoint.

**Figure 29 addRuleForExtendedSingleBinding: details of input data**

The registration of a binding rule for a multiple binding is affected through the administration method:
• `addRuleForMultipleBinding(String processID, String partnerRole, IEnMultipleBindingRule bindingRule);`

This method allows the user to specify several web service endpoints (bindings) to use during the process execution, for all invoke activities with the specified partner role. It’s important to note that the engine will take care to automatically “merge” results of each single call producing a result that must have a type consistent with the one expected by the interface associated to the partnerRole. Several merge methods are supported, able to return different types depending from the return type T of the operations invoked on the single web services.

![Diagram showing the relationship between `IEnMultipleBindingRule` and `EnBindingInfo` with merging methods](image)

**Figure 30 addRuleForMultipleBinding: details of input data**

Currently the following merge methods are supported:

<table>
<thead>
<tr>
<th>SingleInvoke Return Type</th>
<th>Multiple Invoke Return Type</th>
<th>Merge Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>Map&lt;String, T&gt;</td>
<td><code>elementToMap</code></td>
</tr>
<tr>
<td>T[]</td>
<td>Map&lt;String, T[]&gt;</td>
<td><code>arrayToMap</code></td>
</tr>
<tr>
<td>SortedSet&lt;T&gt;</td>
<td>SortedSet&lt;T, String&gt;</td>
<td><code>ordElementsToOrdCouples</code></td>
</tr>
</tbody>
</table>

**Table 1 DOE: supported merge methods**

The first merge method states that if the operation, invoked on each selected web service, returns an element of type T then the engine will build a return type for the multiple invoke Map<String, T> where the map key is the endpoint of the invoked web service. The second method specifies that if the operation of the provided web service returns an array of elements of type T then the engine will build a return type Map<String, T[]> where the map key is the endpoint of the invoked web service. Finally the last method will build a return type which is an ordered set of couples <T, String> being the second element the endpoint of the first web service returning that value and assuming that such web service, returns an ordered set of elements of type T.

The registration of a binding rule for an extended multiple binding is effected through the administration method:

• `addRuleForExtendedMultipleBinding(String processID, String partnerRole, IEnExtendedMultipleBindingRule bindingRule);`

With such method the user specifies, as in the previous case, several web service endpoints (bindings) and, in addition, the process variable (bindingsVariable) where the engine will store such endpoints for a manual change of partner roles endpoints.
Similarly to the previous methods there are methods for associating abstract binding rules to a process.

**ABSTRACT BINDING RULES MANAGEMENT METHODS:**

An abstract binding rule is registered through the administration method:

- `addAbstractRuleForSingleBinding (String processID, String partnerRole, IEnAbstractRule abstractRule);`

With such a method the user states that, at run time, the SLAT, specified in the abstract rule, must be used by the BindingListener component to find a negotiated web service which SLA matches such a SLAT. The BindingListener will specify the concrete endpoint of the web service to use using the same administration interface (see `addRuleForSingleBinding`)

In this case the user specifies, as in the previous case, the SLAT that must be used, by the BindingListener component, to find a negotiated web service which SLA matches the specified SLAT; furthermore he specifies also the name of a
process variable in which to copy a list of alternative bindings. The component can, in turn, specify the endpoint of the web service to use and the process variable name always by means of administration interface (see addRuleForExtendedSingleBinding).

![Diagram](image)

**Figure 33 addAbstractRuleForExtendedSingleBinding: details of input data**

The registration of an abstract binding rule for the automatic multiple binding, is effected through the administration method:

- `addAbstractRuleForMultipleBinding(String processID, String partnerRole, IEnMultipleAbstractRule abstractRule);`

Such method states that the SLAT inside the rule must be communicated to the BindingListener that will find an ordered list of negotiated web services which SLAs match the specified SLAT. This component will, in turn, specify, always by means of the administration interface, the list of found endpoints.

Let’s note that the component exploits the `mergeMethod` information, contained in the rule (see Figure 34), to correctly set the found bindings (see addRuleForMultipleBinding for details).

Last but not least the `rankingFunction` (see Figure 34) specifies a QoS function applied to the services corresponding to the matching SLAs. The value returned by such function is used by the BindingListener to order the matching SLAs and the associated web service endpoints. Currently the attribute `rankingFunction` accepts two possible expressions of the form: `kpiName` or `-kpiName`; the first form states that the matching SLAs must be ordered on the base of the monitored value of the specified KPI, considering the service having the highest value of the KPI as the best one. The second form, of course, specifies the inverse order.
The registration of an abstract binding rule for the automatic and extended multiple binding is affected through the following method:

- `addAbstractRuleForExtendedMultipleBinding(String processID, String partnerRole, IEnExtendedMultipleAbstractRule abstractRule);`

The semantic of the method is basically the same of the previous one except that now, the user may specify also a process variable name in which to copy the found bindings. As usual, the BindingListener in turn, specifies the endpoints list and the process variable name again by means of the administration interface (see `addRuleForExtendedMultipleBinding`).

Of course the administration interface provides also methods to read and delete the set binding rules. It’s important to note that the illustrated methods are the ones of the `IBinder` interface (see Figure 23), but that analogous methods can be added on the ActiveBPEL administration web service since this engine already offers an interface for management actions (e.g. process deployment, process termination).

Finally Figure 36 shows an updated imagine of associations between partner roles and binding rules highlighting, in yellow, new interfaces involved in the methods signatures.
4.2.3 Run time environment

As described in Y2, at runtime the DOE binding rules associated to an abstract process are applied by the Binder component. The Active BPEL engine has been customized in order to delegate to the Binder the actual invocation of web services. The Binder invokes the right service chosen on the base of the specified rules and then returns the control to the engine. The interception is realized by means of the standard mechanism of ActiveBPEL (see [15] for details) by implementing a custom invoke handler (EngInvokeHandler).

In Y3 this runtime mechanism has been extended as shown in Figure 37, where yellow activities highlight the modified updated aspects.

Figure 36 Class diagram: Binder internal relationships
In Y3 we needed to modify the run time environment in order to:

- support the execution of invocations with multiple bindings
- allow the notification of BME to SLA@SOI platform
- allow the SLA@SOI platform to perform the dynamic binding of abstract processes on the base of monitored data

To satisfy the first requirement we extended the Binder by implementing the needed logic to support the merge methods.

To fulfil the second requirement, instead, a custom listener, named BindingListenerBridge, has been implemented. It publishes the BME, notified by the Binder, to the SLA@SOI monitoring event bus.

Finally, in order to satisfy the third requirement, we have implemented a SLA@SOI specific AbstractBindingListener, which is a subscriber of the monitoring event bus and can access to the monitoring data produced by the SLA@SOI RCGs (Reasoner Component Gateway) and to the content of a SLA registry. This listener implements a default binding algorithm that, at the reception of a BME, queries the associated SLA registry to obtain all the SLAs matching the SLAT specified by an abstract binding rule and then selects the SLA of the service with the best monitored performances (i.e. the service with highest value of the rankingFunction specified by the binding rule). The AbstractBindingListener uses as default SLA Registry an extended version of the Y2 SLAT Registry released by WP A5, that can store both SLAs and SLATs and can perform SLAT based queries. A domain specific PAC can be realized by extending the AbstractBindingListener, allowing to customize, if needed, the default binding algorithm. Figure 38 shows the interaction between the AbstractBindingListener, the SLA@SOI Framework Components and the DOE.
4.3 **DOE: installation notes**

The implemented modules have been released by means of the following jars:

- *doe-binder.jar* which offers the functionalities exposed by the *Binder* (i.e. invoke activities execution and binding rule administration).
- *doe-default-listener.jar* which offers a default BindibListener and AbstractBindingListener.
- *doe-event.jar* containing the BindingMissingEvent data structure definition.
- *doe-invoker.jar* to allow interaction with ActiveBPEL engine.
- *doe-slaatsoi-listener.jar* offering a BindingListener and the AbstractBindingListener needed to allow interactions with SLA@SOI framework as described in previous section.

Documentation and distribution will be available on the project SVN repository.

4.4 **Related Work**

Related work we have considered falls with following areas: portability and efficiency of services composition, support to definition and execution of SLA based abstract processes, services composition multi-party conversational model. About first area the work has taken into account mainly a survey of several WS-BPEL engines [65]. Concerning the second area our work is highly related to a “template based” automatic composition approach adopted in several projects. In SeCSE project [62], for instance, is proposed a complete solution to the dynamic service discovery and composition; anyway this approach implies an extension of WS-BPEL language and it does not allow the adoption of a SLA template as description of queried service of an abstract task. A similar approach is proposed by SCENE [63] project where a completely new composition description language is proposed with a low separation of concern.

Finally, concerning the third area, we have analyzed the Service Component Architecture (SCA) approach [64] which presents the limits already described in sec.4.2.
4.5 Conclusions & Outlook

We have presented the progresses made during the project third year in order to support the multi-party conversational model overcoming the limitations of current implementations. In detail we have described the needed extensions to the DOE Y2 architecture to implement our solution to the problem.
5 Service Monitoring

5.1 Overview

In the previous SLA@SOI A3 period (as reported in [18]) we reported on providing a mechanism to support dynamic configuration of monitoring infrastructures by way of selecting service providers from their matching capabilities of providing service monitors for consumer configuration preferences. Mechanical support is provided for this monitoring configuration for the wider service level agreement driven architecture. In this period, we have specifically contributed to the SLA@SOI monitoring architecture solution with the following innovations and achievements:

- **Enhanced Support for SLA Model Reasoning.** Additional support for Agreement Term and Guaranteed State Pre-condition reasoning and configuration, as described in the SLA model specification are now implemented.
- **SLA Coverage Testing.** An extensive SLA term and expression coverage activity was undertaken to ensure that the semantics of the SLA model where appropriately implemented in the monitoring manager module.
- **Planning and Optimization Validation.** Further integration testing with the Planning and Optimization (POC) component as part of ORC scenarios.
- **Monitoring Engine Validation (as with the POC).**
- **Monitorability Instrumentation for the SLA@SOI Eclipse Studio.**

In this period we have also published this work in [21, 22, 23].

5.2 Monitoring Architecture

As a reminder of the architecture for Service Monitoring in the SLA@SOI project, it is illustrated in Figure 39. In this section we focus on the GenericSLAManager (providing generic support for monitoring planning, optimisation, adjustments and configuration) and monitoring component features (as reasoners, sensors and effectors). The Planning and Optimization Component (POC) is a local executive controller for a ServiceManager. It is responsible for assessing and customizing SLA offers, evaluating available service implementations and planning optimal service provisioning and monitoring strategies. The POC generates a suitable execution plan for monitoring (based upon a configuration obtained from the MonitoringManager component) and passes this to the Provisioning and Adjustment Component (PAC). The PAC collects information from the Monitoring System, analyses the incoming events and decides if a problem has occurred or it is about to occur, identifies the root cause and if possible decides and triggers the best corrective or proactive action. In case the problem cannot be solved at a local level, the PAC escalates the issue to a higher level component, namely the POC. In case of an SLA violation, adjustment can trigger re-planning, re-configuration and/or alerting to higher-level SLA monitoring. These capabilities are considered to be important in order to assure preserving service provision and resource quality. The MonitoringManager (MM) coordinates the generation of a monitoring configuration of the system. It decides, for an SLA specification instance it receives, which is the most appropriate monitoring configuration according to configurable selection criteria.
A monitoring configuration describes which components to configure and how their configurations can be used to obtain results of monitoring Guaranteed States. The Monitoring System is also a central entity for storing and processing monitoring data. It collects raw observations, processes them, computes derived metrics, evaluates the rules, stores the history and offers all this data to other components (accessible through the ServiceManager). It implements the monitoring part of a ProvisioningRequest, containing constraint based rules (time and data driven evaluations) and ServiceInstance specific Sensor related configurations. It is general by design, so capable of supporting monitoring of software services, infrastructure services and other resources. Since the POC and PAC functionality is very closely related to domain specific requirements, they are provided as extendible components. For SLA@SOI case studies, they are already extended for either software service monitoring or infrastructure service monitoring. The MM aims to be generic for all and is provided as one solution.

There are three types of Monitoring Features in the monitoring system (as defined in the Service Construction Model [26]). First, **Sensors** collect information from a service instance. Their designs and implementations are domain-specific. A sensor can be injected into the service instance (e.g., service instrumentation), or it can be outside the service instance intercepting service operation invocations. A sensor can send the collected information to a communication infrastructure (e.g. an Event Bus) or other components can request (query) information from it. There can be many types of sensors, depending on the type of information they want to collect, but all of them implement a common sensor interface. The interface provides methods for starting, stopping, and configuring a sensor.

Second, **Effectors** are components for configuring service instance behaviour. Their designs and implementations are also domain-specific. An effector can also be injected into a service instance or can interface with a service configuration. There can be many types of effectors, depending on the service instance to be controlled, but all of them implement a common effector interface. The interface provides methods for configuring a service. The third type of monitoring feature is a **Reasoner** (or also known as a Reasoning Engine) which performs a computation based upon a series of inputs provided by events or messages sent from sensors.
or effectors. An example reasoner may provide a function to compute the average completion time of service requests. In this case it accepts events from sensors detecting both request and responses to a service operation and computes an average over a period of time. Reasoners also provide access to generic runtime monitoring frameworks such as EVEREST [24].

5.3 Monitorability Assessment

Given an SLA specification and a set of component Monitoring Features, our approach to dynamic assessment of monitoring infrastructures is based on the process illustrated as an activity diagram in Figure 40. The process starts by extracting the Guarantee States from Agreement Terms of the SLA specification. These terms are, in turn, parsed into a formal Abstract Syntax Tree (AST) for the expressions of the states and provide a highly efficient and formal basis for analysing SLA requirements. The AST is then used as input to select each expression of each state (by traversal of the AST) and match each left-hand side (lhs), operator and right-hand side (rhs) of the Guaranteed State with appropriate component monitoring features.

![Monitorability Assessment Activity Diagram](image)

Figure 40 Monitorability Assessment Activity Diagram

Following selection, the delegate components form a SelectedComponents list, which in turn, is used to generate a complete Monitoring System Configuration (MSC) for an SLA. If no suitable monitoring configuration can be formed (i.e. not all monitoring requirements could be matched) then an empty configuration is returned for a particular agreement term. The approach can be used for two perspectives; first, to configure the monitoring system when a new SLA needs to
be monitored, and second to perform adjustments to an existing configuration when requirements change or violations are detected. The end result of the configuration process is an MSC representing the configuration of selected Monitoring Components which reason on provide events to monitor each Agreement Term of the input SLA. The monitor matching and selection algorithms are described in detail in the previous period [18].

5.4 Packages for Deployment

The Monitoring System Configuration approach and algorithms discussed in this report are delivered in a number of implementation packages. In particular the MonitoringManager component is available as an OSGI-enabled JAVA service and can also be hosted as a Web Service. In this section we describe each of these packages with classes and their relationships (as depicted in Figure 41). The MonitoringManager module is split into a number of packages. The core package implementation supports the MonitorConfig algorithm provided by a checkMonitorability method which accepts an SLA model and a set of Monitoring Features. In turn, the implementation package depends initially on a parser package to support parsing each AgreementTerm in an SLA model. The parser package provides an AgreementTerm class containing a parse method which accepts an AgreementTerm of the SLA and produces an expression AST. A subpackage of the parser package is the core parser itself, built from the compilation of a JAVACC grammar for the SLA agreements. Additionally, the implementation package then references the methods of a SelectionManager class contained within the Selection package. This class provides methods and an overall framework for matching and selecting the most appropriate monitoring feature components with that of the expressions parsed previously.

To enable future dynamic configuration of selection algorithm, the SelectionManager refers to an extendable ComponentSelector module offering a flexible selectAppropriateComponent method which may be redefined for preferred component selection strategies. Finally, the configuration package is used by the checkMonitorability to configure the component selections in to a required MonitoringSystemConfiguration format.

![Figure 41 MonitoringManager Module Packages](image)

5.5 SMaRT Workbench
The primary goal of the Service Monitorability Reporting Tool Suite (SMaRT), which includes a Monitorability Reporting workbench, is to provide ease of access for defining, managing and reporting feedback on results for the monitorability of SLAs for service monitoring. First, we describe architecture for the SMaRT components, their relationships and interactions, and how this is implemented in a user workbench. We then describe some validation activities to ensure the approach and techniques are valid for such assessment and reporting.

5.5.1 Architecture

The architecture (illustrated in Figure 42) is based upon the Model-View-Controller (MVC) software architecture pattern providing management of models (SLA, MonitoringFeatures and Monitoring System Configuration) and their representing views by way of some controlling components. The views layer provides a component for each of these models and representing their basic state. In addition the use of these views provides input for changes to SLA, Monitoring Features or results of producing a monitoring system configuration. Whilst the SLA and Monitoring Feature Editors provide basic input and output operations on their models, the Monitorability Report view provides a much more complex view on the state of monitorability assessment and monitoring system configuration. Use of the workbench is as follows.

When both SLA and Monitoring Features are in a ready state (i.e. they are complete and updated in their models) the service support analyst can invoke check operations through the report view which starts the monitorability assessment. The MM takes the inputs of SLA and features, and performs the monitorability assessment and configuration steps as described in section 4.

![Figure 42 The SMaRT Architecture](image)
5.5.2 Monitorability Reporting

Whilst the SLA and Monitoring Feature Editors provide basic input and output operations on their models, the Monitorability Report view provides a much more complex view on the state of monitorability assessment and monitoring system configuration. Use of the workbench is as follows. When both SLA and Monitoring Features are in a ready state (i.e. they are complete and updated in their models) the service support analyst can invoke a check operation through the report view which starts the monitorability assessment. The MM takes the inputs of SLA and features and performs the monitorability assessment and configuration steps as described in section 5.3, whilst adding reporting items as part of the activities. The results of assessment yield a monitorability assessment report (with a structure as illustrated in Figure 43).

![Monitorability Report Diagram](image)

**Figure 43 Tree Structure for Monitorability Reporting Items**

The report contains for each step of assessment a log item. Each log item has a type (a kind of step, for example matching or selection etc), a result (an indicator for filtering debug, warnings or errors), a value (the value processed depending on the type of step), an optional selection item, a location indicator for the SLA processed item and a complete monitoring configuration as an MSC. The selection item is only included when a successful match and selection has occurred. In the case that a selection is made then features matched and monitors selected are reported. This information is presented back to the user in the form of a table which can be inspected and used to report information back to service consumer and provider. This is particularly useful to assess where further service monitor providers or capabilities are required. Even though an assumption in negotiations of SLAs is that the terms agreed can be monitored, additional consumer preference information (such as cost of overall monitoring, geography etc) in provision may not be known in advance. This dynamic provision of monitors may also mean that providers "supply and discontinue" monitors, leading to support issues. Additionally, the MSC is useful for service monitor providers, and service support staff, to reference matched components for particular SLA terms that can be monitored.

5.5.3 Workbench

An accessible implementation of monitorability assessment and report is provided as a workbench view, built upon the Eclipse IDE. The main Monitorability Report View (illustrated in Figure 44) provides a single view for assessing whether a supplied SLA specification and a set of features provide sufficient support for a monitoring system configuration. The workbench platform is supported by the Eclipse Modeling Framework (EMF) for underlying models of Features and Monitoring System Configuration. EMF provides native basic editors for these models (as resource views). Additionally, the SLA can be represented in a basic
XML editor. Future work will explore how these model representations are presented more appropriately to the end-user and linked to the monitoring assessment reports (e.g. bi-directional referencing). The Monitorability Report view provides an accessible interface for service monitoring support staff to check the monitorability of SLAs and examine the assessment results. On the right side of the view in Figure 44, note that the tables listed represent a break-down of SLA Agreement Term Status, Status Indicators for supporting features and Monitoring Component assignment. The table at the bottom of the report view lists each logged entry in the assessment, with good (DEBUG only), warnings (WARN) and errors (ERROR) in assessment matching and selection. Additional, more detailed, information on the report is provided in further views.

![Figure 44 SMaRT: Monitorability Report View](image)

A second view provides detailed information on the SLA Decomposition. This includes the parsing of SLA terms; term feature matching and term feature selection steps of the monitoring configuration activities (see section 5.3). The view (illustrated in Figure 45) shows a break-down of SLA terms on the left-hand side of the figure. Selecting a particular branch of the SLA decomposition (e.g. an Agreement Term and a Guaranteed Term) lists updates the details of the decomposition for that element on the right-hand side. For example, on the parse detail, the expression that is generated from a complete Agreement Term expression is listed as a series of precondition, Guaranteed Term and variable expressions. Selecting the matching detail shows how each element of an expression is matched with a monitoring feature, and the selection detail shows how a monitoring component is selected from one of this feature set.

A third view provides detailed information on the construction of the Monitoring System Configuration. The view is illustrated in Figure 46. In the event of successful selection of monitoring features for elements of the SLA, the left-hand side of the view provides a list of components configurations generated to support
either the events required (as Sensors) or the reasoning of SLA expressions (as Reasoners).

Figure 45 SMaRT: SLA Term Decomposition Results View

Figure 46 SMaRT: Monitoring Components Specification View
Selecting a particular component provides the detail of the configuration on the right-hand side of the view. For example, the reasoner configuration selected in Figure 46 is for the configuration of reasoning the term operation `coremodel:less_than`. This operation outputs a computation result of the input value from another reasoner configuration. The result of this configuration is applied to the configuration specification listed in the bottom right-hand side of the view. The SMaRT Tool Suite is open source software, available as part the SLA@SOI project platform, examples and related set of tools at http://sourceforge.net/projects/slaat-soi/. Further tools developed as part of the workbench will be released under the same conditions at that address.

### 5.6 Testing and Validation

#### 5.6.1 SLA Coverage Testing

To thoroughly test the correctness of monitoring system assessments and configurations produced, we devised an SLA coverage test based upon each of the model elements described in the SLA@SOI SLA Model and the features available by a monitoring engine. Aligned with the work on translating and Monitoring SLA(T) Service Level Agreements Using EVEREST [19] we listed each element along with the identification in a test SLA (ID), the events required to be monitored (Events), whether the model element expression in the SLA could parsed by the MonitoringManager (Parse), whether a suitable configuration was produced (MSC), whether the configuration was accepted by a client monitoring component (Client) and whether any violation or service request and response events where successfully captured (Monitored). A sample of the results is listed in Table 4. The results are split in to two, between core model elements (those which structure the SLA expressions and define variables) and expression term elements (vocabulary for specifying particularly constraints of service provision and usage). In addition to these, we also tested SLA model metrics (such as units of time) and primitive types (such as BOOL, CONST, TIME etc), mixing them and providing permutations for exhaustive testing.

<table>
<thead>
<tr>
<th>Element</th>
<th>ID</th>
<th>Event</th>
<th>Parse</th>
<th>MSC</th>
<th>Client</th>
<th>Monitored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InterfaceDeclrs</td>
<td>ID1</td>
<td>None</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AgreementTerms</td>
<td>AT1</td>
<td>Violation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Guaranteed Actions</td>
<td>GA1</td>
<td>Violation</td>
<td>No*</td>
<td>No*</td>
<td>No*</td>
<td>No*</td>
</tr>
<tr>
<td>Guaranteed States</td>
<td>GS1</td>
<td>Violation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>VariableDeclrs</td>
<td>VD1</td>
<td>Computation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Terms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core:and</td>
<td>GS1</td>
<td>Computation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Core:equals</td>
<td>GS1</td>
<td>Computation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Core:sum</td>
<td>GS1</td>
<td>Computation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Core:series</td>
<td>GS2</td>
<td>Computation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Core:availability</td>
<td>GS1</td>
<td>Request-Response</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* The element is not currently supported
5.6.2 Validation

The other main testing that has been carried out on for validation with the use-cases featured in the SLA@SOI project. The SLA specifications for both B4 (Infrastructure Services) and B6 (Business Services) have been covered in configuration testing. For example, in the coverage testing discussed in section 5.6.1, the B4 and B6 use-cases are used for input to test the various elements of the SLA model. Figure 47 illustrates the specification, planning and capture of results of these extended validation tests.

### Table 1: Properties of Model

<table>
<thead>
<tr>
<th>Property of Model</th>
<th>Preparation</th>
<th>Implementation in EMM</th>
<th>Test Cases</th>
<th>Results Checks (Rep. no.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLA Template Model</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InterfaceCheck (IT/Tx)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response Formulation</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agreement Terms (AT)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Policy Formulation</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opened/Completed (Provided)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LocalVariables</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guaranteed-Subset</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guaranteed-Value</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guaranteed-Weight</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VariableCheck (Vx)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 47 Partial Results of Testing with Use-case Elements**

Functional integration testing is on-going as the Open Reference Use-Case (ORC) [20] testing scenarios are carried out across the work package implementations.

### 5.7 Related Work

Related work in this paper falls within two areas. First we consider the approaches for decidability in the monitorability of SLAs and second the reporting on the results of this monitorability assessment. Our work is highly related to automatic monitoring from software requirements such as reported in [27], where the authors describe flexibility in generating monitoring configurations based upon some constraints specified. In this work, the user expresses his/her requirements and assumptions for monitoring in FLEA (a Formal Language for Expressing Assumptions). This language provides a set of tailored constructs, which may be composed, for the convenient expression of a wide range of monitoring concerns including sequences, combinations and time sensitive events. We base the constraints on those already specified in the SLA, and automatically derive suitable monitoring configurations based upon some monitorability calculation. Several works have focused on deciding monitorability based upon some calculation of the total-cost of monitoring the SLA between provider to provider. For example in [28, 29], the authors describe an approach to determine the monitorability of systems of SLAs (a system of SLAs is closely related to a set of Agreement Terms in the SLA notation used in our work). Analysis of SLAs in their work assumes a set of pre-configured properties (e.g. availability, satisfaction etc) and does not dynamically seek reasoning components as our work provides. Hence our work is different as it aims to report on those terms that are supported.
and unsupported, based upon constraints specified in the SLA and also service consumer monitor selection preferences. The TrustCOM project [30] has also produced a reference implementation for SLA establishment and monitoring. This implementation, however, does not involve the dynamic setup of monitoring infrastructures or reporting. The SLA Monitoring and Evaluation architecture presented within the Gridipedia project [31] has several similarities with the approach presented in this paper, such as the need to separate SLA from service management. Their focus of work, however, is on statically binding services and monitors, whilst ours is on dynamically allocating monitors to SLA parts, based upon matching the exact terms that need to be monitored and the monitoring capabilities available in different services.

Second, on monitorability assessment and reporting, the services monitoring infrastructure design and initiation shares many related aspects with broader interacting communications equipment. For example, in [32, 33] the authors describe monitorability assessment for communications infrastructure based upon a specification for monitoring (synonymous for an SLA) and how the capabilities of the monitoring infrastructure can be improved for effective reporting. Our work aligns with this but proposes how this is achieved for service level agreements rather than industrial communication standards. Bridging business and technical monitorability reporting requirements has been discussed in the NextGrid project [34], where the authors propose a human-centric architecture for SLA composition and checking. In particular, they stipulate the importance of business-level objectives such as "utilize my resources a hundred percent" or "get the maximum productivity, while spending as little money as possible" alongside the SLA quality of service terms. In a way, these business-level objectives are where the support group discussed in our work liaise with consumers and providers to reach such objectives.

5.8 Conclusions and Outlook

In this section we have reported on an approach to advanced configuration of service systems, in particular, ones in which an SLA agreement has been established and has services requiring monitoring. The work aims to provide a generic module applicable not only to the architecture illustrated but also to other architectures (although still based upon SLAs and Monitoring Component Features). A tool suite is provided to visualise the assessment of monitoring such systems. Our work will be extended to cover further elements of the SLA specification (such as Guaranteed Actions, which are not presently considered) and also including preferential selection of monitoring components. Preferential selection of components is useful where there are multiple monitoring components offered for the same term. Preferences could be based upon monitoring cost (both in computing resource and financially) or non-functional requirements. The existing implementation is already part of the wider SLA@SOI project monitoring platform, providing integration and validation testing, and we are keen to seek other environments to test it within.
6 Conclusions

This deliverable provides details about the progress made in Y3 in the A3 work package. A number of topics have been addressed in Y3. Regarding service modelling, a major focus was on the implementation of a business continuity toolkit for enhanced SOA modelling. A significant progress was also made in Y3 in the Dynamic Service Discovery & Composition area to support the multi-party conversational model overcoming the limitations of current implementations. In the Service Monitoring area, an approach is defined to advanced configuration of service systems, in particular, ones in which an SLA agreement has been established and has services requiring monitoring. A tool suite is provided to visualise the assessment of monitoring systems. Regarding the runtime management of software services, significant progress has been made with a special focus on BPEL processes. Event general and domain-specific event correlation capabilities have been added to the ManageabilityAgents for behavioural monitoring. For process adaptation, an algorithm has been implemented for structure re-writing. Additionally, an extension has been made to the ActiveBPEL BPEL execution engine to allow changes to the process’ structure on-the-fly. Finally, DOE manageability façade has been extended to deal with these novelties, as well as with the new multi-party binding feature provided by the DOE.
7 References

17. SCA BPEL Client and Implementation V1.00: http://www.osoa.org/download/attachments/35/SCA_ClientAndImplementationModelforBPEL_V100.pdf?version=1
18. DA3a Year2 Reference
19. Da5a Year3 Reference
20. B2 Year3 Reference
26. DA1a Year1 Reference for Service Construction Model...


62. SLA@SOI project, SotA document section: 1.29
63. SLA@SOI project, SotA document section: 1.30
64. SCA BPEL Client and Implementation V1.00: http://www.osoa.org/download/attachments/35/SCA_ClientAndImplementationModelforBPEL_V100.pdf?version=1
Appendix A: Glossary

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

**Agreement Initiator**: An agreement initiator is a party to a 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.

**Agreement Offer**: 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.

**Agreement Responder**: 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.

**Agreement Template**: An agreement template is an XML document used by the agreement responder to advertise the types of offers it is willing to accept.

**Agreement Term**: Agreement terms define the content of a service level agreement.

**Business Service**: A business service is exposed/invoked via at least some non IT elements.

**Business Manager**: A specialization of service provider: person that defines the SLATs of products and joins available services in a product.

**External Service**: External services are exposed across the boundaries of an organization, i.e. across at least two administrative domains.

**Framework Administrator**: A specialization of service provider: person that configures/adapts the SLA@SOI framework for a specific application.

**Guarantee Term**: Guarantee terms define the assurance on service quality associated with the service described by the service definition terms. They refer to the service description that is the subject of the agreement and define service level objectives, qualifying conditions and business value expressing the importance of the service level objectives.

**Hybrid Service**: A hybrid service is a set or bundle of other services where all these services are exposed to the customer but have different service interface types (e.g. an IT service and a business service).

**Infrastructure Manager**: A specialization of infrastructure provider: person/system that is interested to measure and control infrastructure properties.

**Infrastructure Provider**: A specific kind of service provider that focuses on the provisioning of infrastructure services.
Infrastructure Service
An infrastructure service is a specific IT service which exposes resource/hardware-centric capabilities.

Internal Service
Internal services are exposed within the boundaries of an organization, i.e. within one administrative domain.

IT Service
An IT service is exposed/invoked by means of information technology. Specific classes of IT services may be software services, infrastructure services or media services.

Offered Service
An abstract service (more precisely: service type) which is offered by a specific Service Provider to its Service Customers.

Operation Level Agreements: A specification of the conditions under which an internal service or a component is to be used by its “customer”.

Service
A means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks.
See also service interface type, service concreteness, service exposure.

Service Concreteness
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.

Service Consumer
Person(s) who actually consume/use the provided services. Typically they belong to the service customer.

Service Customer
Someone (person or group) who orders/buys services and defines and agrees the service level targets.

Service Description Term
Service Description Terms describe the functionality that will be delivered under the service level agreement. The agreement description may include also other non-functional items referring to the service description terms.

Service Exposure
Services can be exposed either internally (within the same administrative domain) or externally.
See also internal service, external service.

Service Implementation
A service implementation is a possible concrete realization of a given service type.

Service Instance
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.

Service Interface Type
Describes the nature of an actually exposed service, i.e. about the nature of his invocation interface.
See also business service, IT service, hybrid service.

Service Level Consequence
An action that takes place in the event that a service level objective is not met.

Service Level Agreement
An agreement defines a dynamically-established and dynamically managed relationship between parties. The object of this relationship is the delivery of a service by one of the parties within the context of the agreement. The management of this delivery is achieved by agreeing on the respective roles, rights and obligations of the parties. The agreement may specify not only functional properties for identification or creation of the service, but also non-functional...
properties of the service such as performance or availability. Entities can dynamically establish and manage agreements via Web service interfaces.

**Service Level Objective**
Service Level Objective represents the quality of service aspect of the agreement. Syntactically, it is an assertion over the agreement terms of the agreement as well as such qualities as date and time.

**Service Provider**
An organization supplying services to one or more internal customers or external customers.

**SLA Manager**
A specialization of *service provider*: person/system that is responsible for managing SLATs and SLA relationships.

**Software Designer**
A specialization of *software provider*: person that designs/develops the architecture and components of a specific SLA based application.

**Software Manager**
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.

**Software Provider**
An organization producing *software components* which might be used by a *service provider* to assemble actual *services*.

**Software Service**
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.

**Software Component**
Software components are the entities produced at design-time by a *software provider*.

**Service Type**
A service type (or abstract service) specifies the external interface of a service possibly including non-functional aspects. It does not specify any means (components, resources) which are needed for the actual provisioning of that service.
### Appendix B: Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>CIM</td>
<td>Common Information Model</td>
</tr>
<tr>
<td>CML</td>
<td>Common Model Library</td>
</tr>
<tr>
<td>DMTF</td>
<td>Distributed Management Task Force</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SLO</td>
<td>Service Level Objective</td>
</tr>
<tr>
<td>SML</td>
<td>Service Modelling Language</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>WSDM</td>
<td>Web Services based Distributed Management</td>
</tr>
<tr>
<td>BCM</td>
<td>Business Continuity Management</td>
</tr>
<tr>
<td>BSI</td>
<td>British Standards Institution</td>
</tr>
<tr>
<td>BIA</td>
<td>Business Impact Analysis</td>
</tr>
<tr>
<td>RTO</td>
<td>Return Time Objective</td>
</tr>
<tr>
<td>FTA</td>
<td>Fault Tree Analysis</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Modelling Notation</td>
</tr>
<tr>
<td>IT BC</td>
<td>IT Business Continuity</td>
</tr>
<tr>
<td>M2M</td>
<td>Model to Model</td>
</tr>
<tr>
<td>BEAM</td>
<td>BEhaviour Analyses Model</td>
</tr>
<tr>
<td>BPAM</td>
<td>Business Process Annotation Model</td>
</tr>
<tr>
<td>BCAM</td>
<td>Business Continuity Analysis Model</td>
</tr>
</tbody>
</table>
SLA@SOI Branding

Additional Colours for diagrams