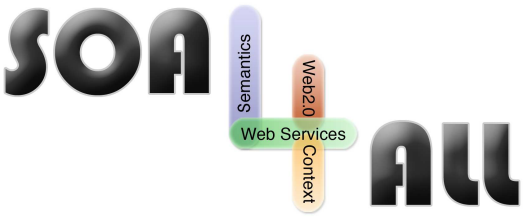


Project Number: **215219**

Project Acronym: **SOA4All**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D1.4.1A SOA4All Reference Architecture Specification

Activity N:	Activity 1 - Fundamentals and Integration Activity	
Work Package:	WP1 - SOA4All Runtime	
Due Date:		M12
Submission Date:		11/03/2009
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		UIBK
Revision:		1.0
Authors:	Reto Krummenacher UIBK Ioan Toma UIBK Christophe Hamerling EBM Jean-Pierre Lorre EBM Francoise Baude INRIA Virginie Legrand INRIA Philippe Merle INRIA Cristian Ruz INRIA Carlos Pedrinaci OU Dong Liu OU Tomas Pariente Lobo ATOS	
Reviewers:	Christopher Bussler Juergen Vogel SAP Elena Simperl UIBK	

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	2008-11-11	Initial TOC	Reto Krummenacher (UIBK)
0.2	2009-02-17	Pre-final draft prepared for reviewers	All
0.3	2009-02-25	Considering feedback by Ch. Bussler	All
0.4	2009-02-27	Integration of feedback by J. Vogel	Reto Krummenacher
0.5	2009-02-28	Integration of feedback by E. Simperl	All
0.6	2009-02-28	Creation of Section 8.1	Philippe Merle (INRIA)
0.7	2009-03-03	Integration of remaining reviewer comments	All
1.0	2009-03-06	Final release for submission	All
Final	2009-03-11	Overall format and quality revision	Malena Donato (ATOS)

Table of Contents

EXECUTIVE SUMMARY 9

1. INTRODUCTION 10

1.1 INTRODUCTORY EXPLANATION OF THE DELIVERABLE 10
1.2 PURPOSE AND AUDIENCE 10
1.2.1 Purpose 10
1.2.2 Audience 10
1.3 STRUCTURE OF THE DOCUMENT 11

2. ARCHITECTURE OVERVIEW 12

2.1 SERVICE BUS 12
2.2 SOA4ALL STUDIO AND PLATFORM SERVICES 13
2.3 BUSINESS SERVICES (WEB SERVICES) AND PROCESSES 13

3. ARCHITECTURE METHODOLOGY 15

3.1 OVERVIEW 15
3.2 COMPONENTS AND INTERACTION MATRIX 15
3.3 COMMUNICATION OBJECTS 15
3.4 FUNCTIONAL PROCESSES 15

4. SOA4ALL CORE INFRASTRUCTURE SERVICES 17

4.1 SOA4ALL DISTRIBUTED SERVICE BUS 17
4.1.1 PEtALS Enterprise Service Bus 17
4.1.2 The ProActive Grid Technology: Quick Overview 20
4.1.3 SOA4All Distributed Service Bus 20
4.1.4 Using Semantic Spaces at the SCA Application Level 24
4.2 SOA4ALL DEPLOYMENT FACILITY 26
4.2.1 Motivations 26
4.2.2 Overview 26
4.2.3 The SOA4All Artefact Repository 28
4.2.4 The SOA4All Deployment Description Language 28
4.2.5 The SOA4All Deployment Design-time GUI 29
4.2.6 The SOA4All Deployment Engine 30
4.2.7 The SOA4All Deployment Runtime GUI 30
4.3 MONITORING PLATFORM 31
4.3.1 Raw Monitoring Data Generation 32
4.3.2 Monitoring Data Communication 33
4.3.3 Monitoring Data Storage 34
4.3.4 Monitoring Data Processing 35
4.3.5 Monitoring and Management Interface 35

5. ARCHITECTURE COMPONENTS AND INTERACTION MATRIX 37

5.1 COMPONENTS 37
5.1.1 SOA4All Distributed Service Bus 37
5.1.2 Semantic Space 37
5.1.3 WSML Reasoning Framework 37
5.1.4 WSMO Data Grounding 38
5.1.5 Crawler 38
5.1.6 Service Registry 38
5.1.7 Discovery 39
5.1.8 Ranking and Selection 39

5.1.9 Design-Time Composer 39
5.1.10 Template Generator 40
5.1.11 Composition Optimizer 40
5.1.12 Execution Engine 41
5.1.13 SOA4All Studio 41
5.2 INTERACTION MATRIX 42

6. COMMUNICATION OBJECTS 45

6.1 WEB SERVICE 45
6.2 ONTOLOGY 46
6.3 GOAL 47
6.4 QUERY 47
6.5 PROCESS 47

7. SOA4ALL FUNCTIONAL PROCESSES 49

7.1 THE PROCESS METHODOLOGY 49
7.2 INTEGRATION METHODOLOGY 49
7.3 STRUCTURE OF THE SOA4ALL ARCHITECTURE PROCESS 50
7.4 FUNCTIONAL PROCESSES 51
7.4.1 Service Creation and Execution 51
7.4.2 Service Invocation 52
7.4.3 Service Discovery 52
7.4.4 Service Composition (Processes) 52

8. IMPLEMENTATION 54

8.1 THE SOA4ALL DSB IMPLEMENTATION ARCHITECTURE 54

9. CONCLUSION 57

REFERENCES 58

ANNEX A. AN INTEGRATION USE CASE WITH PETALS 60

ANNEX B. SCA SEMANTIC SPACE BINDING XML SCHEMA 62

ANNEX C. SOA4ALL SOFTWARE AND SERVICES TO DEPLOY 64

ANNEX D. A SOA4ALL DEPLOYMENT DESCRIPTION 65

ANNEX E. EVENT ONTOLOGY (EVO) 66

ANNEX F. ACTIVITY DESCRIPTION FORM 67

ANNEX G. COMPONENT DESCRIPTION 68

ANNEX H. INTERFACE DESCRIPTION 69

List of Figures

Figure 1: SOA4All Overall Architecture....12
Figure 2:Internal and External Communication Flow....14
Figure 3: The Distributed PEtALS ESB....17
Figure 4: Federation of DSBs Relying on GCM/ProActive-based Messaging....22
Figure 5: Federation of DSBs Connected with a Semantic Space Infrastructure....23
Figure 6: SCA Web Services Binding....24
Figure 7: SCA Semantic Space Component....25
Figure 8: SCA Semantic Space Binding....26
Figure 9: The SOA4All Deployment Facility....28
Figure 10: Illustration of a Graphical Notation for SOA4All Deployment Descriptions....29
Figure 11: Illustration of the SOA4All Deployment Runtime GUI....31
Figure 12: DSB Node Monitoring....34
Figure 13: Monitoring Data Aggregator....34
Figure 14: Components of Monitoring Data Processing....35
Figure 15: SOA4All M&M API....36
Figure 16: Interaction Matrix of SOA4All Platform Services....42
Figure 17 The SOA4All Process Meta Model....50
Figure 18: Service Construction Framework at a Glimpse....53
Figure 19: The SOA4All DSB Implementation Architecture....55
Figure 20: The Travel Agency Scenario as an SCA Composite.....60
Figure 21: Mapping the Travel Agency Scenario to JBI Components.....60
Figure 22: An Example of SOA4All Deployment Description....65
Figure 23: State Model Followed by EVO, see [32].....66

List of Tables

Table 1: Interaction Details in Terms of Discovery....43
Table 2: Interaction Details in Terms of Process Creation....43
Table 3: Component Implementation Release Dates....54

Glossary of Acronyms

Acronym	Definition
ADL	Architecture Definition Language
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
BC	Binding Component
BPEL	Business Process Execution Language
BT	British Telecom
CBSE	Component-Based Software Engineering
CDK	Component Development Kit
COP	Constraint Optimization Problem
CSP	Constraint Satisfaction Problem
D	Deliverable
DSB	Distributed Service Bus
EBNF	Extended Backus-Naur Form
EIP	Enterprise Information Portal
EJB	Enterprise Java Beans
ESB	Enterprise Service Bus
FDF	Fractal Deployment Framework
FP	Framework Program
FP7	The 7th Framework Program
FTP	File Transfer Protocol
GCM	Grid Component Model
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineerings
IST	Information Society Technology
IT	Information Technology

JBI	Java Business Integration
JDBC	Java Data Base Connectivity
JEE	Java Enterprise Edition
JMS	Java Messaging Service
JMX	Java Management eXtensions
JRE	Java Runtime Environment
M	Median, Milestone
M&M	Monitoring and Management
NAT	Network Address Translation, Network Address Translator
NESSI	Networked European Software and Services Initiative
NEXOF	NESSI Open Service Framework
NEXOF-RA	NEXOF Reference Architecture
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OSGi	Open Service Gateway Initiative
OWL	Web Ontology Language
QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer
RMI	Remote Method Invocation
SAP	Systeme Anwendungen und Produkte
SAWSDL	Semantic Annotations for WSDL
SCA	Service Component Architecture
SD	Standard Deviation; Service Discovery
SE	Service Engine
SEE	Service Execution Environment
SFTP	Secure File Transfer Protocol
SOA	Service-Oriented Architecture
SOA4All	Service-Oriented Architectures for All
SOAP	Simple Object Access Protocol
STP	SOA Tools Platform
SWS	Semantic Web Service
T	Task

TC	Technical Committee
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WP	Work Package
WS	Web Service
WSDL	Web Service Description Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
XML	eXtended Markup Language
XSLT	eXtensible Stylesheet Language Transformations

Executive Summary

This is the first of two deliverables about the reference architecture of SOA4All. The second one will follow with the month M24 milestone. The primary objective of this deliverable is to present the SOA4All architecture from several view points. We provide a high level description of the SOA4All platform by showing the relationship between the SOA4All core infrastructure services (the Distributed Service Bus and Semantic Spaces that are subject to WP1), the SOA4All Platform Services (aka components from WP3, WP5 and WP6) and the SOA4All Studio of WP2, respectively the external third-party business services. Based on a general architecture design methodology, we specify in more details the different components of the SOA4All platform, and their relationships to and dependencies. Another objective of this deliverable is to clearly determine the role of, and the links between different components, and to define that objects that are exchanged and shared; mainly services, ontologies, goals, and processes. These definitions are, from a conceptual point of view, closely aligned with the recent standardization work of the OASIS SEE Technical Committee and their SEE Reference Ontology specification. The last viewpoint that we provide on the architecture is related to functional processes that are realized in SOA4All. This last point of view is partly an outlook to work that will be conducted in the second year of the project, that is to say, the definition and realization of more complex processes that involve multiple SOA4All Platform Services; for instance, the discovery of service, or the composition of processes to name two. The descriptions of the functional processes are given along the lines of the NEXOF-RA requirements specification for service architectures. This alignment with NEXOF-RA has two advantages: i) the architecture specification in terms of functional processes is well-founded in terms of the definitions and conceptualizations of other NESSI strategic projects, and ii) the SOA4All reference architecture is easily comparable to the NEXOF-RA expectations and requirements, which clearly eases the exploitation of research results in NEXOF-RA. In terms of architectural specifications, there is a short section that outlines the expectations of WP1 in the context of the upcoming implementation task. A first outlook to the implementation plan and the technologies to be used in WP1 are given.

As stated above, the core infrastructural services of SOA4All are subject to WP1, and are presented in more detail (as compared to the SOA4All Studio and the SOA4All Platform Services of work packages WP3 – WP6) in this deliverable. In other words, the SOA4All Distributed Service Bus with its deployment facility, the monitoring platform and the Semantic Space bindings constitutes a special component which is presented conceptually, as well as technologically in this deliverable. Note that the Semantic Space infrastructure is presented in Deliverable D1.3.2A and thus no technical details about spaces is given here, only the aspects that concern the binding of spaces to the Distributed Service Bus. The same applies to all other components of the project (i.e. the SOA4All Studio and Platform Services).

1. Introduction

1.1 Introductory Explanation of the Deliverable

The service-oriented approach to IT system has received increasing attention from both industry and academia in the last years. The notion of service is at the very centre of this approach, abstracting from the underlying software implementations and hardware resources. Embracing the service-oriented paradigm rises a set of new and hard challenges, especially in the context of large systems such as the Web. The Web itself is evolving in what we call a Service Web, in which billions of parties are exposing and consuming services via advanced Web technology. To realize this vision, the SOA4All project will deliver a framework and infrastructure that integrate into a coherent and domain independent service delivery platform the following technologies:

- Web principles and technology as the underlying infrastructure for the integration of services at a worldwide scale.
- Web 2.0 as a means to structure human-machine cooperation in an efficient and cost-effective manner.
- Semantic Web technology as a means to abstract from syntax to semantics as required for meaningful service discovery.
- Context management as a way to process in a machine understandable way user needs that facilitates the customization of existing services for the needs of users.

Building a scalable, domain independent service delivery platform requires generic solutions for each of the challenges introduced by the Service Web vision (e.g., service discovery, reasoning, construction). But most important, it requires a well designed, scalable infrastructure that integrates and supports the interaction between various artefacts. A first important step towards the realization of this infrastructure is the definition of its architecture. It is the goal of this deliverable to specify the first version of SOA4All architecture. More precisely it establishes and it defines the integrated technical plan for SOA4All and outlines the process necessary to satisfy the general and detailed requirements for the design and development of the SOA4All platform.

1.2 Purpose and Audience

1.2.1 Purpose

The purpose of this deliverable is to provide a first version of the SOA4All reference architecture. It captures and conveys the significant architectural decisions along with their associated architectural drivers. The deliverable provides a comprehensive architectural overview of the SOA4All platform, using various architectural views to depict different aspects of relevance. First, a common terminology and a conceptual model to foster understanding across all actors involved in the development process is provided. Second, following a pragmatic approach to architecture specification, the architecture is described from the conceptual and technological view point. Interfaces and control flows are defined for all components of the SOA4All. The goal of the architecture is to provide a high-level overview of the necessary system components and their interactions, to provide understanding of the internal processes of SOA4All, and to support developers in providing components that can interoperate with SOA4All and its components.

1.2.2 Audience

This deliverable is relevant to all technical work packages in SOA4All (WP1-WP6). The target audience includes: component providers, users, and any person inside or outside of the SOA4All project interested in learning about the internal processing of the SOA4All service delivery platform. As such this deliverable presents the technological fundament,

guidelines and technological details for the implementation of the SOA4All Runtime, the various components and the integration of the SOA4All project infrastructure as an integrated whole.

1.3 Structure of the document

This deliverable is organized as follows: Section 2 provides a short introduction to the overall architecture and the various components and artefacts of SOA4All. Section 3 describes the architecture methodology used in SOA4All. It briefly presents the set of steps that lead to the final architecture specification, including the conceptual and technical realization of the architecture. Section 4 contains the description of the Distributed Service Bus (DSB), the infrastructure service and core integration platform of SOA4All. The three main technologies and artefacts, namely the PEtALS ESB, the Semantic Spaces and the ProActive distributed programming language are investigated and integrated in order to define the DSB. Furthermore, Section 4 details the SOA4All deployment facility and monitoring platform that provide deployment, monitoring and management mechanisms able to propagate and derive detailed information about the execution of services. Following the architecture methodology described in Section 2, sections 5 - 7 provide detailed descriptions about the major steps of the methodology: Section 5 discusses each component of the architecture in terms of their inputs, outputs and interaction with the other components, summarized as an interaction matrix. Section 6 describes the communication objects exchanged between components and Section 7 presents the functionalities the architecture supports, explained by means of the NEXOF-RA system requirements, Section 8 provides an outlook towards the implementation task of Year 2, and Section 9 concludes the deliverable.

2. Architecture Overview

The overall architecture of SOA4All can be structured into four parts: SOA4All Studio, Distributed Service Bus, SOA4All Platform Services, and Business Services (3rd party Web services and light-weight processes). Each of these parts is subject to a dedicated work package of the project, while this deliverable presents the project's overall approach towards a SOA4All service delivery platform and defines how the various parts integrate and interact. Figure 1 below shows a high-level depiction of the SOA4All platform.

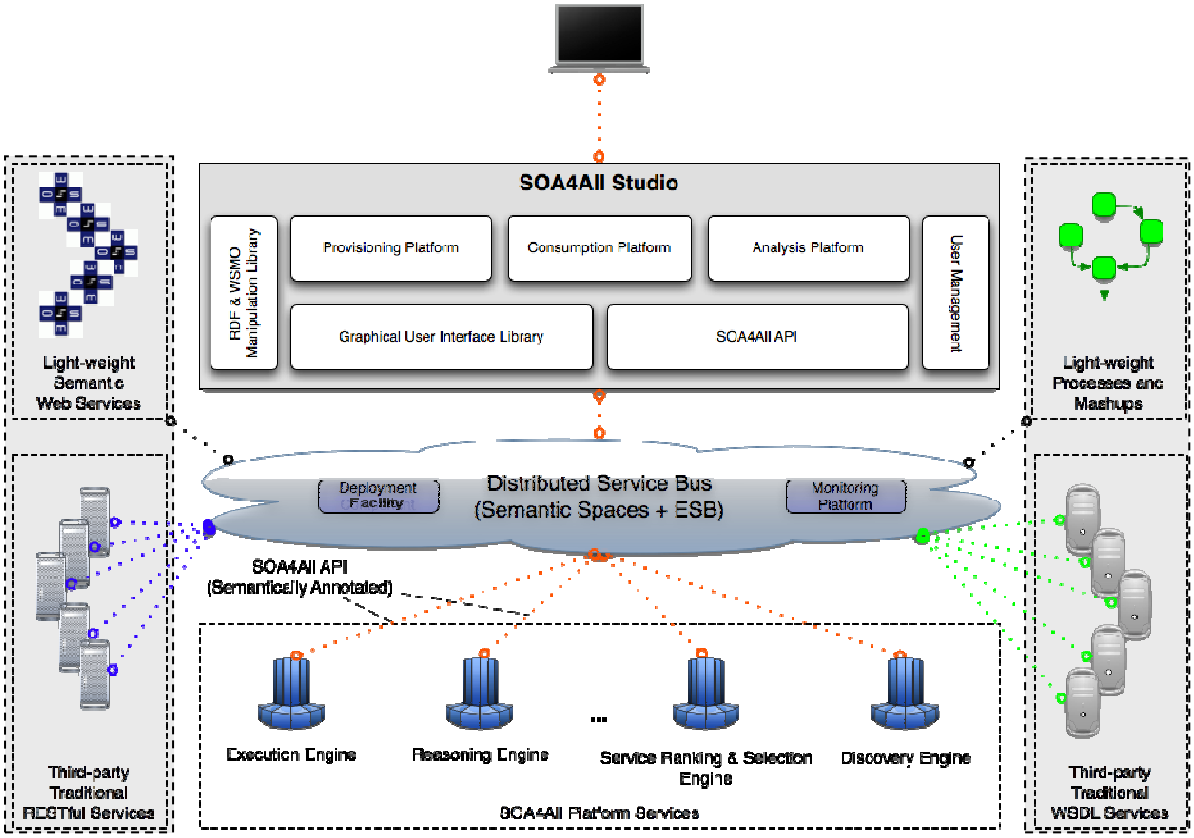


Figure 1: SOA4All Overall Architecture

2.1 Service Bus

In the very centre, there is the SOA4All Distributed Service Bus (detailed in Section 4.1). The Distributed Service Bus (DSB) serves as infrastructure service and core integration platform and is a direct evolution of the open-source PEtALS Enterprise Service Bus (ESB) that is promoted by the OW2 Consortium[1]. The DSB delivers the necessary extensions and developments of the project to augment PEtALS towards large scale, open, distributed and hence Web-scale computing environments, as the one targeted by SOA4All. These extensions and consequently novelties of service bus infrastructures include ease of deployment, openness, scalability in terms of internal repositories, amongst others (consult Section 4.1). To realize these extensions, the DSB will benefit from the internal use of the ProActive distributed programming language also promoted by OW2.[2] Problems to be solved to augment PEtALS as required are of similar nature as those raised when programming,

[1] See http://petals.ow2.org

[2] See http://proactive.inria.fr

deploying, securing, monitoring, adapting a distributed application on a computing grid infrastructure. Furthermore, the DSB is enhanced to incorporate a scalable Semantic Space infrastructure. The Semantic Space infrastructure is used as a shared memory to build repositories, as cooperative access to monitoring data, and as communication infrastructure to enhance the traditionally message-oriented bus towards a publication infrastructure for anonymous and asynchronous service communication with a notion of event-driven architecture; something currently neither available from openly-accessible ESB technology (cf. Section 4.1.4 and Deliverable D1.3.2A).

In addition, the Deployment Facility (Section 4.2) provides a uniform, declarative, user-friendly, and automatic support for the management and distributed deployment of all software composing the whole SOA4All service computing environment (i.e., DSB, Studio, and platform services). Finally, the Monitoring Platform of the SOA4All DSB (Section 4.3) collects monitoring data about the usage of SOA4All Platform Services and traditional 3rd party Web services.

2.2 SOA4All Studio and Platform Services

Around the integration platform, the SOA4All Distributed Service Bus, there are the SOA4All Studio and the SOA4All Platform Services. These are the components that are delivered by other research and development work packages in the project. The SOA4All Studio is defined in WP2 and delivers the a fully Web-based user front-end that enables the creation, provisioning, consumption and analysis of the platform services and various 3rd party business services that are published to SOA4All. The studio supports different types of users at different times of interaction. As Figure 2 depicts, SOA4All knows two major groups of users: i) a large group of service consumers that use SOA4All for the consumption of functionality that is provided by either platform services or by business services, and ii) a significantly smaller group of (administrative) users that exploits the capabilities of the platform services (via the SOA4All Studio) to annotate, select and compose Web services.

The platform services are products of WP3, 5 and 6 and deliver service discovery, ranking and selection, composition and invocation functionality, respectively. These components are exposed to the SOA4All Distributed Service Bus as Web services and hence consumable as any other published service. Their functionalities are used by the SOA4All Studio to offer clients the best possible functionality, while their combined activities (i.e., discovery, selection, composition and invocation) are coordinated via the DSB. The ensemble of DSB, SOA4All Studio and platform services delivers the innovative, fully Web-based and Web-enabled service experience of the SOA4All project: global service delivery at the level of the bus, Web-style service access via studio, and advanced state-of-the-art service processing, management and maintenance via platform services.

2.3 Business Services (Web Services) and Processes

The third part shown in Figure 1 is the semantic service descriptions and processes (composed services) that are created and processed by means of the SOA4All infrastructure. First, there are available Web services that are exposed either as traditional RESTful services, or as traditional WSDL-based services (lower parts of the figure). These are invokable third-party business services that SOA4All enhances in terms of automation, composition and invocation. Second, the top-left of Figure 1 depicts the semantic annotations of the business services, so-called Semantic Web services. The semantic descriptions are published in the Service Registry, and used for reasoning with service capabilities (functionality), interfaces and non-functional properties, as well as context data. These semantic descriptions are the main enablers of the automation processes related to Semantic Web services. Third, in the top-right corner, light-weight processes and mash-ups are shown that are the basis for the definition and execution of composed services (the SOA4All definitions of the concepts process and mash-up is given in 6.5). Both, mash-ups and semantic descriptions of service compositions are published to shared Semantic

Spaces, and become a public good for automated large-scale service computing.

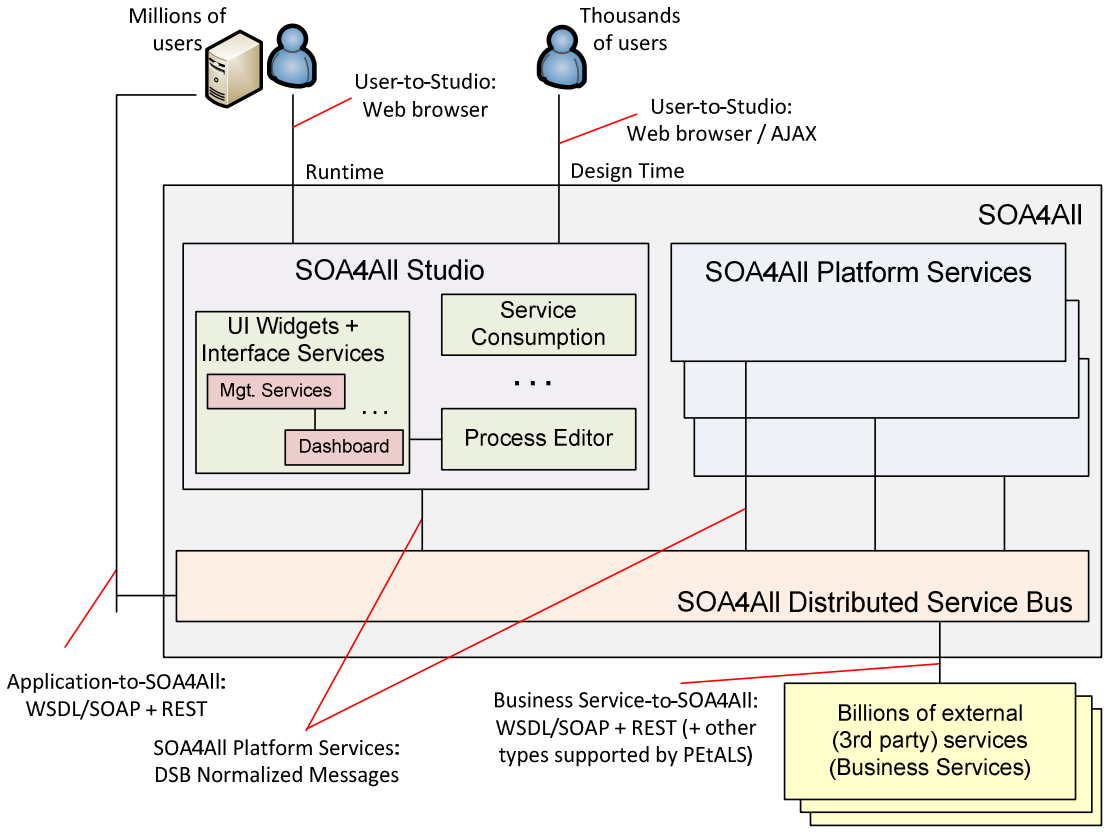


Figure 2:Internal and External Communication Flow

Before turning to the more detailed presentation of the architecture, we provide with Figure 2 a more communication-centric overview of the SOA4All infrastructure. It depicts the main building blocks: the SOA4All Studio, the Distributed Service Bus, the Platform Services and 3rd party business services. The figure shows that all communication is conducted via the Distributed Service Bus, while humans access the SOA4All platform solely via the studio, no matter if service users, or administrators that annotate services, create processes or analyse service executions.

3. Architecture Methodology

3.1 Overview

In order to define the architecture of SOA4All, we are taking a pragmatic approach by depicting the various components of SOA4All (platform services and artefacts) and putting them into context of the overall objectives and developments of the project. The goal of this deliverable is to define the integrated technical plan for SOA4All and to outline the process necessary to satisfy the requirements for the design and development of the SOA4All platform. This includes the definition of a common terminology and the conceptual model to foster understanding across all actors involved in development. Furthermore, it is the task of the architecture to align the interfaces and control flows for all components of the SOA4All platform, and to present a common framework for future roles: i) the developers of the SOA4All platform, and ii) the clients of the infrastructure.

The following sections shortly present the architecture methodology that is assumed for the SOA4All project. We shortly introduce the various steps before addressing them in more detail in subsequent chapters of the deliverable.

3.2 Components and Interaction Matrix

In a first step we present the components and an interaction matrix to showcase the invocation and communication dependencies. Primary infrastructure components of the architecture are the SOA4All Distributed Service Bus with is integrated monitoring infrastructure, and the Semantic Spaces (Deliverable D1.3.2A). Moreover, we provide the reader with a clear idea of the functional and technological aspects of the SOA4All Platform Services and show how they integrate by means of an interaction matrix. The interaction matrix helps to understand the dependencies between the various components of SOA4All, and is an important tool to describe the communication links, and consequently the communicated objects of SOA4All: service descriptions, ontologies, processes.

3.3 Communication Objects

The section of communication objects presents the various artefacts that are exchanged and shared amongst SOA4All Platform Services. While the components part discusses the interfaces of the individual components and which interfaces rely on which objects, this section lists the normative set of communication objects and provides more detailed definitions and links to the respective technical deliverables. The definitions of the objects are aligned with the conceptual model of the OASIS Semantic Execution Environment TC (SEE), and in a more terminological sense with the SOA4All Glossary (see Deliverable D1.1.1), which is a SOA4All-dedicated version of the NEXOF-RA Glossary.[3]

3.4 Functional Processes

This first release of the SOA4All architecture does not give any details about the realization and implementation of complex functional processes that will be offered by the SOA4All infrastructure. Such functional processes embed the whole life-cycles of Semantic Web services and their compositions. In a first iteration, we specify a methodology that will ease the creation and implementation of SOA4All functional processes during the second iteration of the architecture specification. This methodology will moreover allow partners to ensure that the project as a whole is appropriately equipped to deliver integrated results that fit the SOA4All overall architecture, and the needs of the project's use cases (specified and realized in WP7-WP9).

[3] http://www.nexof-ra.eu/?q=node/187

Furthermore, in the context of the functional processes, we present a first set of functionalities that the architecture should support. The functional processes are explained by means of the NEXOF-RA system requirements that were specified in their deliverable "D7.3 Conceptual Architecture View" [8]. Note that the architecture specification released with this deliverable does not yet include the documentation of complete processes that require the integration of multiple platform services, but considers those rather in isolation. Still, in Section 7.4 we provide some high-level indications and explanations of how the most relevant processes in a service-oriented architecture can be realized in SOA4All. We introduce the SOA4All view and approaches on aspects such as the creation, publication, discovery, composition and invocation of services.

4. SOA4All Core Infrastructure Services

4.1 SOA4All Distributed Service Bus

4.1.1 PEtALS Enterprise Service Bus

PEtALS is an Open Source (LGPL License) Enterprise Service Bus provided by the OW2 middleware consortium[4]. PEtALS is built with and on top of agile technologies such as:

- The Java Business Integration (JBI) v1.0 specification [11]. This is the Java standard for enterprise application integration. PEtALS has recently been certified by SUN Microsystems as a valid JBI implementation.
- The Fractal Software Component Framework provided by the OW2 consortium[5]. Fractal is a modular and extensible component model that can be used with various programming languages to design, implement, deploy and reconfigure various systems and applications, from operating systems to middleware platforms and to graphical user interfaces [5] [6]. From the PEtALS's point of view, all the container services (such as service registry, message router, message transporter, discovery, etc.) are implemented as Fractal components. This is a major feature which allows core developers to specialize a PEtALS distribution by choosing the software components to be used for specific needs.

The main PEtALS feature is the extension of the JBI specification by providing a distributed support for the JBI platform.

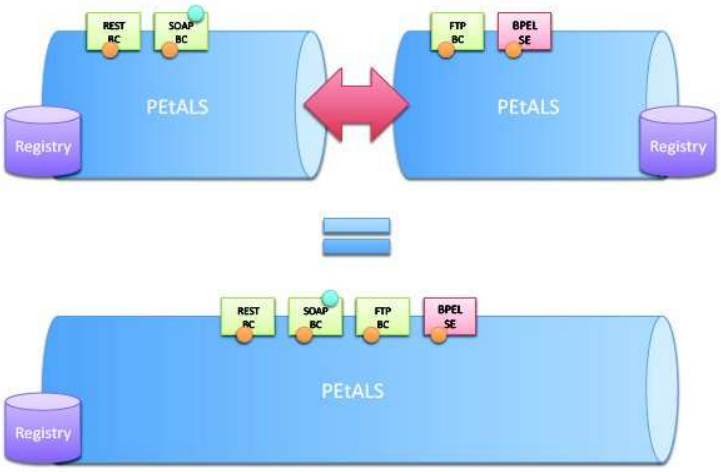


Figure 3: The Distributed PEtALS ESB

Figure 3 shows that several PEtALS containers distributed across several nodes are equivalent to a single unified PEtALS container. This transparent distribution approach ensures that all services remain accessible just as in a typical standalone JBI environment. When other JBI implementations provide a distributed approach by connecting their JBI containers with the use of JBI Binding Components plus huge configurations, PEtALS provides this feature natively without any additional configuration. This distributed behaviour is provided by the following software components:

- The technical registry. The PEtALS JBI services, endpoints, interfaces, WSDL [21] descriptions and container location (physical network address) are stored in the technical registry. This registry is used by the PEtALS container to register services and to route

[4] See http://petals.ow2.org
[5] See http://fractal.ow2.org

the JBI messages to the right endpoint. The registry entries are replicated among all the PEtALS nodes using a Distributed Hash Table over a multicast channel. This is equivalent to data flooding between registries, i.e., when an entry is added to the registry, the data is sent to all the network registries. In this way, all the registries have a complete view of the services hosted by all the containers. This approach is the base of the unified bus but the currently used multicast technique is not scalable. In SOA4All, we will tackle this current scalability issue by using the OW2 ProActive technology as described in next Sections 4.1.2 and 4.1.3.

– The message transporter. This layer is not defined in the JBI specification. Its role is to exchange JBI messages between containers. In a standard JBI implementation, the Normalized Message Router gets the local endpoint reference from the local registry and sends the message to local JBI endpoint. In the PEtALS approach, once the endpoint is retrieved from the local registry, the message and the endpoint reference are sent to the transport layer which is in charge to deliver the message to the JBI endpoint independent of the location of the container (local or remote).

PEtALS is not only a standard JBI container but in addition provides various frameworks, components and tools for extension, service integration, management and monitoring purposes:

1. The Component Development Kit (also named CDK) is a software framework which abstracts all the JBI related API and provides an easy way to develop high performance JBI components (Service Engines and Binding Components) with a small set of Java classes.
2. All JBI components are based on the previously cited CDK framework. The actual collection of PEtALS JBI components is composed of binding components (provide connectivity to/from external services) such as SOAP, FTP, JDBC, and, as planned in SOA4All, to the Semantic Space, and service engines (provide internal technical services) like BPEL, BPMN, XSLT, EIP, RULES, or SCA.
3. The WebConsole is a Web GUI used to monitor and manage the PEtALS containers (further detailed in Section 4.3). This tool provides a single access point to monitor and manage all the distributed containers. The console is connected to the containers through a JMX based data collector which is a mediator between the console and all the PEtALS containers.

Beside JBI, PEtALS also supports the Service Component Architecture (SCA) [7] [16] as standardized by the Open Composite Services Architecture (CSA) section of the OASIS consortium[6]. SCA is promoted by a group of major companies including IBM, IONA, Oracle, SAP, Sun and TIBCO working together in the Open Service Oriented Architecture (OSOA) collaboration[7].

SCA is a set of specifications for building distributed heterogeneous applications and systems using the principles of Service Oriented Architectures (SOA) and Component-Based Software Engineering (CBSE). SCA proposes a programming language-independent hierarchical component model where components offer services, make references to services supplied by others, provide configurable functional properties, and are combined together by composites which wire references to services. Wire references declaratively apply bindings for communication methods and apply policies for aspects such as security and transactions. As any hierarchical model, an SCA component can be implemented either by primitive programming language entities (e.g., a Java class) or by an SCA composite. SCA composites are declarative based on an XML-based architecture description language

[6] http://www.oasis-opencsa.org
[7] http://www.osoa.org

(ADL [13]). SCA extends and complements prior approaches to implementing services, and SCA builds on open standards such as Web services.

Four main principles underlie the design of SCA and are meant to define a distributed service computing environment which is as independent as possible from the underlying technologies:

1. **Independence from programming languages**. SCA does not assume that components are implemented with a unique programming language. Rather, several language mappings are supported and support programming SCA components in Java, C/C++, BPEL, Spring, COBOL, to name a few.
2. **Independence from interface definition languages**. SCA components provide services and require references through precisely defined interfaces. SCA does not assume that a single interface definition language (IDL) will fit all needs. Rather, several IDL are supported such as WSDL and Java interfaces.
3. **Independence from communication protocols**. Although Web Services are the preferred communication mode for SCA components, this solution may not fit all needs. In some cases, protocols with different semantics and properties may be needed. For such cases, SCA does not assume a fixed set of supported communication protocols and it provides the notion of binding: a service or a reference will be bound to a particular communication protocol such as SOAP for Web Services, Java RMI, Sun JMS or REST.
4. **Independence from non-functional properties**. Non functional properties may be associated to an SCA component with the notion of policy set (also referred to with the term of intent). The idea is to let a component declares the set of policies (non-functional services) that it depends upon. The platform is then in charge of guaranteeing that these policies are enforced. So far, security and transactions have been included in the SCA specifications. Yet, developers may need other types of non-functional properties. For that, the set of supported policy sets may be extended with user-specified values.

These four main principles make SCA an industrial SOA-centric model and standard appropriate for SOA4All reference architecture, integration, and implementation activities. SOA4All platform services designed and implemented in WP3 to WP6 and compositions/orchestrations of business third-party services addressed in WP6 can be easily encapsulated into SCA composites and components in order to benefit from the four main SCA principles previously mentioned.

The support of SCA in PEtALS [22] is done in the context of the SCOrWare project[8] [14] founded by the French national research agency. This support is composed of two parts: 1) An SCA deployment facility translating XML-based SCA descriptors into JBI deployment descriptors and especially mapping SCA bindings to JBI binding components and 2) an SCA SE hosting and executing SCA composites and components implemented in Java. Both parts are implemented by reusing the LGPL open source FraSCAti SCA platform hosted by the OW2 Consortium[9] and mainly developed by INRIA. As PEtALS, FraSCAti is implemented on top of the Fractal component model enhancing its extensibility and reconfigurability [18].

Annex A illustrates a simple integration use case showing how agile a system is built on top of the PEtALS ESB.

[8] http://www.scorware.org
[9] Available at http://frascati.ow2.org

4.1.2 The ProActive Grid Technology: Quick Overview

ProActive hosted by the OW2 consortium[10] and developed at INRIA, is a 100% pure Java solution. More precisely, it is an asynchronous RMI such as active-object based library that offers the notion of asynchronous calls with futures (a future is a promise to get back a response) among distributed objects, extended with the possibility to handle transparently group of objects [1] [2] and security (authentication, encryption, etc.) for inter-object communications. The transport layer communication protocol used by ProActive remote method invocation can be chosen at will (e.g., RMI, RMI over SSH, or pure HTTP, amongst others). ProActive also implements a component-oriented programming model: a grid extension of the Fractal model [3], also used within PEtALS, named Grid Component Model (GCM). Active objects and components can be exposed as Web services if needed.

ProActive runtimes act as containers for active objects. Those runtimes are grid-aware in the sense that they can be started remotely from any machine, using any remote access protocol, taking in charge if needed all the required file transfers (i.e., Java version, ProActive bundles, etc.) at the remote places using any file transport protocol. To cope with potential firewalls or NAT addressing within an administrative domain, that may refrain from direct point-to-point communication between any pair of ProActive runtimes, the description then enactment of the deployment process can include some ProActive runtimes launched typically on front-end machines and acting as relays for incoming and outgoing messaging at the application level. The ProActive suite also includes a resource manager gathering existing ProActive runtimes started automatically, e.g., at boot time to form an underlying peer-to-peer network, ready to be used by applications on demand (what is called a peer-to-peer computing grid). Such a peer-to-peer like computing infrastructure is the underlying support for the SOA4All architecture.

ProActive runtimes, active objects and components are all equipped with some monitoring capabilities: JMX-based probes are placed at the level of runtimes, and monitoring statistics are gathered at the level of active objects or components. All those information can be collected in a distributed, parallel way, using an extension of JMX connectors (ProActive/JMX) [4], and are typically shown within some Eclipse-based application, the ProActive-based IC2D tool.

4.1.3 SOA4All Distributed Service Bus

The aim of this section is first to explain how the DSB can benefit from ProActive features, thus offering a grid-enabled extension of PEtALS and as a result the notion of a federation of service busses. It also addresses for which purpose and how the DSBs interact with the Semantic Space.

DSB and Federation of DSBs

The SOA4Alll Distributed Service Bus is the core infrastructure of the SOA4All project. One of the WP1 main goals is to be able to address billions of services. In order to achieve this goal, the SOA4All Distributed Service Bus will be based on robust technologies such as the PEtALS Enterprise Service Bus and the combined use of ProActive. The current PEtALS ESB will be extended in order to handle services at the Web scale (potentially billions of services) where it is actually limited to the enterprise scale (hundreds to thousands of services, and several tens of PEtALS containers). In more detail, we think the following is a reasonable calibration of the expected scale:

- Billions of third-party services may exist, some of them are associated with a semantic description and some are not, some are stored in the SOA4All Semantic

[10] Available at http://proactive.ow2.org

Space and some are not; in any case, they exist and can be invoked independently from SOA4All ;

- Thousands of SOA4All users are interacting with the SOA4All Runtime through the SOA4All Studio (a tool that should offer more than one entry point into the DSB for robustness and availability).
- Those users create several thousands of new SOA4All composite services involving probably only a fraction (e.g., the most popular) of the billions of third-party services and existing composite services. However, those composite services are hosted by the SOA4All DSB, require one JBI endpoint per invoked external service, and their semantic description and related information are stored and further looked for in the Semantic Space.
- Besides, the DSB hosts all SOA4All Platform Services (e.g., the WP6 Execution Engine), that might be available in several copies for robustness, availability and performance.

The planned overall architecture of the SOA4All runtime is that of a federation of DSBs. Indeed, we expect that some operators around the Internet (e.g., SAP, BT, Amazon, Yahoo, and others) will allocate some computing resources hosting a middle-scale DSB, offering one or several access points to their respective SOA4All end-users. The challenge is to interconnect all such DSBs together so that SOA4All end-users benefit from the Internet wide SOA4All infrastructure, and in a totally seamless manner. A composite service should be able to use any existing third-party or SOA4All hosted composite service without being aware of the actual location of it (local to the same DSB or hosted onto a remote one). The same applies for the SOA4All Platform Services: sharing of load among these services of the same type must be possible, without visible impact for the clients of such services, except an improvement of the quality of service.

Our proposition is to rely on the GCM/ProActive distributed programming technology for supporting the required extensions for the PEtALS bus acting as the core SOA4All infrastructure. Key element in this idea is that PEtALS containers rely on GCM/ProActive components in place of Fractal components whenever needed, in particular for the inter-container message transport module of PEtALS. It might be necessary to define and to implement a set of standardized APIs allowing the integration of a new DSB into the federation. From an implementation viewpoint, this results in dynamically configuring the underlying GCM/ProActive messaging application so that it includes the necessary routing information to deliver messages to/from this new partner.

To be able to scale, the PEtALS ESB needs improvements on how it handles services references. According to Section 4.1.1, a main limitation is the way the technical registry replicates the service references in all the registries of the PEtALS network, based on multicast. This requires that the bus is deployed within a single administrative domain, and it is not very scalable, when several thousands of entries for services references must be replicated in all the registries at the level of a single DSB, and even worse at the level of the federation of DSBs.

The distributed registry management part of PEtALS will be redesigned and implemented as a GCM/ProActive based application: message exchanges required by registry operations will be handled as ProActive communications, thus benefiting from parallel, asynchronous, secured interactions among remote containers. Additionally, to reduce the volume of messages needed to maintain a coherent view of distributed registries over the federation, transport of messages will be organized in a hierarchical manner. This means that the transport module on which PEtALS containers of the federation will rely on a distributed and hierarchically organized application capable to handle point-to-point and collective communications be they intra or inter-DSBs. The use of GCM/ProActive for building such an application has been validated previously [12] in a rather different setting: a runtime for a

parallel programming MPI-like library, using GCM/ProActive components.

Figure 4 below summarizes this vision. It also illustrates that the end-user through the SOA4All studio, and the Generic API, will connect to the virtually flat and unique SOA4All infrastructure. However, it will connect to its 'preferred' DSB. Whenever an endpoint reference of a service hosted by PEtALS (in the figure: circles at the bottom of the JBI components) indicates that the service is remote, then, a message to the service must be triggered. It will be transported by the GCM/ProActive messaging module, using inter-DSBs interconnections if needed.

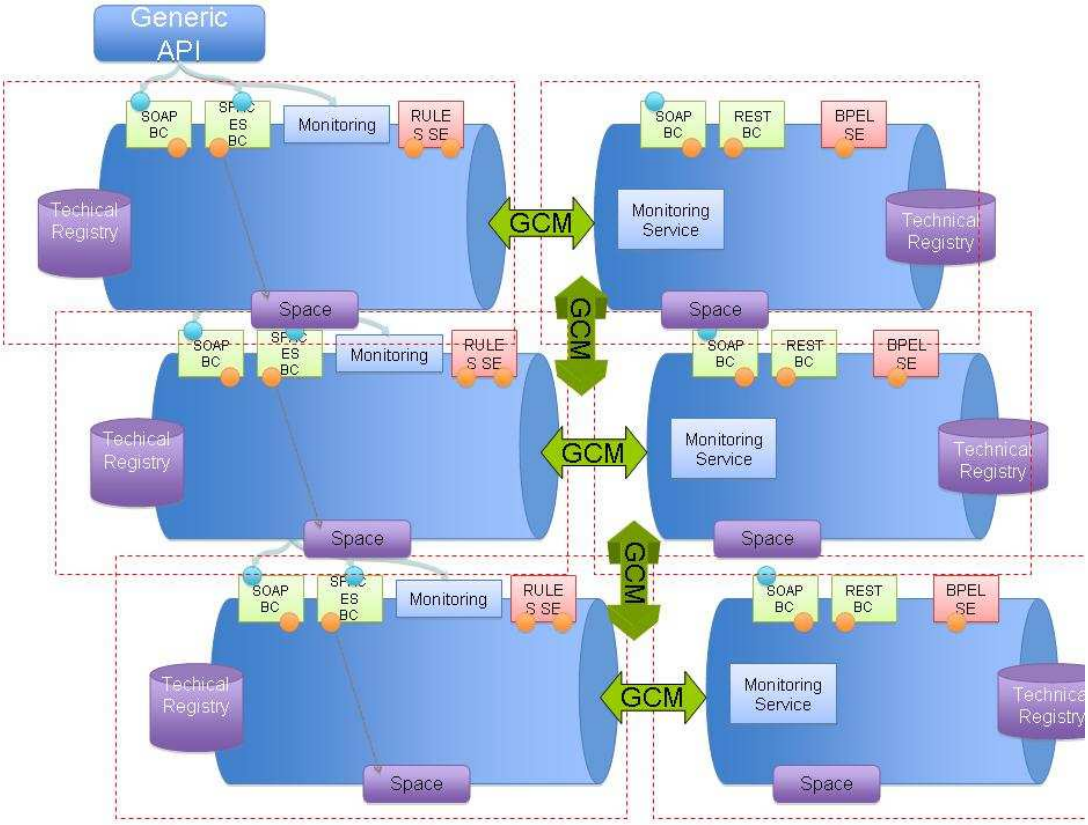


Figure 4: Federation of DSBs Relying on GCM/ProActive-based Messaging

DSB and Semantic Space Interconnection

We just briefly recall the arguments that drive our motivation to connect the federation of DSBs to the Semantic Space infrastructure (see Deliverable D0.1 and Deliverable D1.3.1A for additional details).

1. It will serve as a shared store of semantic information associated to services, (interfaces semantic description, monitoring, annotations information, etc), specifically to SOA4All composed services that are hosted by the DSBs.
2. It can serve as an event-driven publish-subscribe mechanism in order to define an alternative to the client-server service interaction mode. Section 4.1.4 further develops how a service composition along the SCA model, relying on semantic-space based publish-subscribe interaction protocol can be effectively expressed, and further translated according to the JBI specification in PEtALS.

We will develop two complementary ways to exploit Semantic Spaces in the context of PEtALS ESB:

- Implementing a specific JBI Binding Component. The Binding Component is then in

charge of translating JBI messages to/from triples and then interacting with the space. The component needs to be configured to listen to the space and to publish data into the space. This configuration is based on the JBI Service Unit capabilities. An example of configuration could be 'the JBI endpoint E publishes JBI messages to the space into a triple with pattern P' or 'the component listens to space notification with pattern P and sends JBI messages to the JBI service S'.

- Implementing a new inter-container message transporter. Thanks to the agile PEtALS architecture, a new message transporter based on the space message transport facilities can be plugged to the bus. This approach is not JBI compliant and needs evolutions on the PEtALS side but is more flexible and needs fewer configurations than the Binding Component solution one. We can imagine that all the activated endpoints are then reachable from any JBI service consumer on a 1 to N communication way. This approach will lead us to an event-driven architecture. To satisfy this, when a JBI endpoint is activated on one PEtALS container, the endpoint activation service registers specific space listeners. The listeners must take care of the JBI endpoint definition (a service name, an endpoint name, an interface name and a WSDL description). As a result, once a message is published into the space by a PEtALS container, which means that a JBI service is invoked, listeners will be triggered and the associated JBI services will be invoked.

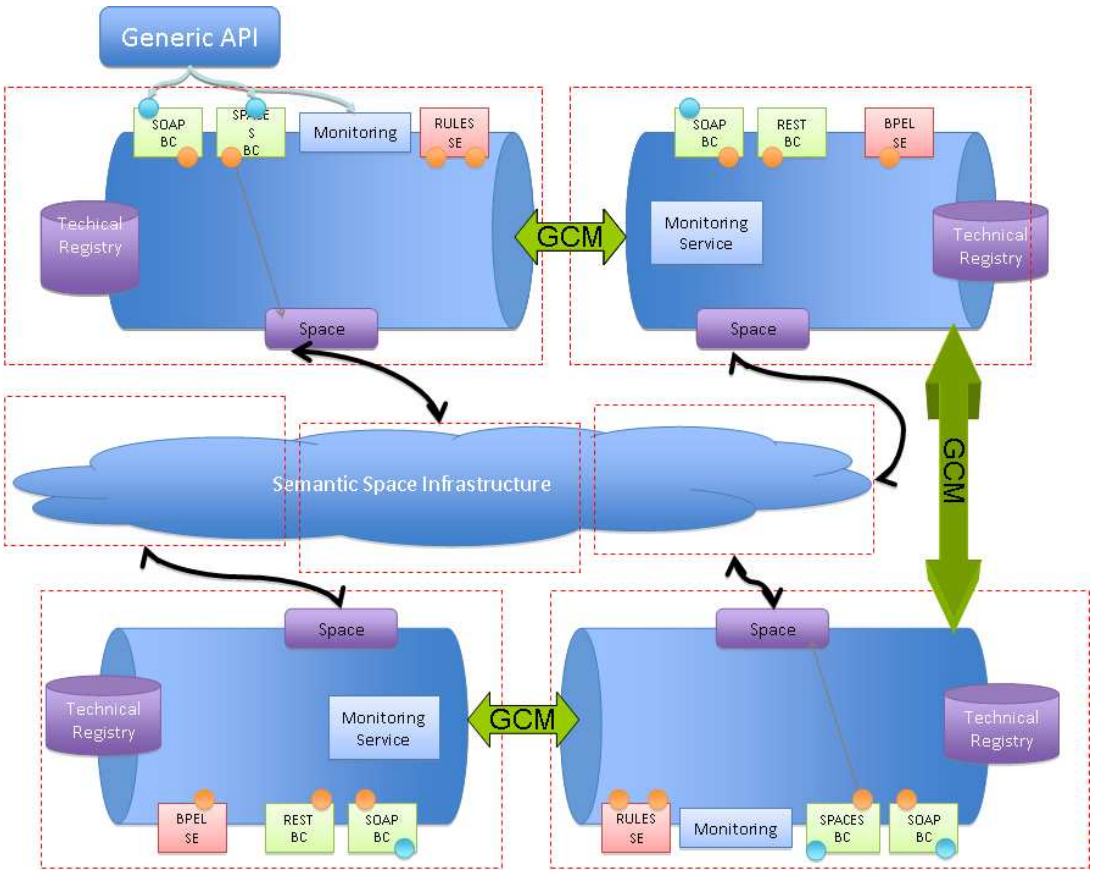


Figure 5: Federation of DSBs Connected with a Semantic Space Infrastructure

The Semantic Space itself is defined in Deliverable D1.3.1A and its foreseen implementation will be achieved by using the ProActive technology and programmed using active objects. Nodes constituting the back bone of the Semantic Space will thus be ProActive nodes (depicted by dotted lines rectangles in Figure 5 above). The back bone itself will be a possibly growing set of machines (along peer-to-peer grid computing or cloud computing principles). Note that ProActive nodes hosting the PEtALS containers and those hosting the Semantic Space nodes will not have to be co-located on the same computers. Of course,

from the DSBs federation viewpoint, the space will be a virtually unique and flat object and the space binding component will be the gateway to access it. The precise definition of the API to be offered by this new binding component will be conducted in the next months (it will enable to store and lookup information in the space, or to register to events). Its implementation will have to comply with the technology used to deliver the space. As the space will be developed using ProActive, resulting messages have the BC or the message transporter interact with the space (small black arrows in Figure 5) will be in fact ProActive messages.

4.1.4 Using Semantic Spaces at the SCA Application Level

Traditionally, distributed service-oriented applications are built on top of WSDL, SOAP and HTTP technologies. As shown in Figure 6, a service is described with WSDL and implemented by a provider, and then consumers use the same WSDL service contract to interact with the service through a SOAP/HTTP communication channel.

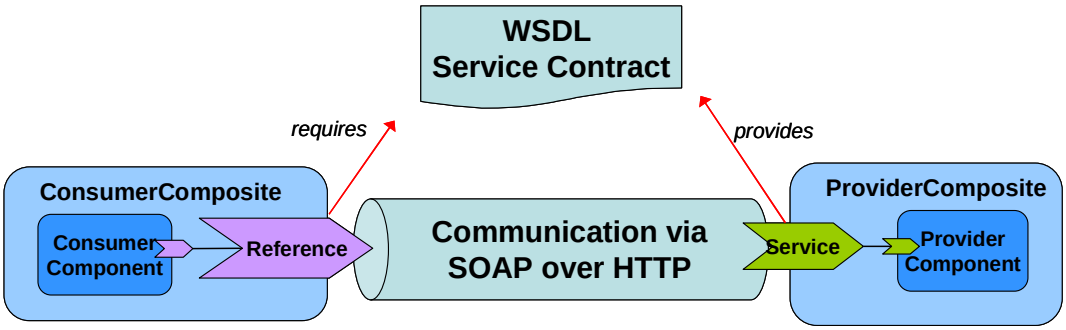


Figure 6: SCA Web Services Binding

Even if this traditional approach is widely accepted, it raises some issues:

- **Only a client/server communication style**. This approach only allows client/server communication interactions where a consumer (or a set of) invokes the service provider. But other communication styles like publish/subscribe or event-based ones can be more appropriate for certain distributed service-oriented applications.
- **Only a common service interface**. This approach imposes that the consumer uses the same WSDL service contract that the provider implements. However, in integration use cases, an already existing consumer can have a different interface and then an adapter between the consumer and provider interfaces is required.
- **Only known service provider endpoints**. Consumers need to know the exact service provider endpoint (or physical network address) in order to invoke the service. This avoids dealing with load balancing of consumer requests between different providers, or fault tolerance when the provider crashes; an issue with traditional SOAP/HTTP.
- **Synchronicity**. The provider must always be active (under execution) when consumers send service requests. However, it could be useful to allow consumers to send requests before the provider is active, and then these requests must be buffered by the communication layer until the provider is available.
- **A weakly adaptable approach**. Finally, we could expect that a large-scale distributed service computing environment as the SOA4All DSB provides more adaptability than traditional Web services technologies. For instance, providing different communication styles (client/server, publish/subscribe, event-based, semantic-based, etc.), allowing having different WSDL contracts at both sides and making the adaptation transparent, load balancing of consumer requests between various providers, supporting anonymous and asynchronous service interactions, selecting the service provider according to the context of service consumers.

To address these issues, the SOA4All DSB integrates Semantic Spaces (specified in Deliverable D1.3.1A) as a large-scale distributed support for interaction and coordination between service providers and consumers. The rest of this section presents two complementary approaches for integrating Semantic Spaces and SCA: 1) the SCA Semantic Space Component and 2) the SCA Semantic Space Binding.

The first approach, called SCA Semantic Space Component and illustrated in Figure 7, must be used when SOA4All platform services or composite services require using the complete Semantic Space API to realize their functionalities, e.g., using Semantic Spaces as a shared RDF metadata repository. This approach consists of encapsulating the implementation of SOA4All Semantic Spaces into an SCA component. This approach inherits all benefits from SCA like 1) providing a declarative architectural view of the integration thanks to the XML-based SCA ADL and existing Eclipse STP SCA graphical tooling[11], 2) configuring the behaviour of the Semantic Spaces implementation via SCA component properties, and 3) applying non-functional properties (like security, transactions, or logging) via SCA intents and policy sets. This SCA component provides the API of Semantic Spaces defined in Deliverable D1.3.1A as an SCA service. It must be included into all SCA composites requiring using Semantic Spaces, and then it can be seen as a local access point to distributed Semantic Spaces. Consumer and provider components are wired to their local SCA Semantic Space Component and use directly the whole API of Semantic Spaces to interact and coordinate.

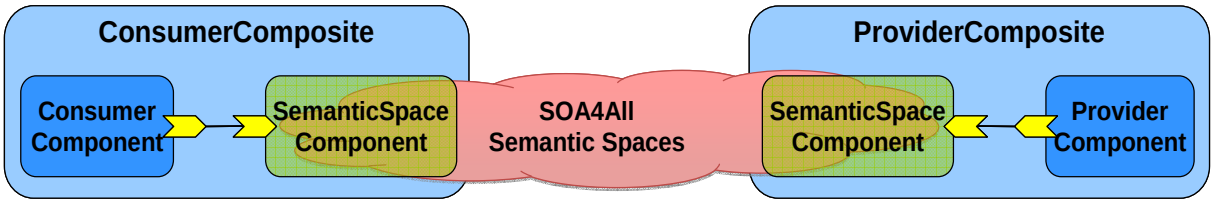


Figure 7: SCA Semantic Space Component

The second approach, called SCA Semantic Space Binding and illustrated in Figure 8, must be used when SOA4All platform services and composite services would like to interact through services invocations transported by a Semantic Space but they do not require using the Semantic Space API directly. This approach consists in providing a new SCA binding using Semantic Spaces as SCA communication channels. The main idea is to bind provider services and consumer references to a same Semantic Space in order to transparently deliver consumer requests to the service provider. From the provider's point of view, binding an SCA service to a Semantic Space means subscribing to this Semantic Space for receiving incoming requests, then handle Semantic Space notifications as service invocations, and writing back to the Semantic Space the result of the service invocation. From the consumer's point of view, binding an SCA reference encompasses the translation of service invocations into a triple written in the Semantic Space, and then subscribing to the Semantic Space to receive the result of service invocations. Then, service invocations of the SCA consumer component are transparently sent to the SCA provider component. Annex B provides the specification of the XML Schema for SCA Semantic Space Binding and illustrates its usage on the definition of SCA provider and consumer composites.

[11] see http://www.eclipse.org/stp/sca

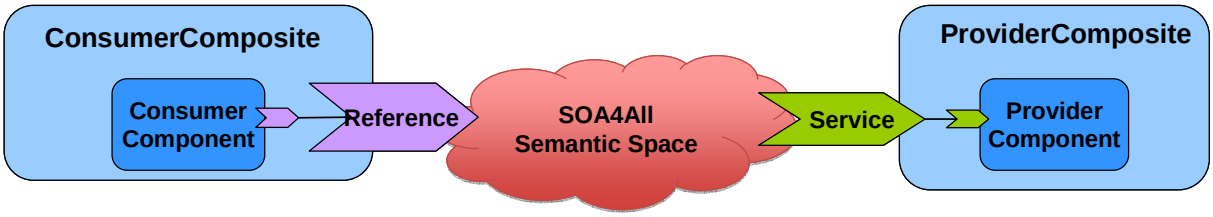


Figure 8: SCA Semantic Space Binding

The two complementary approaches presented in this section will be implemented in the OW2 FraSCAti SCA platform which is integrated into the OW2 PEtALS distributed ESB as discussed in Section 4.1.1.

4.2 SOA4All Deployment Facility

4.2.1 Motivations

At runtime, the SOA4All distributed service computing environment is composed of various software technologies: the SOA4All DSB composed of PEtALS nodes, Semantic Space nodes and the ProActive middleware running on top of Java Runtime Environments (JRE), SOA4All platform services deployed on and accessible via the SOA4All DSB, web servers running the whole SOA4All Studio and its components, semantic repositories, database servers, and so on. The SOA4All Deployment Facility provides an automated support for the deployment of all the software involved in the SOA4All distributed infrastructure (cf. Annex C for a complete list).

Deployment encompasses all the activities to upload, install, configure, and launch software and service artefacts onto physical nodes automatically. These artefacts are at least business services, components, processes, and workflows, but also encompass the whole underlying runtime infrastructure containing system libraries, language runtime, database servers, middleware services, application servers, process/workflow engines, and all their software dependencies.

Deployment of service-oriented systems as the SOA4All environment on large-scale infrastructures like the Web poses difficulties for users as they have to deal with a lot of deployment issues (often manually): 1) the heterogeneity of physical nodes (e.g., various operating systems and network protocols) where software is deployed, 2) the diversity of service-oriented technologies used to develop and deploy services (e.g., SCA [16], JBI [11], OSGi [17], JEE [19], .NET, etc.), 3) the efficient orchestration of all the deployment activities (uploading, installation, configuration, launching, etc.) required to place individual software/service artefact or complete stack onto a target system, 4) the correct management of dependencies between services and underlying infrastructure software, 5) the static verification of software deployment configurations in order to early detect errors before enacting deployment activities, and 6) the scalability to tackle large scale systems composed of thousands of physical nodes. Currently, large-scale deployments are a significant challenge for end-users as they must deal with all these issues manually, i.e., this is often error-proven and time consuming. The SOA4All Deployment Facility targets to address all these previously mentioned challenges.

4.2.2 Overview

The SOA4All Deployment Facility provides a uniform, declarative, user-friendly, and automatic support for the deployment of all software composing the SOA4All infrastructure (i.e., SOA4All DSB nodes, SOA4All platform services, and SOA4All Studio servers – cf. Annex C for more details). Based on a graphical language and its associated design-time GUI, users are aided in describing i) the set of physical nodes composing the distributed target system and their properties (their network address, supported network protocols, operating system, etc.), and ii) the software/service artefacts to deploy and their properties

(the target node, the location of artefacts to upload, the installation place, software/service configuration parameters, and dependencies to other software and services). The SOA4All Deployment Facility enables execution of these users' descriptions in order to deploy SOA4All software stacks onto the distributed target system automatically, i.e., without requiring human interventions. The SOA4All Deployment Facility supports:

- **The diversity of service-oriented technologies** – In a world-wide multi-providers context as SOA4All, several service-oriented technologies are used to develop and deploy SOA4All services. The SOA4All Deployment Facility supports commonly used technologies as BPEL, JBI, SCA, OSGi, JEE, etc. However the SOA4All Deployment Facility is extensible by plug-ins to support any other kind of technologies. A particular attention will be provided to support deployment of the SOA4All DSB, platform services, Studio, and of technologies used by SOA4All use cases (WP7, 8 and 9).
- **The orchestration of deployment acts** – Deploying software/services automatically requires executing several kinds of activities like installation of software and service artefacts onto target nodes, configuration of services and software, as well as launching them. Installation refers to making the provisioned service or software stack known or available to the operating system on target nodes. Installation can encompass sub activities like uploading service and software artefacts onto distant target nodes via remote file transfer protocols, unpacking/uncompressing software archives into the right target directory, etc. Installed software stacks and services might not be ready for use. Configuration is making installed services or software stacks ready for use. Configuration typically requires setting both system-specific and business-specific information of the installed software and services. Finally, launching is making software and services available for runtime use. The SOA4All Deployment Facility provides an extensible support for commonly used deployment activities such as installation, uploading, uncompressing, configuration, and launching services. New activies can be added as plug-ins as required.
- **The management of software and service dependencies** – One software or service often depends on others or underlying software stacks (e.g., an SCA business composite must be executed on top of an SCA-compliant runtime platform, or a service can use a database hosted by a database server). The SOA4All Deployment Facility provides the capability to describe dependencies between services and underlying software stacks, and to manage them automatically during distributed deployment.
- **The variety of software and service artefact packaging formats –** Various formats for packaging and delivering software and service artefacts exist (e.g., .zip, .tar, .tar.gz, and .rpm archives). The SOA4All Deployment Facility supports these commonly used packaging formats, and support for other formats will be added as plug-ins as required by SOA4All use cases.
- **The heterogeneity of network protocols and operating systems** – In a distributed system at the scale of the Internet, several network protocols are available to transfer service and software artefacts and to submit atomic deployment commands to remote target nodes. The SOA4All Deployment Facility supports commonly used protocols like FTP, HTTP, JMX, SCP, SFTP, SSH, and Telnet. Other protocols will be supported as plug-ins according to requirements of SOA4All use cases.

As shown in Figure 9, the SOA4All Deployment Facility is composed of the following five elements:

- The SOA4All Artefact Repository
- The SOA4All Deployment Description Language
- The SOA4All Deployment Design-time GUI

- The SOA4All Deployment Engine
- The SOA4All Deployment Runtime GUI

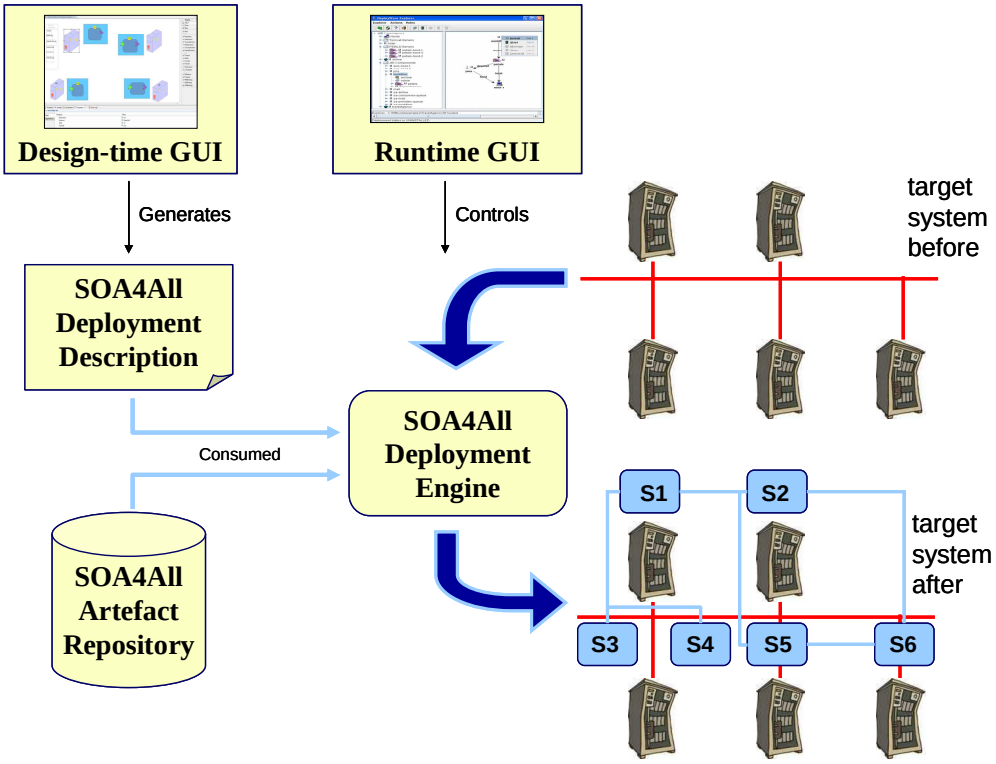


Figure 9: The SOA4All Deployment Facility

4.2.3 The SOA4All Artefact Repository

The SOA4All Artefact Repository is a repository where software and service artefacts (e.g., archives, binaries, etc.) can be stored and retrieved. At development time, this repository is filled by developers providing software and service artefacts and their associated metadata (e.g., name, version, author, system requirements, artefact dependencies, etc.). At deployment time, the SOA4All Deployment Engine queries this repository for retrieving software and service artefacts according to metadata defined into SOA4All deployment descriptions.

Different implementations of this component are to be considered as standard HTTP or FTP servers, Maven repositories, file systems, or databases to store SOA4All artefacts. We will build this component on top of the SOA4All Semantic Spaces (Deliverable D1.3.1A) in order to store semantic metadata attached to SOA4All artefacts and retrieve them via semantic queries.

4.2.4 The SOA4All Deployment Description Language

The SOA4All Deployment Description Language allows users to express SOA4All deployment descriptions, and can be seen as a kind of Architectural Definition Language (ADL [13]). With this language, SOA4All administrators describe the distributed service-oriented systems they want to deploy. These descriptions contain i) the set of physical nodes composing the distributed target system and their properties (their network address, supported network protocols, operating system, etc.), and ii) services to deploy and their properties (the target node, the required artefacts to upload, the installation place, service configuration parameters, and dependencies to other services and software). The SOA4All Deployment Engine consumes these descriptions in order to deploy described distributed

service-oriented systems automatically.

Annex D gives an example written with a user-friendly but extremely simple SOA4All Deployment Description Language. However, this language can take more complex forms like EMF meta-models, XML-based Schema, or EBNF, but then descriptions could become less human-readable.

4.2.5 The SOA4All Deployment Design-time GUI

The SOA4All Deployment Design-time GUI is the user interface that allows SOA4All administrators to graphically design their SOA4All deployment descriptions instead of writing them directly with the SOA4All Deployment Description Language. Before deployment time, administrators use this GUI to define graphically the distributed service-oriented systems they want to deploy, and the GUI generates SOA4All deployment descriptions conform to the syntax of the description language.

The following image (Figure 10) illustrates a possible ad hoc graphical notation for designing SOA4All deployment descriptions. This example represents a complex distributed SOA-based travel agency system composed of two Web applications running on two Tomcat containers and ten JBI components distributed on three PEtALS servers, the whole system is deployed on three physical nodes. In the context of such complex distributed systems, the graphical notation must allow administrators to focus firstly on the composition of the whole system (i.e., dependencies between services and software, relations between software and physical nodes), and secondly on the properties of each individual service/software/host.

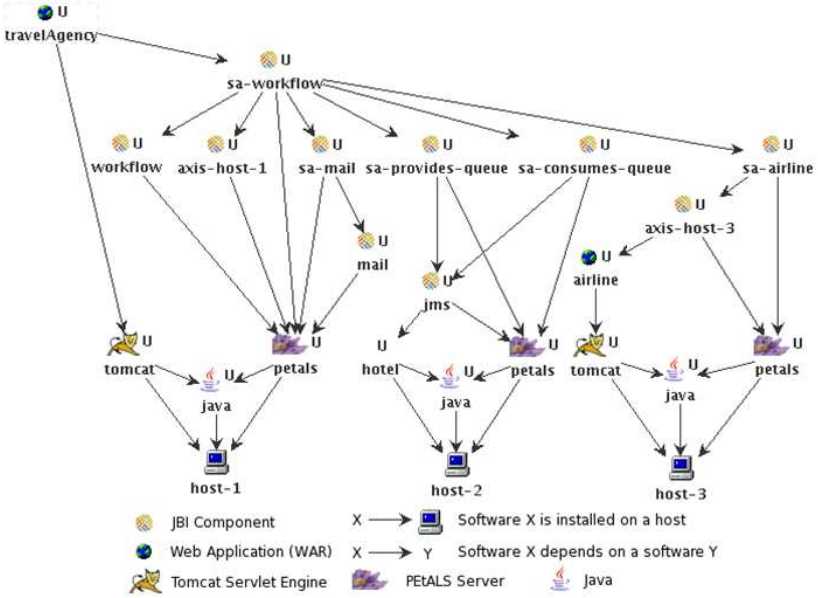


Figure 10: Illustration of a Graphical Notation for SOA4All Deployment Descriptions

Moreover, the SOA4All Deployment Design-time GUI must also apply static verifications on users' descriptions to avoid describing systems not deployable. This encompasses checks to verify that 1) all mandatory properties of each service/software/host are set like the `host`, `archive`, `home`, `host`, `hostname`, `petals`, `transfer`, `protocol` properties in the example of Annex D, and 2) no conflicts between service/software to deploy exist like deploying a Java-based service on a node where no JRE is deployed, or deploying two services using same Internet ports to receive client requests, etc.

Various graphical notations could be considered for the SOA4All Deployment Design-time GUI. We will focus next implementation efforts to support the non standard graphical notation used in Figure 10, but also the more standard and widely accepted OMG UML Deployment Diagrams graphical notation [15]. This GUI will be implemented on top of the Eclipse IDE in order to benefit of its portability and graphical capabilities. Note that this GUI can not be

included into the SOA4All Studio as one of its components because this GUI will be used to describe how to deploy the whole SOA4All platform including the SOA4All Studio and so before the SOA4All Studio and its components are deployed and available to use.

4.2.6 The SOA4All Deployment Engine

The goal of the SOA4All Deployment Engine is to deploy distributed service-oriented systems automatically. At deployment time, SOA4All administrators provide SOA4All deployment descriptions as inputs for the SOA4All Deployment Engine. This engine queries the SOA4All Artefact Repository to retrieve required software and service artefacts according to metadata defined in SOA4All deployment descriptions. Finally, this engine executes the deployment onto the distributed target system automatically, i.e., uploading and installation of software artefacts, configuration and launching software and services according to their dependencies.

The SOA4All Deployment Engine can be accessible as a service, allowing for instance the SOA4All Deployment Runtime GUI and any other programs to control it remotely. Moreover the SOA4All Deployment Engine can be distributed on several nodes for addressing load balancing and efficiency issues. Finally, this engine must be extremely extensible in order to support various deployment technologies, deployment acts, network transfer and access protocols, service artefact packaging formats, etc.

For all these reasons, the SOA4All Deployment Engine is implemented on top of DeployWare/FDF, an open source framework[12] developed by INRIA and dedicated to deployment of large-scale distributed and heterogeneous software systems [9] [10]. DeployWare is a deployment-specific meta-model to specify any kind of distributed and heterogeneous software system deployments and to verify some safety properties statically. FDF is the virtual machine to execute and manage concrete deployments addressing heterogeneity, orchestration, and scalability challenges. As PEtALS, FDF is implemented on top of the Fractal component model [5] [6] in order to be highly extensible and dynamically configurable.

4.2.7 The SOA4All Deployment Runtime GUI

The SOA4All Deployment Runtime GUI is the user interface to control the SOA4All Deployment Engine. This interface allows SOA4All administrators to load SOA4All deployment descriptions into the SOA4All Deployment Engine, and to control the deployment of the described distributed service-oriented systems. Figure 11 illustrates what the look and feed of the GUI would be. The left panel allows users to navigate into SOA4All deployment descriptions loaded in a SOA4All Deployment Engine. The right panel provides a graphical view of the dependencies between services, software, and physical nodes. A toolbar and popup menus allow users to execute deployment acts, as installation, starting, stopping, and deinstallation of services and software stacks.

[12] A preliminary version is available at http://fdf.gforge.inria.fr.

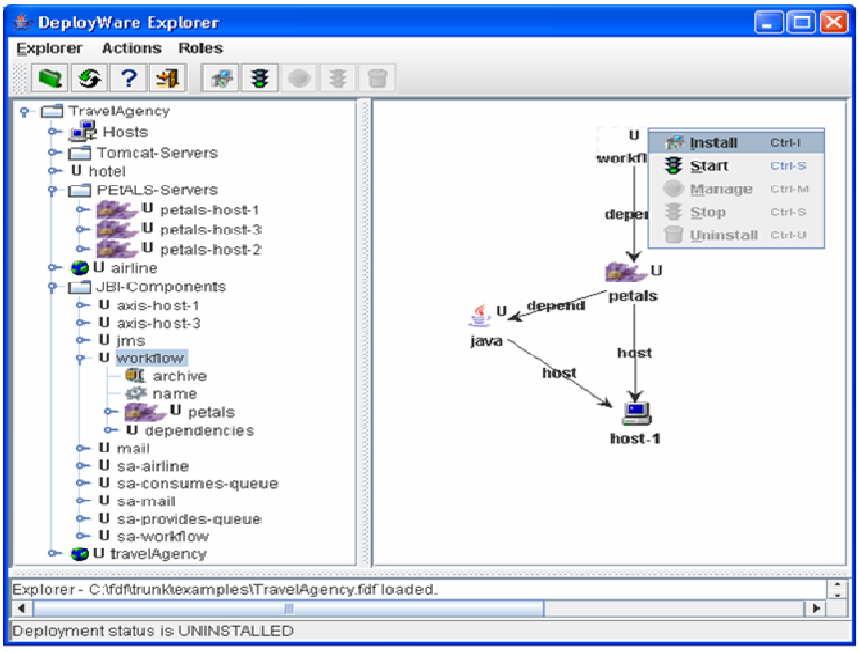


Figure 11: Illustration of the SOA4All Deployment Runtime GUI

The implementation of the SOA4All Deployment Runtime GUI will be based on the existing DeployWare/FDF Explorer GUI, and will be extended to be a Web 2.0 user interface integrated into the SOA4All Studio.

4.3 Monitoring Platform

SOA4All envisions a Web of services where billions of services are seamlessly provided and consumed by billions of users over the Web. This ambitious goal necessarily requires the inclusion of advanced monitoring and management mechanisms able to propagate and derive detailed information about the execution of services as well as offering the means for controlling or adapting the runtime infrastructure to better deal with ongoing situations. In this section we shall present the architectural decisions adopted in order to provide fully advanced monitoring and management facilities for SOA4All. This will be provided by what we refer to as the monitoring platform which will support the monitoring and management of the SOA4All infrastructure, the monitoring and management of SOA4All composite services (processes or mashups), and the monitoring of external services invoked through the SOA4All runtime infrastructure.

The information generated by the monitoring platform is required and consumed by the Service Analysis Platform, described in D2.3.1. Among these requirements we can find:

- Scalable storage and querying of monitoring information
- Notification mechanism of monitoring events
- Execution of management commands

To address these requirements, the DSB will provide several data collectors which will be accessible to other parts of the architecture, specially the Analysis Platform of the SOA4All studio.

- Bus collector. Includes the higher level information of the nodes hosting the DSB. This kind of information can be obtained through the current PEtALS data collector, which works as a monitoring entry point in a PEtALS bus architecture, allowing collecting data about the message exchanges among the service components; finding the containers involved in a specific service, and collected statistics related to

this service. This kind of collector will be extended along with the DSB implementation, permitting to extend this statistics collector to the federation of DSBs.

- Grid collector. Details about the infrastructure used by the DSB architecture. This includes nodes and clusters used for hosting the DSB services. This kind of information can be obtained from the ProActive nodes hosting the DSB and the Semantic Space through JMX notifications which can be subscribed by a client application (for example, IC2D), and derive lower-level details like process load in the machines, CPU/memory usage, or number of nodes involved in a computation.
- Engine collector. Information on the services accessible through the SOA4All runtime, which can be single services currently hosted on the DSB, external third-party services which are being accessed through the DSB facilities, or compositions of services which are also finally hosted on the DSB. For capturing information on these compositions, the implementation SCA composites can be instrumented with probes and actuators that gather information on the involved components and expose this information to the collector.

In the remainder of this section we will present these details by addressing i) the techniques for generating raw monitoring information; ii) the technologies utilised for propagating this information to interested components; iii) the approach for processing the information; iv) the means for storing this information; and v) the ways through which monitoring and management functionality will be offered to applications.

4.3.1 Raw Monitoring Data Generation

Raw monitoring data generation is concerned with the generation of information at runtime about the ongoing execution of software components. As previously introduced, three main kinds of components being monitored are distinguished: SOA4All composite services, external services being invoked through the SOA4All runtime as part of the execution of some composite service, and the SOA4All runtime infrastructure. The nature and the control over each of these types of components differ, thus, the capability for controlling what information can be generated varies.

Data Generation for Composite SOA4All Services

Monitoring composite services (processes or mashups) defined within SOA4All is a must from the perspective of the user or company that defined the service in the first place, as well as from the perspective of the user or company consuming it. Composite services have the particularity that they are domain-specific user-defined orchestrations of services. There are therefore (virtually) no limits with respect to the kinds of orchestrations that can be defined and nothing can be assumed about them. On the other hand however, they are specified using SOA4All technologies and executed by SOA4All components. We therefore know their definition (what process they have to perform) and we have control over the components orchestrating their enactment (e.g., Process Execution Engine, Discovery Engine).

We previously defined in the context of the project SUPER (IST-026850), the Core Ontology for Business pRocess Analysis (COBRA), and a reference Events Ontology (EVO) that provides a set of definitions suitable for capturing monitoring information from a large variety of systems [32]. EVO was successfully applied in SUPER and used to monitoring business processes enacted by an open source BPEL execution engine [36]. We shall therefore reuse the same conceptual model as a starting point for capturing logs of composite services within SOA4All. To do so, it will be necessary to ensure that the Process Execution Engine devised in WP6 generates logs in the aforementioned format which will be adapted for being represented in RDF/RDFS. As a result we shall be able to ensure that an important part of previous research can be used as a solid basis for supporting monitoring in SOA4All. More details how these conceptual models will be used for supporting higher-level analysis are presented in Section 4.3.4 when it comes to architectural and infrastructural concerns and in

D2.3.1 for user-defined domain-specific monitoring and analysis.

Data Generation for External Services

The execution of simple or composite services defined within SOA4All will eventually involve the invocation of services (Web Services, RESTful services, or any service provided by the PEtALS ESB connectors) owned by a third party and deployed in a server we do not have access to other than for service invocation. For this kind of services, the only aspects we can monitor are therefore the interactions the SOA4All runtime infrastructure has during the enactment of SOA4All services.

The information we shall capture is therefore the messages exchange with remote services. Messages both sent to and received from external services during invocation through the DSB will be forwarded to the monitoring infrastructure.

Data Generation for the Runtime Infrastructure

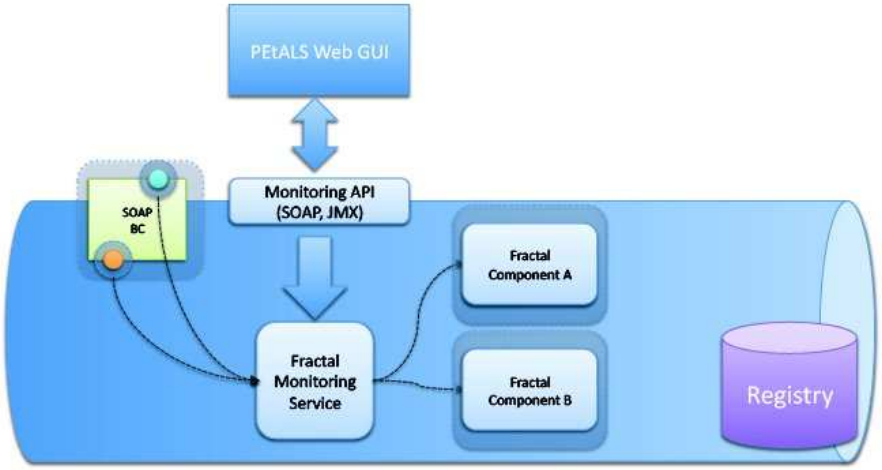
Monitoring the SOA4All infrastructure can address the needs for IT personnel devoted to identify problems or bottlenecks. We understand as runtime infrastructure both the communication and storage platform provided by the DSB as well as the infrastructural services such as the Discovery Engine or the Process Execution Engine. Both cases have the specificity that they are designed and developed within the project. We can therefore ensure that they generate the monitoring information we may require using the protocols and formats we establish. Furthermore, as we shall describe in more detail in Section 4.3.4, we know what their purpose is and how they work internally and we can therefore devise specific monitoring modules for controlling them.

Infrastructural services can basically be seen as very specific and well-known processes that are recurrently enacted for supporting the interactions with the user. This includes the execution of user-defined composite services, or intermediate activities like discovering services. The generation of monitoring information about the execution of infrastructural services will be implemented by the container, which produces logs in container format and transforms into EVO based format. The information logged shall constitute the basis for analysing the execution of infrastructural services, with the particular specificity that the process is well-known in advance and we can therefore embed additional knowledge for detecting, diagnosing and correcting deviations. The service container also exposes specific APIs for access to the monitoring data of infrastructural services.

4.3.2 Monitoring Data Communication

The information gathered at runtime using the techniques outlined in the previous section need to be forwarded to interested components for further processing. This includes monitoring data processing components that aggregate the data or process it in order to detect, predict and possibly correct relevant situations (e.g., anomalies, etc), as well as monitoring data visualization software that will take both raw and derived information and will display it in a form that is better suited for being analysed by humans (e.g., charts).

Indeed, neither all captured information is relevant to every component, nor relevant information does not always need to be processed at the same time. Critical information should be available for further processing as soon as possible whereas information which is



not that important can be processed at later stages. Finally, there may be several distributed components interested about certain monitoring data. In order to cater for this, monitoring data will be communicated by two main modes i) on demand by querying for concrete monitoring data through the monitoring API (see Section 4.3.5); and ii) automated notification when relevant monitoring information has been generated.

Figure 12: DSB Node Monitoring

Because the SOA4All infrastructure will immerse in a Web-based open environment, Fractal monitoring service exposes WS-Notification (WSN) [26] based monitoring API which can be consumed by monitoring tools such as the PEtALS Web GUI (as shown in Figure 12). WSN specify the interaction based on publish/subscribe pattern which can be used alongside the fundamental request/response pattern in service-oriented system. Fractal monitoring service will play a role as notification producers stated in WSN specification, while either the Web GUI or data aggregator (vide infra) will work as notification consumers.

Due to the inevitable Heisenberg Effect [27] resulting from the intrusiveness of software based monitoring, data communication should be controllable and configurable. Thanks to the additional benefits brought by WS-Topics [28], another member of WSN family, the category hierarchies of events can easily be described, and subscriptions can be appended by a topic filter which is used to restrict monitoring data propagation to the degree to which the service bus can endure the invasion. Consequently, risks of over-consumption of system resources are reduced.

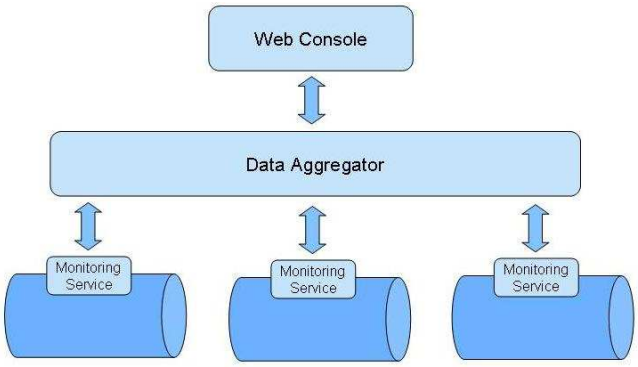


Figure 13: Monitoring Data Aggregator

In order to implement global monitoring, a data aggregator is set up (as shown in Figure 13), which works as the notification broker defined in WS-Brokered Notification [29]. A notification broker can be regarded as both notification producer and consumer, namely, it subscribes to notifications produced by some producers (the PEtALS containers) and also publishes them to some other consumers (the monitoring console). As a result, the proposed data aggregator subscribes to events produced the aforementioned bus, grid and engine monitoring data collector and transfers them to the Web console.

4.3.3 Monitoring Data Storage

Monitoring data storage refers to publishing both the raw monitoring data and analysis results to the Semantic Space. As specified in D1.3.2A, the Semantic Space infrastructure provides an open, scalable and distributed storage approach for semantic data. RDF encoded monitoring data and analysis results will be accessible both internally and externally so as to achieve runtime monitoring from an overall point of view.

The Web Service Distributed Management OASIS working group has published a set of specifications which define how to use Web Services to manage modules and how to manage Web Services with Web Services. These specifications will be used in the monitoring platform to define the format of the monitoring data. As stated, COBRA and EVO are two well-established ontologies and also will be re-used as the metadata of monitoring

data so that allows the instantiations to be serialized in RDF. Refer to Annex E for more details about EVO. With support of Semantic Space, data aggregator or Web console can retrieve relevant information by executing either triples pattern-based or SPARQL queries.

4.3.4 Monitoring Data Processing

The following components participate in processing gathered monitoring data, as shown in Figure 14:

- Message Exchange Pattern (MEP) Detector
- Event Pre-Processor
- Notification Dispatcher

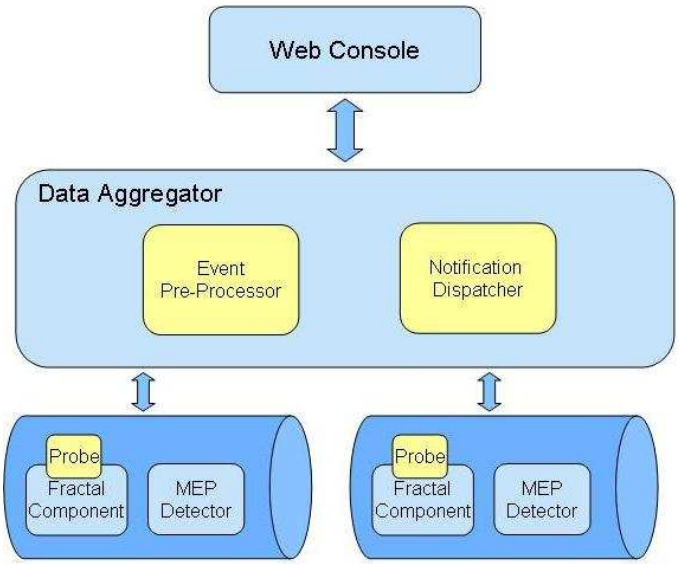


Figure 14: Components of Monitoring Data Processing

A MEP Detector recognizes patterns of message exchange during invocation of external services and maps them to concepts in COBRA or EVO ontologies. MEP describes both provider and consumer involved in a service call or operation and can be recognized by a template of the sequence of message exchange. The W3C has already formalized Web services message exchange into a set of patterns, i.e. In-Only, Out-Only, In-Out, Out-In, Robust In-Only, Robust Out-Only, In Optional-Out, Out-Optional-In, etc [24][25]. In addition to these pre-defined patterns, others adopted by the MEP detector will be documented in the way proposed in [23].

The Event Pre-Processor carries out the necessary steps of event processing before it arrives at the analysis platform, for instance, transforming event into the proper description format. Furthermore, engines with support of active rules or event-condition-action rules (e.g. Drools) can also help implementing diverse monitoring events processing modes: continuous, on-demand and periodically. In this way, administrators can respectively specify proper monitoring mode for every monitored object.

The Notification Dispatcher deals with issues of WSN-based notifications receiving and forwarding. It also accepts subscription requests from the SOA4All Web Console.

4.3.5 Monitoring and Management Interface

The SOA4All Distributed Service Bus is based on monitorable and manageable technologies such as PEtALS ESB or ProActive framework. The SOA4All monitoring and management interface (namely SOA4All M&M API) will encapsulate these tools APIs (called low level APIs) and add new functionalities so that it will be possible to manage and monitor all from a

common API.

As a result, the M&M API will expose entire or part of:

- The PEtALS management API: The PEtALS management API is built on top of the JBI management API which provide capability to install/start/stop/uninstall JBI components, activate/inactivate JBI endpoints, and perform other operations of JBI endpoints.
- The ProActive management API: The ProActive monitoring and management API uses a JMX connector to monitor the nodes, active objects or components hosting a grid application, and obtain low-level data of the infrastructure. It is also able to trigger actions on the provided Mbeans. Client applications, like the SOA4All M&M API can subscribe to these events to get grid level information.
- SOA4All specific API. This API part provides all the capabilities which are not offered by the low level ones and which have been introduced in the previous sections e.g. monitoring data access, managing components, applying rules and filters, to name a few.

The second M&M API part will expose monitoring data generated from low level APIs and from SOA4All level data. Like in other monitoring modules introduced before, it is a good choice to use the WSN specification family so that monitoring consumers can subscribe to specific notifications/events provided by the bus. As an illustration, the PEtALS Enterprise Service Bus JMX monitoring API provides data on service response time, service usage, message content and more. The SOA4All monitoring layer will get this data and pre-process it (filtering, statistics) before WSN API exposition.

The global monitoring API will embed some advanced processes which will be activated by the management part of the M&M API. These processes will potentially correlate data from different low level APIs to get more knowledge about how services and infrastructure interacts together. Rules and filters can be applied to this correlated data, and triggers can be registered to throw monitoring events/alarms when some goal is reached (alarms on CPU or memory usage is a good example). As a result, we can imagine that services will be migrated between nodes, new nodes will be deployed, or some runtime parameters will be adaptively adjusted. In a coherent way, all of these processes and adaptations will be activated with the help of the SOA4All management API.

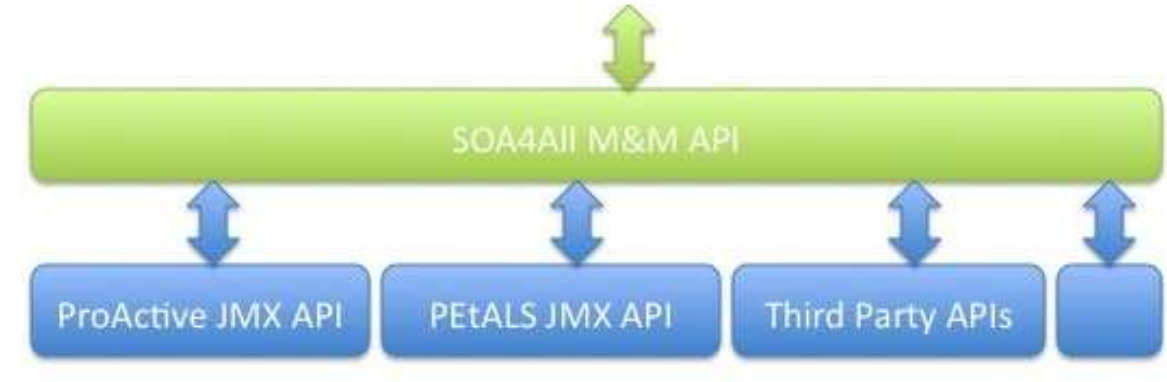


Figure 15: SOA4All M&M API

As shown in the Figure 15, the SOA4All M&M layer wraps, processes, aggregates the low level APIs and exposes a common monitoring and management API. Clients will connect to this agnostic API to manage the DSB and to monitor it.

5. Architecture Components and Interaction Matrix

5.1 Components

In the following we present briefly the different components of the SOA4All platform. The first two are core infrastructure services that are delivered by WP1 and that are presented in more detail in this deliverable and in D1.3.2A. The subsequent components are SOA4All Platform Services, focused mostly to service provisioning respectively service composition tasks. Besides a brief outline of the functionality and role of each component, the paragraphs depict the objects a component consumes (i.e., takes as input), the objects it produces (i.e., offers to other components), and a list of the components it interacts with (as consumer or producer of communication objects). The interactions between the various components are explained in more detail in Section 5.2, while the conceptual models and further technical details of the communication objects are given in Section 6.

5.1.1 SOA4All Distributed Service Bus

The Distributed Service Bus is the core infrastructure service of SOA4All and serves as communication broker for the integration of the SOA4All Platform Services and third-party business services.

- Consumes: JBI BC invocations, depending on the message format
- Produces: JBI BC invocations, depending on the message format
- Interactions: SOA4All Studio (Analysis Platform), all possible services

5.1.2 Semantic Space

The Semantic Space infrastructure serves as a global communication and coordination infrastructure in which any piece of RDF data can be stored and shared. The infrastructure is accessible via SOA4All DSB, and is thus exposed as an integral part of the core SOA4All service infrastructure. The Semantic Spaces are for instance used to store semantic annotation of Web services (cf. Service Registry). Furthermore it is also used to store process descriptions, monitoring data or other semantic artefacts in RDF.

- Consumes: Service Descriptions (in RDF), Processes (in RDF), any artifacts in RDF, SPARQL queries
- Produces: Service Descriptions (in RDF), Processes (in RDF), any artifacts in RDF
- Interactions: Distributed Service Bus (Monitoring), Service Registry, Design-Time Composition, SOA4All Studio (Provisioning Platform, Process Editor)

5.1.3 WSML Reasoning Framework

This is a framework of robust and scalable reasoning components tailored for each of the WSML language variants. A variety of interfaces allow for schema/instance reasoning, satisfiability/entailment checking and query answering. The reasoning framework is mainly used during discovery and composition, and is available as service to any entity in need of reasoning support. The reasoning framework has links to the Service Registry, and the Semantic Space infrastructure to load the necessary service descriptions and ontologies to conclude the desired reasoning processes.

- Consumes: Service Descriptions (in WSML, RDF), Ontologies, Queries
- Produces: Answers to Queries (variable bindings, concept/instance values)
- Interactions: Service Registry, Semantic Space, Ranking, Discovery, Composition Optimizer

5.1.4 WSMO Data Grounding

The data grounding component provides data lowering and lifting support for interoperability between WSML and XML. There will be two types of data grounding implemented: runtime and design time. The runtime grounding component will be used before and after invocation of services and will provide two operations: Lowering - WSML message serialised to XML using the pre-defined mapping schema (XSLT) -, and lifting - transforming the input XML message into WSML using a predefined mapping schema (XSLT) -. The design time mapping component provides functionality for semi-automatic generation of an ontology and a mapping schema from an WSDL input description. From the input WSDL (more specifically the XML schema describing the format of messages exchanged) a WSML ontology will be auto-generated, together with the lifting/lowering mapping definitions that allow the translation between the source XML and the target WSML. Additionally a GUI component will be provided that will allow the user to "beautify" the auto-generated ontology and mappings.

- Consumes:
  - Design time: Service description (WSDL), lifting/lowering mappings (XSLT, XSPARQL)
  - Runtime: Lifting/lowering mappings (XSLT, XSPARQL), XML or Ontologies (WSML)
- Produces:
  - Design time: Ontologies (auto-generated WSML), lifting/lowering mappings (XSLT, XSPARQL)
  - Runtime: XML, Ontologies (WSML)
- Interactions: Execution Engine (at runtime), SOA4All Studio (Provisioning Platform, at design time)

5.1.5 Crawler

A crawler takes care of detecting technical descriptions associated with Services including all related documents such as homepage, documentations, pricing and licensing information etc. on the Web. The resulting output of this component is offered as a service to SOA4All partners, but the component as such is not exposed as a Web service to the core SOA4All platform. The resulting output from this component can be obtained in the form of (1) RDF metadata stored in the Service Registry, or (2) data stored in Heritrix archive files. We additionally develop a set of the simple access methods (crawler API) facilitating access to both metadata and data. We consider currently to either providing Java API or the REST endpoint with the set of methods allowing for retrieval of the crawled documents. The user interface of the crawler allows defining all necessary parameters and requirements for the crawler.

- Consumes: All necessary parameters to schedule focused crawl. Set of algorithms allowing detection of required documents on the Web and deciding which links to follow, and which to abandon.
- Produces: Service Description (WSDL) plus related documents, RDF metadata about the documents, Service Description (WSMO-Lite, MicroWSMO)
- Interactions: SOA4All Studio (Provisioning Platform), Service Registry

5.1.6 Service Registry

The Service Registry is closely bound to the Discovery service, and is a dedicated infrastructure to store and manage service descriptions. The provisioning platform upon user-driven creation of a service annotation, and possibly the Crawler, publish service descriptions

in the repository, while the discovery service, but also the process generation components access the repository. While the repository has no dedicated storage infrastructure bound to it – this is up to a given repository realization – in SOA4All we apply the Semantic Space infrastructure to store service annotations in RDF.

- Consumes: Service Descriptions (in WSMO-Lite, MicroWSMO), Service Identifiers
- Produces: Service Descriptions (in WSMO-Lite, MicroWSMO), SPARQL queries
- Interactions: SOA4All Studio, Discovery, Semantic Space

5.1.7 Discovery

The service discovery component enables users to find service appropriate for their needs. For this purpose, it provides two types of discovery mechanisms, full text based discovery and semantic discovery. Full text based discovery uses the Web service descriptions (WSDLs) and related documents that were crawled by the crawler and allows users to search for services by entering keywords (much like typical Web search engines). The semantic discovery component allows users to enter a more structured goal (service classification, pre-conditions and effects) and finds the service that match the goal by using the reasoning facilities provided by WP3.

- Consumes: Service Descriptions (in WSMO-Lite, MicroWSMO), Goal, Keywords, Service Identifiers
- Produces: Service Identifiers, Service Descriptions (in WSMO-Lite, MicroWSMO)
- Interactions: Service Registry, Crawler, Ranking, Design-Time Composer, Composition Optimizer, Execution Engine, SOA4All Studio (Consumption Platform)

5.1.8 Ranking and Selection

The service ranking and selection component is responsible for ranking services according to user preferences on the non functional properties of the services. This component offers two types of ranking techniques (1) context-independent ranking and (2) context-dependent ranking. The context-independent ranking component ranks the services on the basis of non-functional properties that are of global nature whereas the context-independent ranking also takes the user preferences into account

- Consumes: Service Descriptions (WSMO-Lite, MicroWSMO), User Preferences (NFP),
- Produces: Service Descriptions (WSMO-Lite, MicroWSMO; ordered list)
- Interactions: Discovery, SOA4All Studio (Consumption Platform)

5.1.9 Design-Time Composer

The design-time composer component will carry out two closely related activities, namely process adaptation and composition, offering thus two external interfaces, the Service Composer and the Service Adapter.

- The Service Composer perspective is a scalable system for the flexible and ad-hoc creation of complex services, the environmental context information, and user needs (expressed using the lightweight modelling language).
- The Service Adapter is a subsystem for service context-based adaptation at design-time. The first and basic usage of this tool is the adaptation of services according to context (e.g. personal preferences, business rules, etc.), but also more advanced dynamic adaptation procedures. Mechanisms such as incremental revealing of services descriptions imply that not all the service characteristics have to be revealed at once, but require a reciprocal knowledge, trust and a negotiation process between

parties.

The two activities are closely related, since both composition and adaptation will share the same core of functionalities that are provided by a parametric design engine which will use a catalogue of generic knowledge-level service templates and context-dependant configuration.

- Consumes: Process, User goals and requirements
- Produces: Process
- Interactions: Discovery, Composition Optimizer, Template Generator, Execution Engine, SOA4All Studio

5.1.10 Template Generator

In several complex industrial situations it is not possible to define an "a priori" template for a process, either due to the complexity of the real situation or to the high effort required to formalise such a template. The problem is thus to understand what is the typical workflow followed by the various activities, in order to formalise their sequence in a composed schema. The Template Generator will be able to analyse service execution logs and to generate a hierarchy of process schemas (at different levels of complexity/completeness) and a taxonomy of possible process templates (at different levels of abstraction), in order to support end-users in the selection of the most suitable one. Such a tool will exploit state-of-the-art process mining, process abstraction and clustering techniques in an innovative way, in order to present the end-users with the most suitable template representations and let them choose the one that most fits their needs. The selected template (described with the SOA4All light-weight process language - D6.3.1) can be further validated, adapted or refined by end-users thanks to the SOA4All Process Editor developed in Deliverable D6.2.1.

- Consumes: Services execution logs and monitoring data
- Produces: Process (abstract)
- Interactions: Distributed Service Bus (Monitoring), Design-Time Composer, SOA4All Studio (Process Editor), Semantic Spaces (Execution Logs)

5.1.11 Composition Optimizer

Given an abstract lightweight process the composition optimizer aims at discovering an optimal and executable lightweight process of Semantic Web services that achieves a specific goal. Since an abstract lightweight process consists of generic goals, each goal requires to be assigned with a relevant and executable Semantic Web service in order to compute such an optimal and executable lightweight process. To this end, the composition optimizer considers an innovative and extensible quality criteria model by coupling non-functional quality of service and semantics of the executable lightweight process. On the one hand the non-functional criteria of Web services are valued by means of Quality of Services (e.g. execution price, response time, reliability, availability) while on the other semantics is valued along the semantic links (i.e. data flow in an executable lightweight process) between Web services. The latter criterion requires the WSML reasoning framework to i) give an estimation of semantic matching between functional output and input parameters of services and ii) estimate robustness issues (through a non-standard Description Logics inference) in data flow of any executable lightweight process. In regards to the latter criteria the problem is formalized as a Constraint Satisfaction Problem (CSP) with i) multiple constraints and ii) a function to optimize. Towards such an issue we model an optimization problem COP (Constraints Optimization Problem), adapted from CSP. Since one of our main concerns is about optimization of large-scale executable lightweight process (i.e., many services can achieve a same goal or functionality), we suggested to follow a Genetic Algorithm-based approach which is faster than applying Integer Linear Programming.

- Consumes: Process, Preferences (NFP), Monitoring data
- Produces: Refined Process
- Interactions: Design-Time Composition, Discovery, Reasoning Framework, Execution Engine.

5.1.12 Execution Engine

The execution engine is an execution infrastructure for lightweight processes, adaptive to environmental changes and flexible enough to allow its context-dependent self-reconfiguration. In SOA4All the term "execution infrastructure for lightweight processes" mainly refers to model and execute composite services and processes in a lightweight manner as described in Deliverable D6.3.1. This execution infrastructure will exploit a set of basic mechanisms such as dynamic discovery, selection, adaptation, invocation, monitoring developed in the other work packages, for supporting the dynamic and adaptive reconfiguration in reaction to environmental changes.

- Consumes: Process, Execution Values, Monitoring Data
- Produces: Effect of the execution
- Interactions: Design-Time Composer, Composition Optimizer, Discovery, Distributed Service Bus, Data Grounding

5.1.13 SOA4All Studio

The SOA4All Studio consists of three main subcomponents that are described independently of each other in the following for clarity.

Provisioning Platform: The Provisioning Platform has two main purposes. First, it provides the necessary tools to annotate services, either WSDL services via WSMO-Lite, or REST APIs via MicroWSMO. Second, it incorporates a Process Editor (Composer) that allows users to create, modify, share, and annotate executable process models based on the lightweight process modelling language defined in D6.3.1. The functional and technical specification of the Provisioning Platform can be found in D2.6.1 (Process Editor) respectively D2.1.3.

- Consumes: User Input (via GUI), Service Descriptions (WSMO-Lite, MicroWSMO, WSDL, HTML), Process
- Produces: Service Descriptions (WSMO-Lite, MicroWSMO), Processes, User Feedback (RDF)
- Interactions: Discovery and Location, Design-Time Composition, Template Generator

Consumption Platform: The Service Consumption Platform (D2.2.1) is the gateway for users to the service world when they act as consumers. The platform allows them to formalise their desires in several ways, defining and refining goals that can be used to discover and invoke the services that fulfil their needs. The platform stresses the characteristic of personalisation, making use of contextual factors to offer a more suitable service consumption to the users, adapting the services and some characteristics in the platform as well (e.g., the recommended Goals for different users, based on their past experience within the platform).

- Consumes: User Input (via GUI), Goal, Service Descriptions
- Produces: Effect of execution, Execution Log (RDF)
- Interactions: Discovery, Ranking and Selection, Semantic Spaces

Analysis Platform: The Analysis module obtains information (monitoring events) from the monitoring subsystem and performs processing in order to extract meaningful information.

Monitoring events should come from data collectors that perform basic aggregation from distributed sources in the runtime infrastructure; they can be JMX (or SOAP etc) notifications coming from the DSB, the underlying grid and the execution engine. Other inputs contain requests from SOA4All Studio components that will include the name of the services/processes that need to be represented with analysis information.

- Consumes: Monitoring events (Execution Engine, Distributed Service Bus).
- Produces: Monitoring data (RDF)
- Interactions: Repository, Execution Engine, Distributed Service Bus, SOA4All Studio

5.2 Interaction Matrix

In this section we consider the various components of the SOA4All architecture in their context to the other core infrastructure and platform services.

	DSB	Space	WR	DG	C	SR	D	R	DTC	EE	TG	CO	Studio
DSB (Monitoring)										Produces		Produces	Consumes/ Produces
DSB (Semantic Space)			Produces			Consumes/ Produces			Consumes/ Produces		Produces		Consumes/ Produces
WSML Reasoning (WR)		Consumes				Consumes	Produces	Produces				Produces	
Data Grounding (DG)										Produces			Produces
Crawler (C)						Produces	Produces						Produces
Service Repository (SR)		Consumes/ Produces	Produces		Consumes		Produces						Consumes/ Produces
Discovery (D)			Consumes		Consumes	Consumes		Produces	Produces	Produces		Produces	Produces
Ranking (R)			Consumes				Consumes						Produces
Design-Time Composer (DTC)		Consumes/ Produces					Consumes			Produces	Consumes	Produces	Produces
Execution Engine (EE)	Consumes			Consumes			Consumes		Consumes			Consumes	
Template Generator (TG)		Consumes							Produces				Produces
Composition Optimizer (CO)	Consumes		Consumes				Consumes		Consumes	Produces			
SOA4All Studio	Consumes/ Produces	Consumes/ Produces		Consumes	Consumes	Consumes/ Produces	Consumes	Consumes	Consumes		Consumes		

Figure 16: Interaction Matrix of SOA4All Platform Services

Figure 16 provide an interaction matrix that shows that dependencies between the different components. The indicators 'Produces' and 'Consumes' have the same meaning as in Section 5.1:

- Produces: the component to the left offers some object(s) to the component at the top of the table; and
- Consumes: the component to the left takes as input some object(s) from the component at the top of the table.

The relationships that are depicted in the interaction matrix are explained in terms of two main process of SOA4All (cf. Section 7.4): discovery of a (semantic) service, and creation of process.

The discovery process is triggered at the level of the SOA4All Studio where a user can indicate a request (either some keywords for full text-based search, or a goal description of is objectives). In other words, the SOA4All Studio provides a query interface for services to the users, which is bound to the discovery component. Note that all interactions between various services are performed via the SOA4All Distributed Service Bus. The discovery component makes either use of crawled data (if keyword-based search was chosen), or uses the reasoning framework in the case of goal-driven discovery. The service descriptions, ontologies and further facts are loaded by the reasoning framework upon request by the

discovery engine. While the service descriptions are maintained in the service registry in terms of WSMO-Lite and MicroWSMO data, the ontologies and facts are directly loaded from the Semantic Space infrastructure. Note, that the service registry, too, is realized on top of Semantic Spaces. However, it provides further functionality like indexing or the transformation from WSMO languages to RDF. In order to publish and query RDF data, the Semantic Spaces expose a SPARQL interface via the SOA4All Distributed Service Bus. There is one more SOA4All Platform Service (indirectly) involved in the discovery process, the crawler. The crawler gathers the WSDL files that are, in form of annotations, stored in the Service Registry, and moreover collects further facts about services that are stored in the Semantic Space and consumed by either the discovery component or loaded by the reasoner. A summary of these interactions is given in Table 1.

Table 1: Interaction Details in Terms of Discovery

SOA4All Studio	← Service Description ; Keyword, Goal →	Discovery
SOA4All Studio	← WSDL, HTML	Crawler
SOA4All Studio	← Service Description →	Service Registry
SOA4All Studio	← Service Description	Ranking & Selection
Ranking & Selection	← Service Description	Discovery
Ranking & Selection	← Answer ; Query →	Reasoning
Discovery	← Answer ; Keyword →	Crawler
Discovery	← Answer ; Query →	Reasoning
Discovery	← Service Description	Service Registry
Reasoning	← Service Description	Service Registry
Reasoning	← Ontology (RDF)	Semantic Space
Crawler	Service Description →	Service Registry
Service Registry	← Service Description (RDF) →	Semantic Space

A further component that was not mentioned above is the ranking and selection one (given in Table 1). Ranking and selection is invoked by the SOA4All Studio in order to retrieve ranked or even pre-filtered discovery results.

The second interaction flow that we consider is built around the first one, i.e. the creation of process relies on the service discovery procedure described above (cf. Table 2). The first access point for the SOA4All Studio is the design-time composer that transforms graphical descriptions of service compositions into process objects (cf. Section 6.5). All subsequent components use the processes to refine the composition. The composition optimizer takes context and monitoring information into account to optimize the chosen service bindings under given constraints – this is an optional step. Eventually the execution engine takes over the process and binds all remaining Semantic Web services in order to expose and invoke a composed service. Although the precise definition of a process changes depending on the component handling it at a given time, the object shared amongst all the named components remains the same – a process. In order to store process descriptions persistently there is RDF formalism planned, which allows the publication of processes to the Semantic Space infrastructure.

In order to ease the creation of processes, the template generator creates abstract process descriptions (templates) from past compositions and execution logs. The template generator thus offers processes to either the user at the level of the studio or directly to the design-time composer.

Table 2: Interaction Details in Terms of Process Creation

SOA4All Studio	← Process →	Design-time Composer
SOA4All Studio	← Process	Template Generator
Design-time Composer	← Process	Template Generator
Design-time Composer	→ Process	Composition Optimizer
Design-time Composer	→ Process	Execution Engine
Design-time Composer	← Service Description	Discovery
Design-time Composer	← Process →	Semantic Space
Composition Optimizer	→ Process	Execution Engine
Composition Optimizer	← Service Description	Discovery
Composition Optimizer	← Monitoring Data	Distributed Service Bus
Composition Optimizer	← Answer ; Query →	Reasoning
Execution Engine	← Service Description	Discovery
Execution Engine	← Monitoring Data	Distributed Service Bus

The remaining connections depicted in Figure 16 are related to the analysis platform of the SOA4All Studio (access to monitoring and user data at the level of the bus and the spaces), and the lifting and lowering of XML respectively WSML representations. However, as stated in the component description (Section 5.1.4), grounding is not a SOA4All Platform Service, but rather a software component that is embedded at the level of the execution engine respectively the studio – for transforming annotations.

In the next section we present the objects that are exchanged in more details.

6. Communication Objects

This section presents in more detail the objects that are exchanged and shared amongst the various SOA4All components presented in Section 5.1: services and their descriptions, goals (as user objectives), ontologies, queries (to the reasoner framework or the discovery engine), and processes as composition of services and goals.

6.1 Web Service

SOA4All assumes the service concept as it is defined in the Reference Ontology for Semantic Service Oriented Architectures by the OASIS Semantic Execution Environment (SEE) TC, which in turn is based on the reference model specification of the OASIS SOA Reference Model TC: a service is "*a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.*" This specification identifies four aspects of services that are essential in any service-oriented architecture:

- A service enables access to one or more capabilities;
- A service enables access through a prescribed interface;
- A service is opaque to the service consumer except from the information and behavioural models in the interface and the information requires to assess if a service meets the requesters needs;
- Consequences of invoking a service should either be response information to the invocation or a change to the shared state of the defined interface.

It is important to note that the cited reference model makes a clear distinction between the capability of a service and the point of access where the capability can be consumed.

The semantic service-oriented architecture reference ontology considers thus the service capability, and the service interface to be the two core elements of a Semantic Web service. The reference ontology moreover states clearly that the service descriptions contain both functional notions, including behavioural aspects, and non-functional notions.

From a more technical perspective, SOA4All supports two kinds of Web services: traditional WS-* services that are described with WSDL and usually communicate via SOAP protocol, and emerging Web-based services that are often called RESTful APIs. WSMO-Lite (Deliverable 3.4.2) defines a minimal RDF model that captures Web service structures, along with a high-level ontology for service semantics, which is combined with domain ontologies in the descriptions of concrete services.

The minimal service model of WSMO-Lite has a top-level concept 'service', which offers any number of 'operations'. The service is a coarse-grained unit of discovery: it offers a specific functionality and it comes from a single provider. An operation is a unit of interaction between a client and a service. Depending on the message exchange pattern, an operation can have an input message, an output message, and also fault messages (as input or output).

On top of this model, WSMO-Lite describes service semantics of four categories: functional semantics (expressed on the level of a service) convey what the service does, or in other words, what functionality it offers. Non-functional semantics, also on the level of a service, describe policies, QoS parameters and restrictions and similar information, commonly used for ranking based on user constraints and preferences. Behavioral semantics, expressed on service operations, guide the semantic client in its invocation of the service. And finally, information model semantics, attached to the messages of operations, describe the data that is exchanged between the client and the service.

WSMO-Lite is defined over WSDL descriptions of Web services, using the semantic

annotations standard SAWSDL to point to service semantics. MicroWSMO (Deliverable 3.4.3) applies the WSMO-Lite service model and semantics ontology to RESTful services by means of the hRESTS microformat that builds the service model structure in otherwise unstructured HTML service documentations. MicroWSMO is a SAWSDL-like layer over hRESTS, serving to point to concrete service semantics. In effect, while WSMO-Lite builds on WSDL and SAWSDL, MicroWSMO and hRESTS expand the reach of WSMO-Lite to RESTful services. For more information, please refer to deliverables D3.4.2 and D3.4.3.

6.2 Ontology

Ontologies in SOA4All are defined by the ontology specification of the SEE TC reference ontology document, and in consequence ontologies in SOA4All will be specified by means of the WSML language. Other languages would be possible as well, e.g. OWL or even RDFS, however, ontologies in SOA4All are used for annotating services and user goals as well. Therefore, we follow the decision of the OASIS SEE TC and stick to the ontology definition facilities of WSML. In fact, WSML is the only language that allows the definition of ontologies, and attaching those to service and goal description with the same formalism.

Ontologies contain concepts, relations, instances and axioms in form of logical expressions. The resulting specification and terminologies are then used to formalize service description (cf. Section 6.1 above), and also user goals to later point in the project. The following definitions of ontology elements are derived from the reference ontology release candidate document by the OASIS SEE TC:

Concepts

Concepts provide a means for describing pieces of terminology and can be related to each other via a subclass-superclass relationship. Concepts define attributes that range over concepts and relations. Instances of the defined concepts then carry attribute values belonging to those concepts and relations ranged over, allowing relationships instances to be captured.

Relations

Relations enable the description of n-ary relationships that go beyond those captured as conceptual attributes between instances. Unlike attributes (binary relations) there is no source to the relationship but there is an arbitrary number (arity) of parameters typed as concepts and other relations so that instances capture multi-party relationships between instances.

Instances

Instances are identifiable or anonymous members of concepts and relations and also provide values to the attributes or parameters of concepts and parameters of relations respectively. Instances may be explicitly declared as members of concepts and relations or they may be implicitly included as members via axioms.

Axioms and Logical Expressions

Through the use of logical expressions, axioms define constraints that must hold over all contents of their containing ontology in order for this to be consistent. These can be used to support an explicit style of modelling, where instances and their concept memberships are declared explicitly and axioms merely constrain their allowed membership and attribute values (cf. relational database constraints), or intentionally, where concepts may be implicitly populated via axioms. In that way, logical expressions allow for formalisms that go beyond the mere definition of term hierarchies, relationships to instances and data values and their instantiations.

6.3 Goal

The goal concept in SOA4All is identical with the goal concept of WSMO, which in turn is to be standardized in the SEE Reference Ontology. While the SEE definition is more abstract than the WSMO version, it is conceptually the same.

The purpose of the goal in SOA4All is the formal specification of the objective (tasks or activities) that a user likes to have performed and for which fulfilment is sought. Consequently, the goal is an implicit description of the service or services that have to be executed in order to satisfy the desired request. In order to simplify the matchmaking process between a goal (i.e. a request) and a provider service, the SEE Reference Ontology defines a goal to be a dual model of the service description. In other words, the goal description consists of the same elements as the service descriptions introduced above. A goal has a capability model that represents the requested or desired functionality in terms of pre- and post-conditions, assumptions and effects, and an interface description that determines the interface a requester intends to use during the invocation process with a matching service.

6.4 Query

There are two different types of queries in SOA4All, at the level of two different components. The first type is a standardized SPARQL query, or subparts of it, as a simple triple pattern at the level of the repository infrastructure (Semantic Spaces). Spaces manage RDF data, and therefore the infrastructure enables the RDF query language SPARQL.

The second type of query is used at the level of the discovery component in terms of the underlying reasoning framework support. A discovery query consists of a set of WSML concepts that a matching service should be a member of, as well as desired pre-conditions and effects of a matching service in form of logical expressions. The expressivity of the WSML logical expression that is allowed depends on the WSML variant that the reasoner is set up to process. Detailed information about WSML logical expressions and the different language variants can be found in WSML Deliverable D16.1 at http://www.wsmo.org/TR/d16/d16.1/v1.0/.

If a WSML query is ground, i.e. does not contain variables, then the query will evaluate to true or false, depending on whether the (WSML) knowledge-base entails this statement. If the query contains variables then variable bindings will be returned, i.e. every unique combination of ground values that make the statement true.

Examples:

?x memberOf Person.

⇨ Returns all identifiers that are instances of 'Person'

?x memberOf Politician[age hasValue ?a] and ?a < 25.

⇨ Returns IDs and ages of politicians younger than 25

In addition to the logical expression-based querying of the reasoner framework, there are query language extensions defined in deliverable D1.4 of the SUPER project. SUPER defines a standalone query language for WSML by extending the WSML logical expression syntax with SQL-like aggregation functions.

6.5 Process

A process in SOA4All is an ordering of Semantic Web services and goal descriptions with associated parameter constraints (e.g. on communication parameters) and control flow information. Depending on the nature of the process, a process could be classified as either executable or as abstract in terms of not yet bound task descriptions (not ready to be bound

to an invokable service). The difference is thus that an executable process has all its tasks associated with a service, while the abstract ones have some task associated with a goal. Note that an executable process is exposed to users as a composed service, and hence perceived as single service and invoked as any other consumable service. Therefore, SOA4All refers to process, as being any composition of service and goal descriptions which jointly describe an eventually composable execution, while the final composition becomes a singular service again.

Although the definition of a process is clear (tasks in form of goals, service, control flow data and constraints), its representation can take various forms. At the level of the provisioning platform (SOA4All Studio) a process is a pure graphical artefact. Subsequently, the process is more and more refined towards an executable composition of services in course of the manipulations in the design-time composer (DTC), the composition optimizer and eventually the execution engine. The DTC's role is primarily to match the graphical model of the user to a process description of services and goals, together with the related constraints and control flow data. The optimizer takes contextual information and monitoring data into account to refine the process description, to define the data flow between Web services, and to optimize the input to the execution engine; this step is however not required, and is purely an optimization from the end-users point of view (mainly based on preferences and constraints on services and their composition quality). Finally, the execution engine resolves all the remaining goal descriptions and executes the whole process. At this point, as mentioned above, the process becomes a single service that is consumable by SOA4All users.

SOA4All moreover knows two further variants of processes that are respectively termed mash-ups and templates. Mash-ups are the simplest form of processes in that they are only defined at the level of data flow of Semantic Web services; i.e. no goal descriptions are embedded, nor is control flow data available. Mash-ups are executed by the Execution Engine, but due to their simplicity, they could also be directly executed at the level of the user interaction platform, the SOA4All Studio. Templates are available in order to simplify the modelling process. Templates are also abstract process descriptions that are produced by the Template Generator based on already defined processes and execution logs. Templates simplify the design phase, as they provide starting points for users, foster reusability and hence ease the composition process. We distinguish templates from processes, as templates are not part of the actual composition procedures, but only input to it, and their processing is seen to be orthogonal to the tasks of the composition components.

In SOA4All, a lightweight, context-aware process modelling language is used that is based on a small subset of the BPMN notation to describe the control flow of a process. Within a process model, we also specify activity goals as unbound activities that are bound to a particular service at runtime. We further allow the users to specify constraints, such as that certain tasks or activities in the process model must be performed by certain pre-required semantic Web services. In short, SOA4All currently uses "starts a process flow", "ends a process flow", "a task/activity unit", "sequence flow", "exclusive gateway", "parallel gateway", "inclusive gateway", and "activity goal" for describing the control flow. In future, we will specify the data flow of a process as well. Further details about the process modelling language of SOA4All are to be found in deliverable D6.3.1.

7. SOA4All Functional Processes

7.1 The Process Methodology

In the previous subsections the SOA4All technical architecture has been presented. The architecture shows the different components that SOA4All will provide and their main interactions. However, although the specification of the architecture shows the building blocks in which SOA4All is founded, there is a clear need of defining the internal processes that run across architecture components. On the other hand, the definition of a simple methodology to ease the integration process of components in the SOA4All architecture is also necessary.

The aim of the SOA4All process methodology is then twofold:

- To provide an explicit expression of what is meant by a SOA4All process methodology within the project, thereby providing a vehicle for attaining and maintaining a common understanding.
- To provide a high-level description of the functional processes onto which project partners or other third-parties can map their existing experience and components.

This methodology will allow partners to ensure that the project as a whole is equipped to deliver integrated results that fit in the proposed architecture. Furthermore, the definition of SOA4All architectural processes will provide a mechanism to communicate what SOA4All is about to others, both within or outside the project.

The need to define an *overall SOA4All process* integrating the different pieces of the project into an integrated functional view was identified at an early stage in the project. All of the different activities in the project will benefit from this overall integrated view because it will facilitate the definition of the interfaces between the different activities.

We do not believe that this SOA4All process can be represented as a list of chronological tasks. Therefore, the SOA4All process methodology**,** as we have chosen to call it, consists of many SOA4All processes or activities which are relevant to the SOA4All view of dealing with services.

On the other hand, this methodology has been applied already to collect the component descriptions and interactions collected in this deliverable, and to have a preliminary view of the functional processes.

7.2 Integration Methodology

In order to ease the process in which all these actors interact and perform their duties, the first task of the Integration is to provide the SOA4All Architecture Process in which all the actors can find their responsibilities and how they contribute to the final integration. The SOA4All Architectural Process Methodology comes to aid in this task, following a centralised approach also during the design and development phase of the project.

We follow a version of the Rational Unified Process (RUP), an iterative process model for building software applications. The ultimate artefacts (software components) produced by this process are to be integrated within the DSB. The main activities of this model are:

1. Specification: description of the user requirements for the components (done in each WP).
2. Design: technical and functional description of the final solution for the artefacts cited above (done in each WP). This design includes the definition of the interfaces and messages with other components plus the external methods (API) of each component.

3. Development: construction of the software and dependent resources for the designed artefacts (done in each WP). The development includes the realisation of the interfaces with the external world.
4. Testing: designing and executing of test suites that guarantees the adjustment of the developed software with the user requirements. This phase also includes the integration testing within the DSB and the interactions with other components.
5. Deployment: packing of the artefacts for simplifying their installation. The components should include the necessary guidelines and steps to be deployed within the DSB when required.

In this document we propose templates to describe the components and their interactions, which were in its initial form already applied to populate Section 5:

1. Component description (see Annex G).
2. Interface description (see Annex H).

7.3 Structure of the SOA4All Architecture Process

The meta model for defining the SOA4All Architecture Process is kept fairly simple as it is not the primary objective of the project to define a detailed methodology. The following diagram gives an overview of this meta model (see Figure 17). Following, we provide a brief description of each class of this logical model.

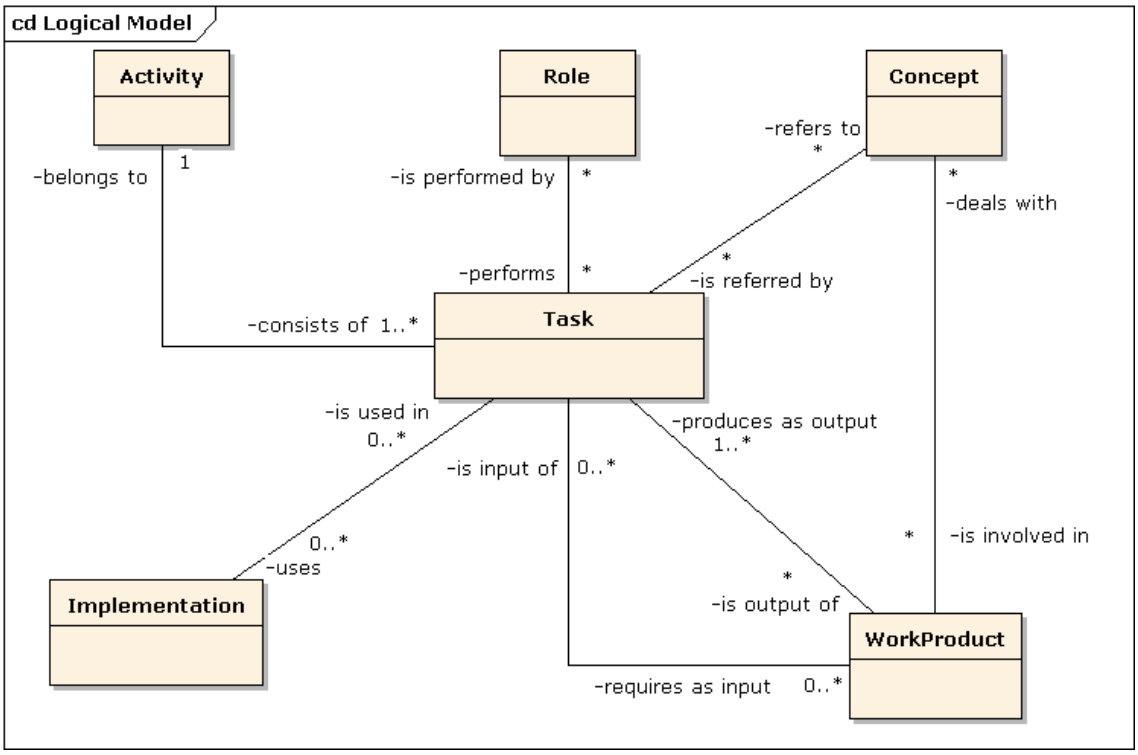


Figure 17 The SOA4All Process Meta Model

Activity: An activity is a collection of processes that have a common general objective or theme (e.g. Discovery of services). It consists of tasks representing the best practices in its domain. Activities are grouped in activity areas.

Task: A task is a process that serves the need of the activity it belongs to. A project task represents a controllable piece of work within a specific project schedule. Based on inputs, it

produces/amends work products and delivers these as output.

Implementation: An implementation is a method for carrying out a process, e.g. a way of performing some task or part of a task.

Tasks are often implementation-independent. In some cases, there will be several alternative or complementary implementations for performing a task. On the other hand, some implementations can be usefully applied to several different tasks.

Concept: The concepts explain the set of fundamental ideas that are used throughout the SOA4All project, which underpin the implementations and activities. The concepts are inter-related and, together, they form a coherent conceptual model.

Role: A role is a set of responsibilities and skills needed in the project.

Work product: A work product is a deliverable that is produced during the performance of one or more tasks.

- The activity explains *why* you are performing some task or implementation
- A task explains *what* to do
- A role defines *who* does it
- An implementation explains *how* to do something

The most important step to be done is the definition of the SOA4All Activities. Within SOA4All these activities can be clearly divided between design time (e.g. create semantic descriptions of services, generate composite services, etc.) and runtime activities (e.g. service discovery, execution, etc.). Currently the project is defining these activities. We propose to describe the activities following the guidelines expressed in 0.

7.4 Functional Processes

Functional processes are the activities defined in the previous section. The following functional processes are explained by means of the NEXOF-RA system requirements questions that were published in deliverable D7.3 Conceptual Architecture View. This further eases the process of feeding SOA4All results back to NEXOF-RA and facilitates the pin-pointing of solutions to requirements. The current architecture specification of SOA4All does not yet include the documentation of complete processes that require the integration of multiple platform services, but considers those rather in isolation. In the previous sections of Chapter 7, we presented a process methodology that will be applied to specify and implement such complete execution processes for the second release of the SOA4All architecture; those will be published and implemented in course of deliverables D1.4.2A and D1.4.2B.

In the following sections we shortly provide high-level rules and indications concerning the most relevant processes in a service-oriented architecture. These concern service-related aspects such as the creation, discovery, composition and invocation of services.

7.4.1 Service Creation and Execution

The main questions of this section are: “how can a service be created?” and “how can a service be executed?”. These two questions correspond to the NEXOF-RA requirement categories SC, respectively SE. SC addresses the methods for service creation, which are to a large extent not subject to SOA4All, while SE refers to the computation performed by the provider agent upon the reception of a client request.

Regarding SC, SOA4All does not address the creation and deployment of services, meaning delivering and deploying an executable piece of software. On the other hand, SC also addresses the semantic annotation process. This means that the delivery of MicroWSMO, WSMO-Lite and SAWSDL annotations is part of SC.

SOA4All then allows the generation of RESTful services annotations based on MicroWSMO. The SOA4All Studio offers the necessary user interface to allow the annotation process and usage of this type of services.

Regarding WSDL-based services, the crawler component is used in the first place in order to gather service descriptions. The SOA4All Studio offers a GUI to define the WSMO-Lite annotations using the semantic annotations standard SAWSDL to point to service semantics. The execution of WSDL-based services is done using the DSB features.

The execution of third-party services is done in the service provider side, while the description of the execution of SOA4All composite services can be found in Section 7.4.4.

7.4.2 Service Invocation

The NEXOF-RA requirement SI addresses the question of “how can a service be invoked?”, and hence concerns those mechanisms offered by the system to enable service invocation.

SOA4All addresses two types of services: WSDL-based and RESTful services as explained in Section 6.1. In the case of WSDL-based services, in SOA4All we distinguish between simple and composite services. Simple services are business services described in WSMO-Lite, which offer any number of ‘operations’. Within SOA4All these third-party services are exposed by the DSB and their semantic description stored in the Service Registry. The invocation of a service unit can be triggered directly from the user using the service consumption part of the SOA4All Studio.

7.4.3 Service Discovery

Service Discovery addresses the mechanisms of the system that allow a service provider to deliver service descriptions to clients and in particular the process of searching for and selecting the appropriate services. In NEXOF-RA this requirement category is named SD, and represents the question of “how can a service by discovered?”.

In SOA4All the discovery process is mainly driven by the service discovery component. This component enables users to find a service appropriate for their needs. The discovery process offers two different mechanisms.

First, there is the possibility of doing full text based discovery, which uses the web service descriptions (WSDLs) and related documents crawled by the crawler component and stored in the SOA4All repositories. This full text search allows users to search for services by entering keywords in a search engine fashion.

Second, the semantic discovery allows users to perform goal-based discovery. In this case a more structured goal (service classification, pre-conditions and effects) is entered by the user and the component finds the service that match the goal by using the reasoning facilities provided by WP3.

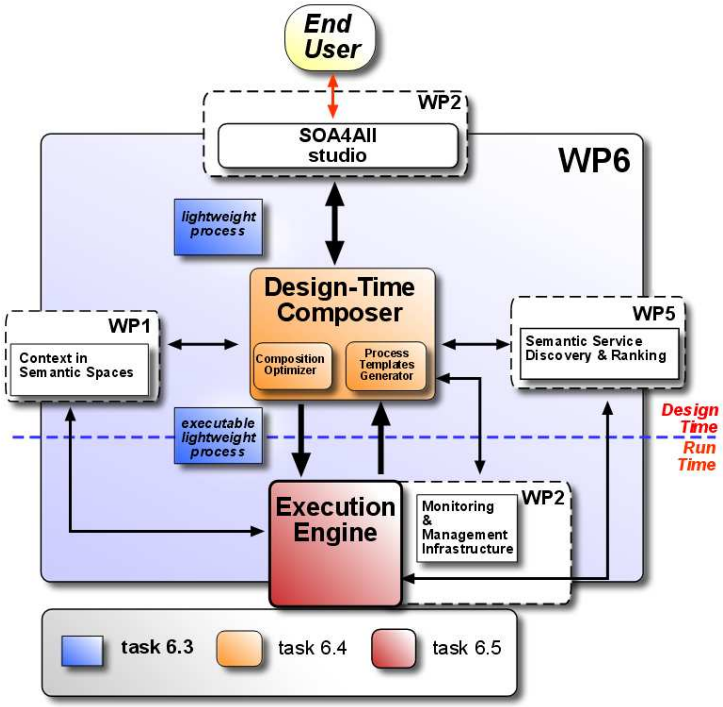
7.4.4 Service Composition (Processes)

Services are clearly related to business functionality, and hence it is a requirement that they can interact and cooperate to realize complex processes. The NEXOF-RA requirement PRO addresses the issue of service composition via the question of “how can a process be realized by composing services?”.

The SOA4All service composition has a clear division between the design time and runtime phases (cf. Figure 18): during *design time*, the service composition process allows users to specify their required composite services and processes (part of the SOA4All Studio) based on a graph-oriented lightweight process modelling language defined within WP6. To improve usability pre-designed and user-designed process templates are stored in the semantic service and template repository. Once created and stored, in order to be used this lightweight processes have to be translated in to more complex processes that can be interpreted by an execution engine in an effective fashion. In order to do that, the design-time composer

component allows a flexible and ad-hoc creation and adaptation of complex services at design time. The aforementioned lightweight processes are then transparently transformed in to optimized complex services orchestrations by using the optimiser component; or already existing complex services processes could be adapted to a specific use. These activities are heavily influenced by the context in which they will be carried out. The final outcome of the design time phase would be a set of complex processes descriptions described in terms of the lightweight language. Finally, regarding the *runtime* phase of service construction, WP6 offers an execution engine. It will execute complex processes that represent orchestration of services. This execution will be adaptive to environmental changes; and flexible enough to allow its context-dependent self-reconfiguration. This engine will consider also context during execution as well. The execution engine receives the processes described during design time. These process descriptions have to be translated to executable BPEL processes in which the different services will be described in WSMO-Lite. For every execution, the engine will interact during runtime with the discovery component and ranking and selection components in order to find the most appropriate service to execute. The invocation of the service would be triggered by the execution engine making use of the service invocation service explained before.

Figure 18: Service Construction Framework at a Glimpse



8. Implementation

The implementation of the SOA4All platform is subject to the second year of the project. The various components (SOA4All Platform Services) including their implementation will be further described in their respective deliverables. Table XXX provides a summary of all components and the expected data of individual implementation releases. Integration-related aspects of the implementation will be in deliverable D1.4.1B with expected release in month M18 (September 2009). Issues considered in this respect are aspects that are relevant to the project as a whole, rather than to individual components; e.g., scalability-driven discussions, error handling. Furthermore, deliverable D1.5.1 specifies testing infrastructures to evaluate the scalability of the SOA4All infrastructure. A mechanism is determined to (semi-) automatically generate suitable amounts of services with given default behaviour and critical QoS parameters. Further details can be found in D1.5.1 and the subsequent deliverable D1.5.2 that is planned for September 2009.

Table 3: Component Implementation Release Dates

Component	WP	Date of Implemenation Release	
SOA4All Distributed Service Bus	WP1	M18 / M30	(early prototype M12)
Semantic Spaces	WP1	M18 / M30	(early prototype M12)
Monitoring (DSB)	WP1	M18 / M30	
Deployment Facility	WP1	M18 / M30	(early prototype M12)
SOA4All Studio - Consumption Platform	WP2	M18 / M30	(early prototype M12)
SOA4All Studio - Provisioning Platform	WP2	M18 / M30	(early prototype M12)
SOA4All Studio - Analysis Platform	WP2	M18 / M30	(early prototype M12)
WSML Reasoning Framework	WP3	M18 / M24 / M30	(early prototype M12)
WSMO Data Grounding	WP3	M18	
Crawler	WP5	M12 / M18	
Service Registry	WP5	M18 / M30	(early prototype M12)
Discovery	WP5	M12 / M30	
Ranking and Selection	WP5	M18 / M30	
Design Time Composer	WP6	M18 (basic), M24 / M30	(early prototype M12)
Template Generator	WP6	M18 (basic), M24 / M30	
Composition Optimizer	WP6	M18 (basic), M24 / M30	
Execution Engine	WP6	M18 / M30	(early prototype M12)

We add a listing of existing core technologies and standards that will provide the building blocks for the SOA4All platform implementation:

PEtALS (petals.ow2.org)	Open-source, JBI compliant ESB
ProActive (proactive.ow2.org)	A parallel, distributed, multi-core computing platform
FraSCAti (frascati.ow2.org)	SCA engine integrated into PEtALS ESB
FDF (fdf.forge.inria.fr)	Framework for deploying distributed software systems
WSML	For describing Semantic Web services, goals and ontologies at the level of the reasoning framework (according to the object specification in Section 6).
MicroWSMO, WSMO-Lite, hREST	For annotating WSDL, resp. REST services
WS-Notification	For event-driven interfaces at the level of the DSB
AJAX	For programming Web applications

8.1 The SOA4All DSB Implementation Architecture

The SOA4All DSB integrates various underlying key technologies described into Section 4.1: OW2 PEtALS, OW2 ProActive, OW2 FraSCAti, and SOA4All Semantic Spaces. Figure 19

shows the overall implementation architecture of the SOA4All DSB and outlines the relationships between these technologies to integrate.

The SOA4All DSB is implemented with the Java programming language and runs on a set of distributed Java Runtime Environments (JRE). The ProActive library provides the distributed middleware runtime to deploy and execute the SOA4All DSB on large-scale distributed infrastructures like Internet and grids. ProActive is the reference implementation of the Grid Component Model (CGM) ETSI standard [3]. PEtALS and SOA4All Semantic Spaces are implemented with and run on ProActive to benefit from its features described in Section 4.1.2. PEtALS will use SOA4All Semantic Spaces as a large scale distributed infrastructure to transport interactions between PEtALS nodes as described in Section 4.1.3. The PEtALS CDK introduced in Section 4.1.1 provides common foundations to implement JBI components (both BC and SE). At least, the SOA4All DSB includes the following JBI components: the REST BC, the WS BC, the FraSCAti SE, and the Semantic Spaces BC. The REST and WS BC allow the SOA4All DSB to communicate with billions of external third-party services deployed over the world. The FraSCAti SE discussed in Section 4.1.1 provides the runtime support for executing SCA applications on top of the SOA4All DSB. The Semantic Spaces BC allows us to integrate Semantic Spaces in the JBI world. If required, other BC and SE can be included into the SOA4All DSB to respectively communicate with non REST or WS external third-party services or to execute some SOA infrastructure services like a BPEL engine and a message transformation engine.

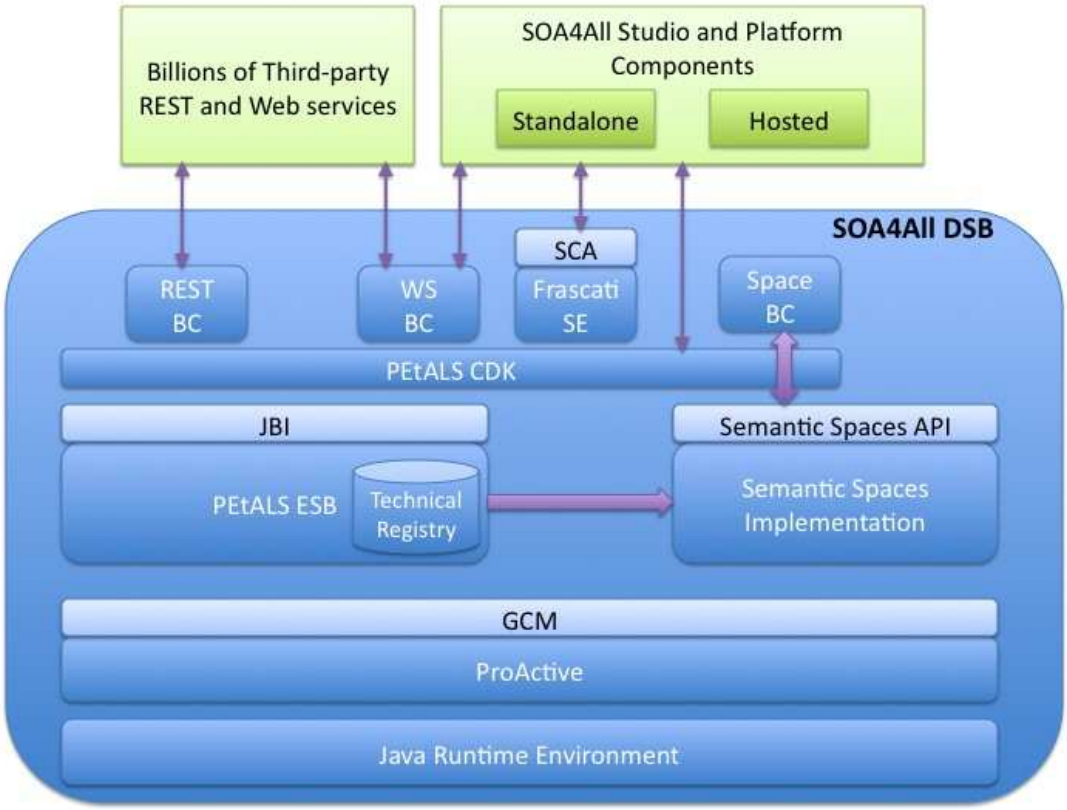


Figure 19: The SOA4All DSB Implementation Architecture

The SOA4All DSB provides three complementary approaches to integrate the SOA4All Studio and platform services, i.e., allowing them to communicate via the SOA4All DSB. Firstly, if a SOA4All platform service (or the studio) runs as a standalone program (i.e., not be hosted by the SOA4All DSB), then this program exposes its services to the SOA4All DSB and accesses other platform services via the WS BC. Secondly, if a SOA4All platform service would like to be fully deployed, hosted, and monitored by the SOA4All DSB, then this

service must be encapsulated into an SCA composite and can have access to the SOA4All Semantic Spaces as specified in Section 4.1.3 and Annex B. This approach is especially used in D6.4.1 where lightweight adaptable processes are transformed into SCA composites which are running on the FraSCAti SE. Finally, if a SOA4All platform service requires having a full fine-grain access to and control on JBI features (e.g., deployment, lifecycle management, monitoring, and communication patterns), then this service must be implemented as a JBI SE on top of the PEtALS CDK. This is typically the case for the WP6 Execution Engine.

9. Conclusion

In this document we have defined the first version of the SOA4All architecture. We have provided a high level description of the SOA4All platform by showing the relationship between the SOA4All core infrastructure services (the Distributed Service Bus and Semantic Spaces that are subject to WP1), the so-called SOA4All Platform Services (aka components) and the SOA4All Studio of WP2 respectively business services. A special attention was given to the core infrastructural services of SOA4All platform. The four artefacts that form the pillars of core infrastructural services, namely the SOA4All Distributed Service Bus, the deployment facility, the monitoring platform and the Semantic Spaces were presented in details. They provide the means for integration, communication and coordination of the other SOA4All components. Each component of the architecture is presented in details, including the interface of the component, the relationships and dependences with the other components as well as the control flow the component is involved. The dependencies between components have been represented by means of an interaction matrix. Data being exchanged and shared between various components of SOA4All architecture is described as communication objects. The communication objects have their definition aligned with the conceptual model of the OASIS Semantic Execution Environment TC (SEE), and in a more terminological sense with the SOA4All Glossary, which is a specialization of the larger NEXOF-RA Glossary. The common terminology and conceptual model defined as part of this deliverable are meant to foster understanding across all actors involved in the SOA4All development. We have specified as well a methodology that will ease the creation and implementation of SOA4All functional processes. This should be used as input for the next version of the architecture specification.

Architectural specifications are “living” documents. Further analysis and evolution of individual components on one hand and more precise architectural requirements on the other hand should and are usually reflected in updated versions of architecture specification. The architecture defined in this document is not final yet as we have to envision possible changes and adjustments that will surface during the further stages of the project. These changes will be applied to the architecture and will be worked into the second version (D1.4.2) of the SOA4All architecture.

References

[1] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, R. Quilici. Programming, Composing, Deploying for the Grid. Book chapter in Grid Computing: Software Environments and Tools, Jose C. Cunha and Omer F. Rana (Eds), Springer 2006.

[2] L. Baduel, F. Baude, D. Caromel. Asynchronous Typed Object Groups for Grid Programming. International Journal of Parallel Programming, Springer. 35(6), 573–614, Dec. 2007, http://www.springerlink.com/content/hj66703036502292/

[3] F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, C. Pérez. GCM:A Grid Extension to Fractal for Autonomous Distributed Components. Annals of Telecommunication – Annales des telecommunications 64(1), pages 5-24, Springer 2009.

[4] F. Baude, V. Legrand, V. Lestideau. Large Scale Service Deployment - Application to OSGi. 3rd Int. conf. on Autonomic and Autonomous Systems (ICAS 2007), June 2007, Athens.

[5] E. Bruneton, T. Coupaye, J.-B. Stefani. The Fractal Component Model. February 2004, http://fractal.objectweb.org/specification.

[6] E. Bruneton, T. Coupaye, M. Leclercq, V. Quema, J.-B. Stefani. The Fractal Component Model and its Support in Java. Software -- Practice and Experience, special issue on ``Experiences with Auto-adaptive and Reconfigurable Systems'', 1257-1284, 2006.

[7] D. Chappell. Introducing SCA. White paper, July 2007. Available at http://www.davidchappell. com/articles/Introducing_SCA.pdf.

[8] P. Corte, D. Desideri, V. Stricker, N. Tsouroulas, K. Mehner, M. Fisher, P. Bisson, R. Jimenez-Peris. Conceptual Architectural View. NEXOF-RA Deliverable D7.3, October 2008.

[9] A. Flissi, P. Merle. A Generic Deployment Framework for Grid Computing and Distributed Applications. In Proceedings of the 2nd International OTM Symposium on Grid computing, high-performAnce and Distributed Applications (GADA'06), Lecture Notes in Computer Science (LNCS), volume 4279, pages 1402 - 1411, Montpellier, France, November 2006. Springer-Verlag.

[10] A. Flissi, J. Dubus, N. Dolet, P. Merle. Deploying on the Grid with DeployWare. In Proceedings of the 8th International Symposium on Cluster Computing and the Grid (CCGRID'08), pages 177 - 184, Lyon, France, May 18-22 2008. IEEE.

[11] Java Community Process. Java(tm) Business Integration (JBI) 1.0 - Final Release. JSR 208, Sun Microsystems, Inc., August 2005.

[12] E. Mathias, F. Baude, and V. Cavé. A GCM-Based Runtime Support for Parallel Grid Applications. In Proceedings of the Workshop on Component-Based High Performance Computing (CBHPC'08) in conjunction with ACM SIGPLAN CompArch 2008.

[13] N. Medvidovic, R. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering, 26, issue 1:70–93, January 2000.

[14] P. Merle and all. SCA Platform Specifications - Version 1.0. ANR SCOrWare Project WP1 Deliverable. September 2007.

[15] Object Management Group. Unified Modeling Language: Superstructure. Available Specification, Version 2.1.2, OMG TC Document formal/2007-11-02, November 2007.

[16] Open SOA. SCA Service Component Architecture - Assembly Model Specification. SCA Version 1.0, Open SOA, March 2007.

[17] OSGi Alliance. Open Services Gateway Initiative Service Platform Specification. Version 4.1, May 2007.

[18] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, J.-B. Stefani. Extensibility and Reconfigurability in the FraSCAti SCA Platform. Submitted to the 12th International Symposium on Component Based Software Engineering (CBSE-2009), East Stroudsburg University, Pennsylvania, USA, 22-25 June 2009.

[19] Sun Microsystems Inc., Java Enterprise Edition 5 Specification. JSR-000244 2.6, November 2003.

[20] World Wide Web Consortium, W3C: Simple Object Access Protocol, SOAP, Version 1.2 Part 0: Primer, (2003). Web site: http://www.w3.org/TR/soap12-part0/).

[21] World Wide Web Consortium, W3C: WSDL: Web Services Description Language (WSDL) 1.1, (2001). Web site: http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[22] V. Zurczak, M. El Jai. SCA Support in PEtALS with Tinfi/FraSCAti. Presentation at the OW2 Quarterly Meeting, Grenoble, France, May 15th 2008.

[23] World Wide Web Consortium, W3C: Web Services Message Exchange Patterns, (2002). Web site: http://www.w3.org/2002/ws/cg/2/07/meps.html.

[24] World Wide Web Consortium, W3C: Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts, (2007). Web site: http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/.

[25] World Wide Web Consortium, W3C: Web Services Description Language (WSDL) Version 2.0: Additional MEPs, (2007). Web site: http://www.w3.org/TR/wsdl20-additional-meps/.

[26] OASIS Open. Web Services Base Notification (WS-BaseNotification). OASIS Open, Version 1.3. October, 2006.

[27] B. Schroeder. On-Line Monitoring: A Tutorial. IEEE Computer, 28, issue 6:72-78, June 1999.

[28] OASIS Open. Web Services Topics (WS-Topics). OASIS Open, Version 1.3. October, 2006.

[29] OASIS Open. Web Services Brokered Notification (WS-BrokeredNotification). OASIS Open, Version 1.3. July, 2006.

[30] B. van Dongen, W. van der Aalst. A Meta Model for Process Mining Data. In: Missikoff, M., Nicola, A.D. (eds.): EMOI-INTEROP, Vol. 160. CEUR-WS.org, 2005.

[31] M. Muhlen. Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems., Vol. 6. Logos, Berlin, 2004.

[32] C. Pedrinaci, J. Domingue, A. Alves de Medeiros. A Core Ontology for Business Process Analysis. In Proceeding of 5th European Semantic Web Conference, 2008.

[33] C. Pedrinaci, D. Lambert, B. Wetzstein, T. van Lessen, L. Cekov, M. Dimitrov. SENTINEL: A Semantic Business Process Monitoring Tool. In Proceeding of Ontology-supported Business Intelligence (OBI2008) at 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany, 2008.

[34] A. Alves de Medeiros, W. Van der Aalst, C. Pedrinaci. Semantic Process Mining Tools: Core Building Blocks. In Proceeding of 16th European Conference on Information Systems, Galway, Ireland, 2008.

[35] A. Alves de Medeiros, C. Pedrinaci, W. van der Aalst, J. Domingue, M. Song, A. Rozinat, B. Norton, L. Cabral. An Outlook on Semantic Business Process Mining and Monitoring. In Proceedings of International IFIP Workshop On Semantic Web \& Web Semantics (SWWS 2007), 2007.

[36] T.van Lessen, J. Nitzsche, M. Dimitrov, D.Karastoyanova, M. Konstantinov, L. Cekov: An Execution engine for BPEL4SWS. In Proceeding of 2nd Workshop on Business Oriented Aspects concerning Semantics and Methodologies in Service-oriented Computing (SeMSoc) in conjunction with ICSOC, 2007.

Annex A. An Integration Use Case with PEtALS

This annex illustrates a simple integration use case which shows the agility of a system built on top of the PEtALS ESB: the travel agency scenario. Figure 20 shows how this scenario can be designed as an SCA composite which exposes the Travel service to other applications, make references to three third-party services (Hotel, Weather, Email) and is composed of two components (TravelService implementing the orchestration of the three services and BookHotelWrapper an adapter between Book and Hotel interfaces). This example is typical of what sort of composite service may be designed and deployed through the SOA4All Studio.

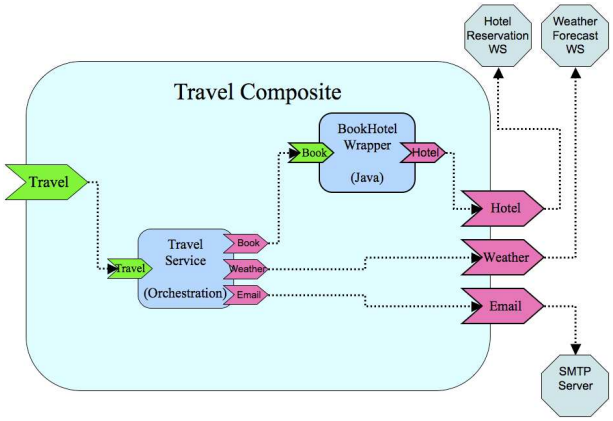


Figure 20: The Travel Agency Scenario as an SCA Composite.

Then as shown in Figure 21, the SCA support in PEtALS transforms this SCA composite into a set of service units running on top of JBI components. SCA components and bindings are mapped to JBI service units deployed on corresponding JBI service engines and binding components respectively.

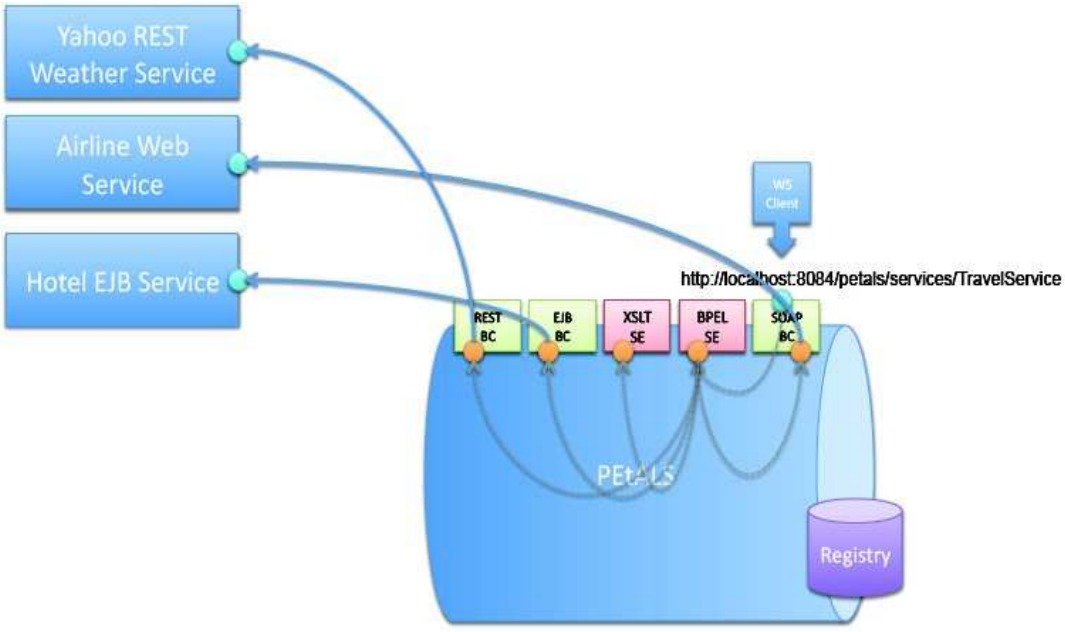


Figure 21: Mapping the Travel Agency Scenario to JBI Components.

1. The agency provides a single consumer service to book a travel from a start date to an end date and from a source city to a destination one. On the other side, the agency will consume one (or more) hotel booking service, one (or more) airline

booking service. Additionally, it will give some forecast details coming from another service.

2. All the business services will be orchestrated by a BPEL engine (or better by an engine of the lightweight orchestration language defined in WP6) and data adaptation between services will be handled by a transformation service.

One of the advantages is that on the orchestration point of view, all is seen as JBI services. Whereas standard service orchestration generally deal with web services bindings, the orchestration language engine JBI Service Engine can instead orchestrate JBI services and there is no restriction on the final service protocol since it is up to the binding component to handle the communication adaptation. One other advantage is that there is no problem if, for example, the external weather service no longer exists. The travel agency just needs to find a new weather service and to potentially add a data adaptation layer to provide the same interface and so JBI endpoint. Note that this is the same if the Airline service is no more a web service but an EJB one.

Annex B. SCA Semantic Space Binding XML Schema and Examples

Following is the specification of the XML Schema for the SCA Semantic Space Binding. This XML Schema extends the XML-based SCA ADL.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.SOA4All.eu/xmlns/sca/1.0"
    xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
    elementFormDefault="qualified">
  <include schemaLocation="sca-core.xsd"/>
  <element name="binding.space" type="sca:SpaceBinding"
      substitutionGroup="sca:binding"/>
  <complexType name="SpaceBinding">
    <complexContent>
      <extension base="sca:Binding">
        <anyAttribute namespace="##any" processContents="lax"/>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

This XML Schema defines a new XML namespace specific to SOA4All (`http://www.SOA4All.eu/xmlns/sca/1.0`), includes the SCA core XML Schema (`sca-core.xsd`), and defines a new element called `binding.space` of type `sca:SpaceBinding` and part of the substitution group `sca:binding` defined in the SCA core. The new complex type `SpaceBinding` is an extension of the `sca:Binding` complex type defined in the SCA core, then it inherits from all the attributes of `sca:Binding` and especially the attribute `uri` used in the two following SCA composite examples. Then a SOA4All SCA Semantic Space Binding can be used to bind any SCA service and reference to a Semantic Space identified by an URI.

The following illustrates how the SCA Provider Composite from Figure 8 can be defined to use the SCA Semantic Space Binding:

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.SOA4All.eu/xmlns/sca/1.0"
           name="ProviderComposite">
  <service name="service" promote="ProviderComponent/service">
    <interface.java interface="Service"/>
    <binding.space uri="http://www.SOA4All.eu/Sample-SCA-Semantic-Space"/>
  </service>
  <component name="ProviderComponent">
    <service name="service">
      <interface.java interface="Service"/>
    </service>
    <implementation.java class="ProviderComponentImpl" />
  </component>
</composite>
```

This SCA provider composite declares one service and an enclosed component implemented by a Java class. The service promotes the service of the enclosed component, is defined by the `Service` Java interface, and is bound to a Semantic Space identified by a Uniform Resource Identifier (URI).

The following illustrates how the SCA Consumer Composite from Figure 8 can be defined to use the SCA Semantic Space Binding:

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns=http://www.SOA4All.eu/xmlns/sca/1.0
           name="ConsumerComposite">
  <component name="ConsumerComponent">
    <reference name="service">
      <interface.java interface="Service"/>
    </reference>
    <implementation.java class="ConsumerComponentImpl" />
  </component>
  <reference name="service" promote="ConsumerComponent/service">
    <interface.java interface="Service"/>
    <binding.space uri="http://www.SOA4All.eu/Sample-SCA-Semantic-Space"/>
  </reference>
</composite>
```

This SCA consumer composite declares a reference and an enclosed component implemented by a Java class. The reference is wired to the enclosed component (`promote`), is defined by the `Service` Java interface, and is bound to the same Semantic Space than the one of the SCA provider composite. However the consumer does not know the exact service provider endpoint that it will invoke, it just knows the existence of the Semantic Space where some providers are connected. Then two successive consumer requests can be delivered to two different providers, i.e., dealing with request load balancing and provider fault tolerance in a transparent way from the consumer's point of view.

Annex C. SOA4All Software and Services to Deploy

This annex lists all the software and services involved in the SOA4All distributed service computing environment which will be deployed automatically thank to the SOA4All Deployment Facility described in Section 4.2:

- SOA4All Runtimes composed of:
  - Java Runtime Environments (JRE) to run PEtALS, ProActive, Web servers, etc.
  - DSB nodes composed of PEtALS, Semantic Spaces and ProActive software
  - All required JBI components deployed on top of PEtALS nodes:
    - FraSCAti SCA Service Engines
    - Web service and REST service Binding Components
    - Semantic Space Binding Components
    - Other required Service Engines or Binding Components; e.g., a BPEL engine, a transformation engine, or an EJB binding.
- SOA4All Studio instances composed of:
  - Web servers as Apache Tomcat
  - WARs packaging Studio components
- All SOA4All platform components coming from WP3 to WP6
  - Packaged as JBI components and/or SCA composites
- Use case services coming from WP7 to WP9
  - Packaged as JBI components and/or SCA composites
- Bindings to existing external third-party services
  - JBI service units to configure JBI Web Service and REST binding components
- Any software required to execute previous software and services

The SOA4All Deployment Facility will not be used to deploy external third-party services as they are deployed by their suppliers and not by SOA4All administrators.

Annex D. A SOA4All Deployment Description

This annex illustrates a simple SOA4All deployment description (see Figure 22), which defines that one JBI component (`myJbiService`) must run on a PEtALS server (`petals-SOA4All`) deployed on an Internet node (`SOA4All`) where a JRE must be (`java`). Each service/software/host contains its own set of associated properties like the location of the service/software `archive` to upload and install, the `petals` server where the JBI service must be deployed, the `home` directory and `host` where service/software must be installed, the `hostname`, the `user` account, the file `transfer` and remote access `protocols`, the `shell` of a remote node, and some of its shared `software`.

```
sample {
  myJbiService = PEtALS.JBI {
    archive = PEtALS.ARCHIVE(myService.zip);
    petals  = /petals-SOA4All;
  }
  petals-SOA4All = PEtALS.SERVER {
    archive = PEtALS.ARCHIVE(http://petals.ow2.org/petals.zip);
    home    = PEtALS.HOME(/software/petals);
    host    = /SOA4All;
  }
  SOA4All = INTERNET.HOST {
    hostname = INTERNET.HOSTNAME(www.SOA4All.eu);
    user     = INTERNET.USER(root,…password…,…key…);
    transfer = TRANSFER.SCP;
    protocol = PROTOCOL.OpenSSH;
    shell    = SHELL.SH;
    software {
      java = JAVA.JRE {
        archive = JAVA.ARCHIVE(http://www.java.sun.com/jdk.exe);
        home    = JAVA.HOME(/software/java);
      }
    }
  }
}
```

Figure 22: An Example of SOA4All Deployment Description

Annex E. Event Ontology (EVO)

Research in Workflow Management Systems and Business Process Management has focussed on capturing information about the execution of processes in ways that can enable the application of diverse analysis techniques, such as Business Activity Monitoring, Process Mining, Reverse Business Engineering, etc. These techniques use as starting points logs generated by the underlying systems which typically come in a plethora of diverse formats coming from heterogeneous systems which need to be aligned and merged before they can be analysed. In the light of this situation, common formats have been proposed as a solution to overcome this problem, e.g., MXML [30] or the Audit Trail Format by the Workflow Management Coalition (WFMC, [31]). Although these formats have proven their benefits, they are supported by technologies that are not suitable for reasoning and therefore limit unnecessarily the level of automation that can be supported. In order to overcome this, we previously defined in the context of the project SUPER (IST-026850), the Core Ontology for Business pRocess Analysis (COBRA), and a reference Events Ontology (EVO) that provides a set of definitions suitable for capturing monitoring information from a large variety of systems [32].

EVO is based on the previously mentioned formats since they provide general purpose solutions that have shown to be suitable to capture logs generated by a plethora of systems. EVO is centred on a state model that accounts for the status of processes and activities, as shown in Figure 23. The figure shows the different states and possible transitions contemplated for both Process Instances and Activity Instances which we believe are self-explaining. The dark dot represents the initial state, arrows represent transitions, the smaller boxes depict states, and bigger boxes encapsulate (conceptual) families of states. The state model has been captured ontologically, an enhanced with additional relations. For instance it is possible to determine whether an Activity Instance has been allocated--*isAllocated*--which is true for those that are either in state Running, Suspended, or Assigned. EVO therefore supports capturing the lifecycle of the execution of processes and their internal activities and links it to higher-level conceptual models for supporting further analysis techniques using COBRA as the core conceptualisation [32] - [35].

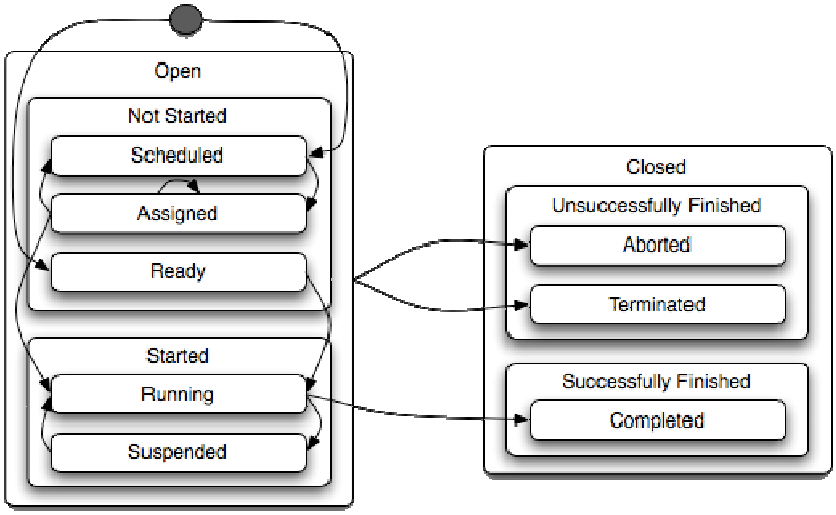


Figure 23: State Model Followed by EVO, see [32].

Annex F. Activity Description Form

Name: Activity name

Description: Activity description

Components involved: Components involved in this description

Sequence of tasks involved in this activity (UC): it implies a list of method invocations (it should be very useful a sequence UML diagram) to the Web service or any valid interface for the DSB provided by the component to accomplish this activity (UC) as a sequence of tasks. The description of the interfaces will be provided also by the component owner. This point is not planned for a functional analysis of requirements on the specification stage but it should be completed afterwards, when the technical design of the component is performed.

UML Diagrams: the whole set of activities should be depicted into a set of UML UC diagrams that shows the dependencies, generalizations, etc. between the different activities.

Annex G.Component Description

Name: Name of the component

Description: A short functional description of the component.

Inputs: A summarized list of entries that this component accepts to performs the operations it provides. This list can include: user interactions, incoming messages (commands), incoming documents, etc.

Outputs: A summarized list of outcomes produced by the invocation of the operations that the component provides. This list can include: outgoing documents, processed data, raw data, etc.

Interfaces exposed: A summarized list of interfaces exposed by the component that acts as an entry points. These interfaces include all the operations (messages) that the component accepts to be processed.

Interaction with internal components: A summarized list of interactions with components of the same activity. This point only includes those interactions initiated from this component. Interactions from other components with this one are included on those component description forms. This way aims to avoid describing the same interactions twice.

Interaction with external components: A summarized list of interactions with components of a distinct activity. This point only includes those interactions initiated from this component. Interactions from other components with this one are included on those component description forms. As above point, this agreement aims to avoid describing the same interactions twice.

Annex H. Interface Description

It consists on a full description of interfaces for those components that are going to be integrated in the DSB and the interactions with other components. Concretely, it consists on:

- The description of all the interfaces that expose the business functionality for a component, to be used by users through the DSB.
- The description of the interaction interfaces between different components.

For each method of the interface one record card should be filled as shown below.

Interface name		Name of the interface to which the operation belongs to
Operation name		
Component		Component name this operation belongs to.
Operation Description		Textual description in natural language of the operation, including a functional depiction.
Operation signature		UML description of the signature of the method, including return type, operation name and a list of its formal arguments including the name and type. See below on section “Operation signature description” for more details.
Pre condition		Description of the conditions to be satisfied before the execution of this operation
Post condition		Description of the effects of the execution of this operation on the environment or upon the parameters of the operation after its execution.
Complex Types	**Name**	Textual description of the complex type used in the operation signature.
Additional Info		Extra information describing some relevant aspects of the operation not included into the above fields.