Project Number:	**215219**
Project Acronym:	**SOA4All**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D1.4.2A Final SOA4All Reference Architecture Specification

Activity N:	Activity 1: Fundamentals and Integration Activity
Work Package:	WP1: SOA4All Runtime
Due Date:	28/02/2010
Submission Date:	28/02/2010
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	UIBK
Revision:	1.0
Author(s):	Reto Krummenacher (UIBK), Jean-Pierre Lorre (EBM), Christophe Hamerling (EBM), Alistair Duke (BT), Matteo Villa (TXT), Francoise Baude (INRIA), Elton Mathias (INRIA), Virginie Legrand (INRIA), Cristian Ruz, Dong Liu (OU), Carlos Pedrinaci (OU), Tomas Pariente Lobo (ATOS), Marin Dimitrov (ONTO), Philippe Merle (INRIA)
Reviewer(s):	Gianluca Ripa (CEFRIEL), Sven Abels (TIE)

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	2010-01-28	Initial TOC	Reto Krummenacher (UIBK)
0.2	2010-02-08	Update TOC, Section 3.2	Reto Krummenacher, Christophe Hamerling (EBM)
0.3	2010-02-12	Section 3.1	Francoise Baude (INRIA), Elton Mathias (INRIA)
		Section 3.3	Dong Liu (OU)
		Interface specification in Section 4.1	Florian Fischer (UIBK)
		Section 5.1	Alistair Duke (BT), Matteo Villa (TXT)
		Section 5.2	Christophe Hamerling (EBM), Francoise Baude (INRIA)
0.4	2010-02-15	Update Section 3.2	Christophe Hamerling
		Update Section 3.3	Dong Liu
		Fixing References	Reto Krummenacher
0.5	2010-02-16	First Draft Section 4.3	Tomas Pariente Lobo (ATOS)
0.6	2010-02-19	Update Section 4.3	Tomas Pariente Lobo
		Update Section 5.2 with configurations	Christophe Hamerling
0.7	2010-02-19	Final Draft (Internal Review)	Reto Krummenacher
0.8	2010-02-25	Incorporation of review comments	Reto Krummenacher, Matteo Villa, Alistair Duke, Tomas Pariente Lobo, Christophe Hamerling
1.0	2010-02-26	Final release for submission	Reto Krummenacher

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

Acronym	Definition
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
BC	Binding Component
BPEL	Business Process Execution Language
BT	British Telecom
CDK	Component Development Kit
COP	Constraint Optimization Problem
CSP	Constraint Satisfaction Problem
D	Deliverable
DSB	Distributed Service Bus
EJB	Enterprise Java Beans
ESB	Enterprise Service Bus
FP7	The 7th Framework Program
GCM	Grid Component Model
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
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
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
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
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

The goal of deliverable D1.4.2A is to provide an extended and more precise definition of the SOA4All architecture and in particular its components. Firstly, it provides an updated specification of the core infrastructural service of SOA4All, the Distributed Service Bus, and how the bus provides the backbone for federations, also termed Service Parks. Moreover, the deliverable yields an updated bus specification in terms of support for both WS-* stack services and RESTful services. In particular the support for RESTful services is new and emphasized as they gained momentum in the SOA4All roadmap; services that are resources on the Web strengthen the ideas of SOA4All. Secondly, the deliverable provides an updated list of the platform services and their interfaces, and presents a first fully integrated example of the SOA4All Global Service Delivery Platform that showcases the core functional processes for service location and service construction.

While the value of the various platform services is discussed in detail in the respective deliverables and working groups, the core infrastructural services such as the bus, the semantic spaces and the monitoring, as well as deployment functionalities are subject to this deliverable. Exploitation possibilities and business cases are for platform services are hence not contained in this document. However, this deliverable covers some fundamental business-oriented issues to give a first impression of the utility and usability of the bus infrastructure. Different application scenarios require different service bus configurations, and might lead to different business motivations and models. Matching infrastructure requirements to bus configurations is important for keeping the infrastructural backbone simple but effective. This work, also not mainly technical, argues for the chosen technical approach and is elaborated in the scope of the exploitation work package of the SOA4All project.

1. Introduction

The Web service technology stack offers a series of languages, protocols and techniques for making software functionality accessible as remote components, independent of particular programming languages and platform implementations. Significant work was done in specifying architectures, middleware, languages, communication protocols and process execution engines that can support the creation of complex distributed systems by seamlessly coordinating Web services. Service-Oriented Architectures (SOAs) foster the development of such distributed and loosely-coupled solutions whereby service providers advertise the services they offer, and solution providers and software developers access the service repositories to search for suitable services to invoke for the given purpose or to build and execute processes.

Within the SOA4All project, the core ideas of SOA are re-thought with the aim of making services ubiquitous on the Web. The chosen approach is to combine the principles which underpin the Web, Web 2.0, semantics, context and SOA and to derive an architecture based on these principles. In particular, from the Web we take openness, decentralization, and the fact that communication is driven by a ‘persistent publish and read’ paradigm rather than by messaging. In SOA4All, Semantic Web languages are leveraged to increase the automation of various common tasks during the life-cycle of services, such as their discovery and composition. From Web2.0 we take the value of easy-to-use interfaces and of social networks. Finally, automated context adaptation capabilities are embedded within the architecture in order to support the use of services in unforeseen contexts. In provisioning a Web were services exist in billions, we argue that the SOA4All architecture provides a Web-based example of a global service delivery platform. In particular, by empowering Web services as resources on the Web, SOA4All yields the fundamental building blocks for the creation of new business opportunities in form of open and loosely-coupled service economies.

A central element of such SOA-based service economies on the Web is the communication and coordination backbone that we term SOA4All Distributed Service Bus. As middleware for a Web of Service, the bus must ensure openness, as anybody should be enabled to contribute and use it. This means that the bus cannot be centrally controlled. Decentralization is also essential in regards to scalability. Consequently, the bus must cope with dynamicity, as openness and decentralization causes an element of chaos and messiness, and ensure interoperability to overcome data, service and platform heterogeneities. Last but not least, in particular in the context of loosely-coupled service economies and service parks, as we discuss in this deliverable, there is a need for enabling n:m interactions, as providers of services become consumers, and vice versa – this new role is generally referred to as prosumers [1].

The bus thus provides the core infrastructure services that are necessary to integrate various distributed platform services and business services. In deliverable D1.4.1A in month M12 [2], a first version of the SOA4All architecture was specified. It established and defined an integrated technical plan for the SOA4All Runtime and outlined the dependencies between platform services and the data/objects being exchanged and shared. This work set the ground for the development and integration activities of the SOA4All project. A first round of implementation cumulated in the software deliverable D1.4.1B [3], and further refinements lead to the content of this document.

1.1 Purpose and Scope

The goal of this deliverable is to provide an extended and more precise definition of the SOA4All architecture and its components. As for D1.4.1A the deliverable’s scope is twofold. First it provides an updated specification of the SOA4All Distributed Service Bus and how the bus serves as backbone for the creation of federations, also termed Service Parks. Furthermore, the updated bus implementation will include support for both WS-* stack services based on WSDL and SOAP, and RESTful services as well as Web APIs. In particular RESTful services gain more importance in SOA4All, as the idea of services as resources on the Web matches the core ideas of the SOA4All project even more. Parts of this deliverable are thus dedicated to the specification of interfaces for integrating the different types of external services; non-bus services. These services are on the one hand the project-internal platform services that constitute the service delivery platform, and on the other the third-party business services.

Additionally, the deliverable provides an updated list of the platform services and their interfaces, and presents a first fully integrated example of the SOA4All (Web) service delivery platform that showcases the core functional processes for service location and service construction by a practical approach.

At last, the deliverable covers more business-oriented issues of the SOA4All Runtime. Different application scenarios require different service bus configurations, and might lead to different business motivations and models. There is thus work dedicated to the presentation of different usage scenarios from an infrastructure point of view, and how different runtime configurations allow for optimized infrastructure deployments that do not require heavy-weight service bus installations for all scenarios. Matching infrastructure requirements to bus configurations is important in keeping the infrastructural backbone as simple as possible.

1.2 Structure of the document

In order to fulfil its purpose and to cover all aspects of the scope, the deliverable is structured in five sections. After this first introductory section, Section 2 will recapitulate the overall architecture of the SOA4All Runtime and will provide a short update to the conceptual architecture given in deliverable D1.4.1A. In Section 3, we discuss the updates to the Distributed Service Bus and the federation aspects of the middleware. Moreover, technical details are given about the support for external services, both WS-* stack services and RESTful services. After the section on federations of distributed service buses, Section 4 focuses on the various platform services. They are categorized into service location, service construction and support services such as reasoning or grounding. In the same section, there is a consolidated practical example that shows how the resulting service delivery platform translates a user goal into a service composition, including the discovery of atomic services and the deployment of the resulting process as Web service. Furthermore, there are more detailed specification of functional processes given. Section 5 finally covers the more business-oriented aspects of this deliverable, before concluding the document with Section 6. Section 5 discusses different usage scenarios, business models and corresponding configurations of the service bus infrastructure.

2. Recapitulation of the Conceptual Architecture

A Global Service Delivery Platform (GSDP) is an open platform through which domain independent services can be used to build problem-specific service solutions. SOA4All establishes a service delivery platform that is targeting Web services (traditional WS-* stack-based and RESTful services, as well as Web APIs). Future implementations of a GSDP will have to consider other exposable functionalities too, such as mobile services or sensors networks in order to fully enable the 'Everything as a Service' paradigm.

In this section we shortly recapitulate the SOA4All conceptual architecture.

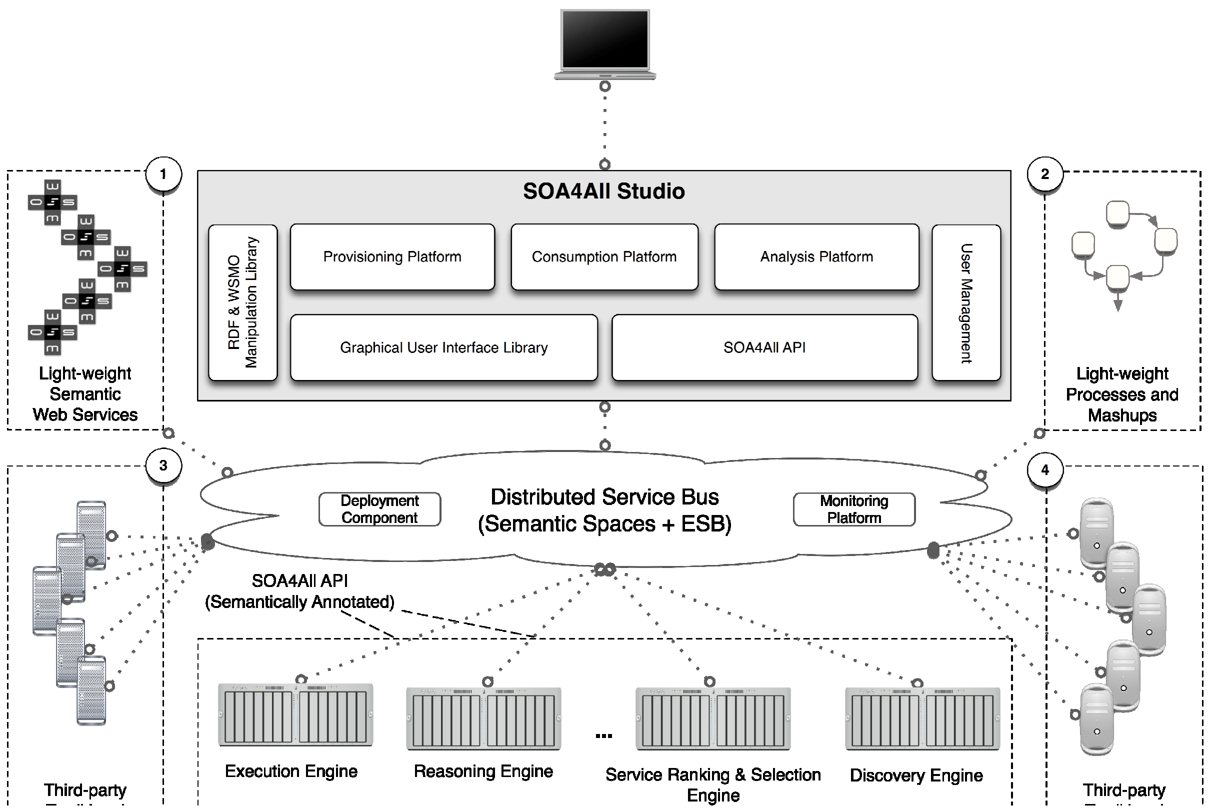
SOA4All Studio
RDF & WSMO Manipulation Library
Provisioning Platform
Consumption Platform
Analysis Platform
User Management
Graphical User Interface Library
SOA4All API
1
2
3
4
Light-weight Semantic Web Services
Light-weight Processes and Mashups
Deployment Component
Distributed Service Bus (Semantic Spaces + ESB)
Monitoring Platform
SOA4All API (Semantically Annotated)
Execution Engine
Reasoning Engine
...
Service Ranking & Selection Engine
Discovery Engine
Third-party
Third-party

Figure 1: SOA4All architecture.

Distributed Service Bus

The Distributed Service Bus enables Web-style communication and collaboration via semantic spaces and service bus technology, and yields the core runtime infrastructure. The DSB augments enterprise service bus technology with distributed service registries, the layering of the service bus on top of established Internet-enabled middleware, and the enhancement of the communication and coordination protocols by means of semantic spaces. Spaces are seen to significantly increase the scalability of the bus in terms of interaction between distributed and autonomous services [4]. Detailed description of the DSB and semantic spaces are given in various deliverables [2][3][5][6]. Several extension to this previously released specifications and prototypes are subject to this deliverable, at least in what concerns the SOA4All Runtime and hence the service bus. Most recent results about the semantic spaces are given in [7].

SOA4All Studio

The SOA4All Studio is a Web-based user front-end that consists of three components which offer service provisioning at design time, consumption and analysis of services at runtime, respectively:

The Provisioning Platform has two main purposes: i) tools to semantically annotate services; and ii) a process editor that allows users to create, share, and annotate executable process models based on a light-weight process modeling language. Service annotations are based on WSMO-Lite [8], a minimal extension to SA-WSDL [9] that empowers the creation of lightweight semantic service descriptions in RDFS. In parallel, MicroWSMO [10] is used to annotate services that are not described in WSDL, such as RESTful services or Web APIs. MicroWSMO is a microformat-based language around the constructs known from WSMO-Lite, however, adapted to support the annotation of HTML-based descriptions, as they are usually available for this type of software exposures. Finally, SOA4All provides a minimal service model in RDFS that yields an overarching conceptual model able to capture the semantics for both Web services and Web APIs, thus allowing both kinds of services to be treated homogeneously within SOA4All.

The Consumption Platform is the gateway for service consumers. It allows users to formalize goals. A goal is a formal specification of an objective and as such yields an implicit specification of the services that need to be executed. User objectives are transformed into processes that are compositions of service descriptions and so-called service templates together with control and data flow information and potentially further constraints on the services and their execution. Service templates define process-internal activities instead of concrete services whenever flexibility in service selection is desired. At runtime, service templates are resolved to specific services that are selected on the basis of conditions and informed by contextual knowledge which may include monitoring data, user location or other aspects that affect the appropriateness of a service endpoint.

The Analysis Platform collects and processes monitoring events from the service bus, extracts and produces meaningful information out of it and displays the results to users. Monitoring events come from data collectors that perform basic aggregation from distributed sources in the service delivery platform. Data collectors are installed at the level of the bus and the execution engine, and are installed to cover all aspects of monitoring necessary in the context of SOA4All: monitoring of process executions, of service end-points that are invoked through the service bus, and finally of the infrastructure itself. While users can select particular services to be monitored in terms of quality of service attributes, simpler and less comprehensive data can also be collected for all other services that are empowered through the bus, but that are not explicitly monitored upon user request; e.g., the moving average for response time.

Platform Services

Platform services provide the minimally necessary functionality of a service delivery platform, such as service discovery, ranking and selection, composition and invocation. These components are offered as Web services via the service bus and are consumable in the same manner as any other published business service. Although distinct in their purpose, they have in common that they operate with semantic descriptions of services, service templates and processes rather than with the syntactical representation of those, as it is traditionally the case in service-oriented infrastructures. The SOA4All platform services, shown at the bottom of Figure 1, are detailed in Section 3.

Business (Web) Services and Processes

Figure 1 was discussed so far with respect to the central components of the SOA4All service delivery platform – the ensemble of DSB, SOA4All Studio and platform services. Jointly, they deliver a fully Web-based service experience: global service delivery at the level of the bus, Web-style service access via studio, and automated service processing and management via platform services. Moving to the corners of Figure 1, we enter the domain of semantic service descriptions and processes: (1) represents the semantic services descriptions, either in form of annotated RESTful services (3) or WSDL endpoints (4); (1) thus represents the so-called Semantic Web services. The semantic descriptions are used for reasoning with service capabilities (functionality), interfaces and non-functional properties, as well as contextual data.

In the right top corner of Figure 1, (2) represents processes and mash-ups. Processes are orderings of Semantic Web services, service templates with associated constraints, data and control flow. A mash-up is a data-centric model of a composition that is almost entirely executable by coordinated access to data in a semantic space. Although being comparably simple, mash-ups provide a promising approach to Web-style service computing, a pre-requisite for light-weight service economies.

3. Federation of Distributed Service Buses

3.1 Recapitulation of Distributed Service Bus v1.0 and Updates

The primary goal of the SOA4All DSB is to ensure that the SOA4All framework scales to the dimensions of the Web, by enabling appropriate distribution techniques that evolve the traditional ESB towards a fully Distributed Service Bus, without altering the communication and interaction patterns of the ESB core. An ESB by itself, even if distributed (a DSB), is not a suitable solution for a service delivery platform resulting of the federation of the involved partners service infrastructures, e.g., their service buses. It mainly lacks inter-connection mechanisms and a global and shared store of meta-data about federated services.

In order to come up with a feasible and practical solution to the interconnection issue, we proposed an external Grid Component Model (GCM)/ProActive component-based platform to interconnect PEtALS DSBs [3][11], despite of their location and interconnectivity restrictions, only supposing an entry point on the administrative domain each DSB is deployed on. Current work consists in refining the initial solution to make the federation more loosely coupled yet introducing the needed important features: ability to route inter-DSB messages, corresponding to service invocation, service replies, or technical registry operations; ability to map at the federation level any particular relationships that may have been agreed between the partners' service delivery infrastructures.

Next sections present the new DSB Federation Architecture and the associated tools that enable the deployment of the federation, easy integration of PEtALS DSBs and management of the federation architecture to express partnerships among service providers.

3.1.1 DSB Federation Architecture

The DSB Federation Architecture is a standalone middleware whose main aim is to route messages (so we name it a routing middleware below). This middleware itself is an application programmed using software components. This enables to (re-)compose the routing middleware even at run-time. As a consequence, the composition will always reflect the effective federation organization. Each DSB will be represented in the federation by an entity we name "Federation Router", which is implemented by a GCM/ProActive software component. So, concretely, the routing middleware corresponds to an adequate composition of Federation Routers. Relationships between DSBs, to express alliance relations between them, are concretized by GCM bindings between the corresponding Federation Router components. Only partnered DSBs will be allowed to communicate directly. If no binding exists between two DSBs, even if they are part of the federation, this prevents them to directly interact. However, if those two DSBs are part of the same federation, this is because, they are indirect partners, e.g. the first one has agreed to partnership with a third one, this third one having itself partnered with the second one. Even if not every DSB pair of the federation has directly partnered, this does not prevent all the DSBs to be part of the federation, thus forming a service ecosystem that is also known as a service park [12].

Figure 2 shows the correspondence between multiple PEtALS DSBs, and resulting multi-domain and multi-DSBs federated architecture. In the context of a given DSB, multiple DSB containers are still capable of communicating using the standard PEtALS transporter mechanism (e.g. as in Domain C which includes two containers in this simple example). When communication happens between different DSBs, the messages are sent to a new sort of PEtALS transporter we defined (FederationTransporter, whose updated version is detailed in next subsection), which is responsible for delegating the message sending to the routing middleware. The routing middleware as we explained is basically composed of a router component (GCMRouter) per domain, installed on the gateway machine of the administrative domain the DSB is deployed onto.

Figure 2 also illustrates the idea that the middleware is completely hidden to the end-user. End-users interact with a virtually flat and unique infrastructure embodied by their preferred

DSB, without having to worry about effective locations of services involved in their service oriented applications or service orchestrations.

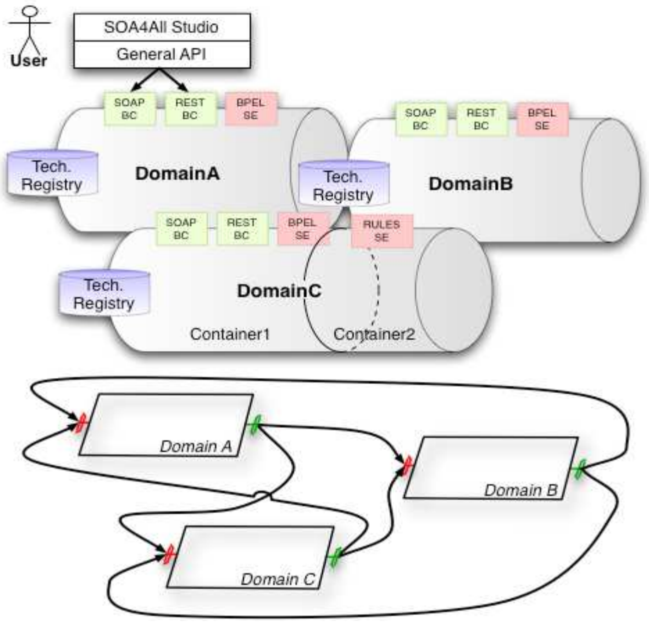

Figure 2: Multi-DSB federated architecture.

3.1.2 Integrating PEtALS DSBs and the DSB Federation

As previously described, the federation is achieved by a completely separated application that runs independently of the PEtALS DSBs. In order to integrate a PEtALS DSB to a federation, we included in PEtALS a new message transporter, called FederationTransporter that is responsible for the interface between PEtALS containers and the federation.

The FederationTransporter behaves at the same time as a server that receives messages from the federation, and as a client that sends message to DSBs running in different domains composing the federation:

- Client side: the client side of the FederationTransporter consists in keeping a remote reference of a federation router component and sending messages which must be routed to other domains;
- Server side: the server side of the FederationTransporter implements an interface named FederationListener that we defined for federation purposes. The aim of the FederationListener is to receive the callbacks automatically triggered by the Federation Router, when a message targeting this DSB arrives into this Federation Router.

Besides the definition of the FederationTransporter by itself, the JBI-Messaging component of PEtALS was extended so to use the FederationTransporter. The JBI-Messaging component seemingly decides the destinations i.e. if the message must go to a local JBI component or to a JBI component located in another container or to the DSB Federation in case the destination is a container running on a different domain. At any moment, a PEtALS DSB (i.e. a service provider) can decide to leave the federation. The federation, nonetheless stills exist and can be contacted again later.

At the configuration level, the integration of the FederationTransporter, and consequently the federation access, only requires the definition of the federation public address in the PEtALS topology.xml file. If this attribute is not defined we consider that the DSB will not be federated and therefore the FederationTransporter is disabled.

3.1.3 Associated Tools

The deployment over heterogeneous resources of the routing middleware supporting the federation, imposes a few activities that are hard to be handled by services providers willing to be prosumers of services deployed on the federation, mainly because they require the technical knowledge to configure the environment including knowing the technologies used to develop the routing middleware and the underlying environment (communication protocols, network organization and restrictions like firewalls and NAT).

In order to simplify the usage of the federation, we are working on two tools to help in: (i) the deployment of the federation routing middleware and (ii) the management of it.

The deployment of one partner of the federation is based upon the ProActive deployment framework. This framework requires the definition of XML-based descriptors indicating the network protocols and port numbers to use, how to acquire and access nodes – at least one node is needed[1], i.e. the one on which the Federation Router will be hosted - and application-related properties (e.g. library dependencies, classpath, etc.).

For the deployment, a set of scripts allow the automatic generation and usage of these XML descriptors, only depending on the definition of gateway nodes and protocols to be used. So, in order to launch locally a federation router and let PEtALS containers connect to it, service prosumers only need to use the following script with its options:

```
Usage:./FederationTransporter.sh -t <topology.xml> -n <server_node> [options]
        Mandatory parameters are:
                -n      <server>   FederationRouter
        Options are:
                -t      <topology.xml> // useful to get containers address
                -c      Run console mode
                -d      FederationRouter home folder on server
                -p      ProActive home folder on server
                -v      Verbose mode
```

For the management, we provide a console which allows a DSB manager to decide the DSB must join or quit a federation (e.g. join means establish a binding to at least an entry point of the federation, each binding reflects a partnership between this DSB and a partner DSB), retrieve information about federation current topology (i.e., current partners and relationships). The current console interface accepts the following commands:

```
   'h' or 'help' : print help message
   'c' or 'connect
             <host> <protocol> <port> :connect the console to the representative Federation Router
             <URL>: same as host, but passing an URL
   'j' or 'join'
             <host> <protocol> <port> : bind to the given federation entry point
            <URL>: same as host, but passing an URL
   'i' or 'info' : print information about the federation topology
   'x' or 'exit' : leave the federation, breaking all established bindings from this DSB to its former
             partners, but able to join again later
   'q' or 'quit' : definitively leave the federation
```

[1] Indeed, it might be possible to have more than one federation router per DSB, so to have many access points within the federation.

3.2 Support for External Services

There are two main categories of services which are addressed within SOA4All:

1. Platform services: These services are the ones which are developed by the SOA4All partners and will be potentially composed together to offer the core functionalities of the project. These services are mostly Web services ones (SOAP/HTTP) and REST ones. The platform services will be bound to the DSB with the help of the SOAP and REST connectors provided by Petals ESB.
2. Third party services: These services are the ones which are developed and hosted by third party Web actors. These services will be composed and/or invoked by SOA4All consumers in order to create client applications. These services are mainly SOAP/HTTP and REST ones.

The Distributed Service Bus provides access points on each main node to invoke platform services and third party ones. The following sections will give details on how the services are bound to the bus and how they can be accessed.

3.2.1 Platform Services

The illustration below (Figure 3) gives details on how platform services are bound to the DSB. The figure serves as example support in the continuation to illustrate how WSDL/SOAP-based, as well as RESTful platform services bindings are realized.

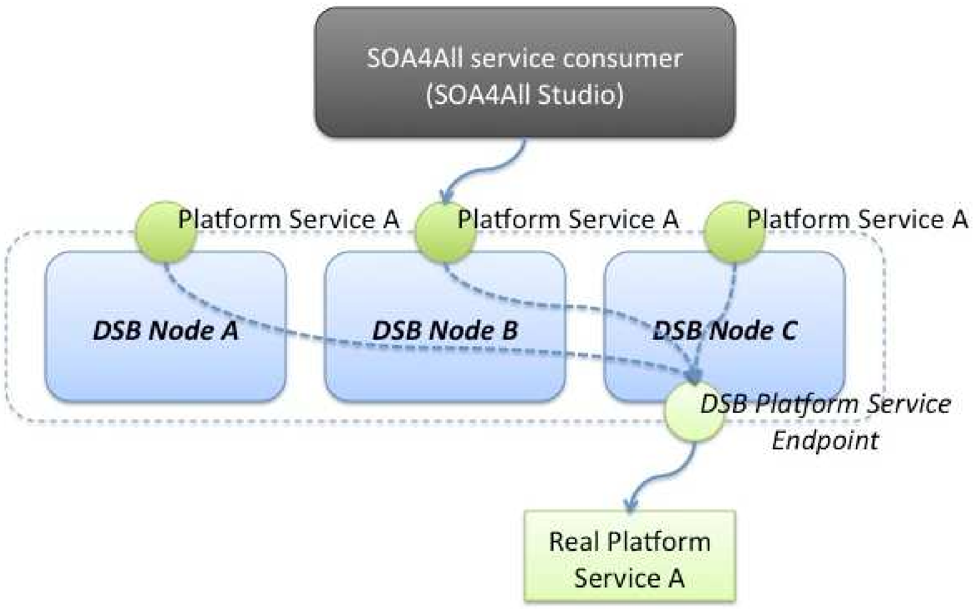

Figure 3: Platform Services Bindings

3.2.1.1 SOAP Platform Services

SOAP-based services are bound to the DSB using the Petals SOAP Binding Component (BC). This component is in charge of exposing DSB services are SOAP based Web services and to expose external SOAP Web services as DSB service.

At the SOA4All manager level, all that is needed to bind a SOAP service to the service bus is to call the bind operation of the Management API on a DSB node. The best practice is to choose the DSB node which closest to the platform service to bind. The deployment process follows the following steps:

a) Invoke bind(URI wsdl) on the management API

b) The DSB deploys local configurations to the SOAP binding component to bind the SOAP service given in step a).

c) After configuration deployment, a DSB endpoint is activated and is ready to forward calls to the SOAP service given in step a).

d) The other DSB nodes will detect that a new DSB endpoint is active and will deploy the configuration to expose this new endpoint as SOAP Web service.

As a result, a platform service will be available in two ways (Figure 3):

1. As an internal DSB endpoint. It means that internal DSB service consumer can invoke the service directly.
2. As Web services. External service consumer can send SOAP requests to the Web service access point of their choice. The SOAP request will be translated by the DSB and will be routed to the right DSB endpoint.

In Figure 3, the platform service A is bound to the DSB by the node C. It means that the SOA4All manager has called the bind operation of the management API on node C. The 'DSB platform service endpoint' is active and visible to all the three DSB nodes. The management engine of the DSB is, in this case, configured to expose the 'DSB Platform Service Endpoint' as Web service ('Platform Service A') on each DSB node. It means that a call to any 'Platform Service A' is forwarded to the 'DSB Platform Service A'. This is totally transparent on the 'SOA4All service consumer' side. The DSB message will be transmitted from a DSB node to the final one by the DSB routing and transport mechanisms.

Accessing a SOAP platform service

A 'Real Platform Web Service' which is hosted on http://<HOST>:<PORT>/services/SemanticSpaceWS and which has a WSDL description where the 'Port name' is 'SemanticSpaceWSImplPort', the 'Service name' is 'SemanticSpaceWSImplService' and the 'PortType name' is 'SemanticSpaceWS' will generate (after binding) a DSB endpoint where the endpoint name is 'SemanticSpaceWSImplPort', the service name is 'SemanticSpaceWSImplService' and the interface name is 'SemanticSpaceWS':

- WSDL port name is the DSB endpoint name
- WSDL service name is the DSB endpoint service name
- WSDL prototype name is the DSB endpoint interface name

Once the DSB endpoint is active, the endpoint introduced before will be exposed as Web service on each DSB node and accessible at http://<DSBHOST>:<SOAPPORT>/petals/services/SemanticSpaceWSImplPortService:

- DSBHOST is the host name of the DSB node
- SOAPPORT is the port which accepts incoming SOAP calls. The default value is set to 8084.

The final service name (SemanticSpaceWSImplPortService) is the DSB endpoint name suffixed with 'Service' or the initial Platform service port name suffixed with 'Service'.

3.2.1.2 REST Platform Services

REST-based services are bound to the DSB using the Petals REST Binding Component (BC). This JBI binding component is in charge of exposing DSB services as REST based services and to expose external REST services as DSB service.

Binding a platform REST service to the DSB is possible by invoking a binding operation on the management API of the bus. The binding process is described as :

a) Call bind(URL rest) on the management API

b) The DSB deploys local configuration to the REST component which is in charge of REST service support.

c) A new endpoint is activated on the REST component and on the DSB node. All the DSB messages which will be send to the newly REST activated endpoint will be forwarded to the service defined in step a).

d) The other DSB nodes will detect that a new endpoint as just been activated and willl deploy the entire mandatory configuration which is useful to expose the new REST endpoint as REST service.

As a result, the REST service will be available as:

1. A DSB endpoint on node where the bind operation has been called. All the calls to this endpoint will be forwarded to the real REST service.
2. A REST service on all the nodes of the DSB network. All the calls to one of this service will be forwarded to the right DSB endpoint, and so to the REST service itself.

In Figure 3, the REST platform service A is bound to the DSB by the node C. It means that the SOA4All manager has called the bind operation of the management API on node C. The 'DSB platform service endpoint' is active and visible to all the three DSB nodes. The management engine of the DSB is, in this case, configured to expose the 'DSB Platform Service Endpoint' as REST service ('Platform Service A') on each DSB node. It means that a call to any 'Platform Service A' is forwarded to the 'DSB Platform Service A'. This is totally transparent on the 'SOA4All service consumer' side. The DSB message will be transmitted from a DSB node to the final one by the DSB routing and transport mechanisms. When the 'DSB platform service A' receives the message, it translates it to the right format and send it to the real REST service.

Accessing a REST platform service

As an example, a REST service which is hosted on http://<HOST>:<PORT>/services/rest/RESTService will be exposed as a DSB endpoint where RESTService will be the endpoint name. All the messages which are received by the REST endpoint will be analyzed and used to build the REST call like:

- The DSB message operation will be used as REST operation. A message that has an operation which is not REST compliant (GET/POST/PUT/DELETE/HEAD) will be rejected.
- The DSB message content will be used as REST message content (all messages content types are accepted)
- The DSB message property called 'rest.resource.path' will be used to build the REST resource URL to call. For example, a DSB message where 'rest.resource.path' = 'resource/foo/bar' will be sent to http://<HOST>:<PORT>/services/rest/RESTService/resource/foo/bar.

Once the DSB endpoint is active, it will be exposed as a REST service on http://<DSBHOST>:<RESTPORT>/rest/services/RESTService where:

- DSBHOST is the DSB host name
- RESTPORT is the port number on which REST services are exposed. This port is defined at the REST binding component configuration level and is unique on each DSB node

All the calls to this REST service will then be sent to the RESTService DSB endpoint and processed as previously described.

3.2.2 Third-Party Services

Third party services can be invoked by SOA4All client (SOA4All studio) directly or using the DSB facilities. Using the DSB to invoke these services means that the DSB acts as a proxy i.e., calling a service through the DSB must be transparent at the service consumer point of view. By transparent, we want to say that the message payload is the same but the endpoint is adapted to reach the DSB. The DSB role is (not only) to route the message to the right DSB endpoint which is able to call the real third party service (Figure 4).

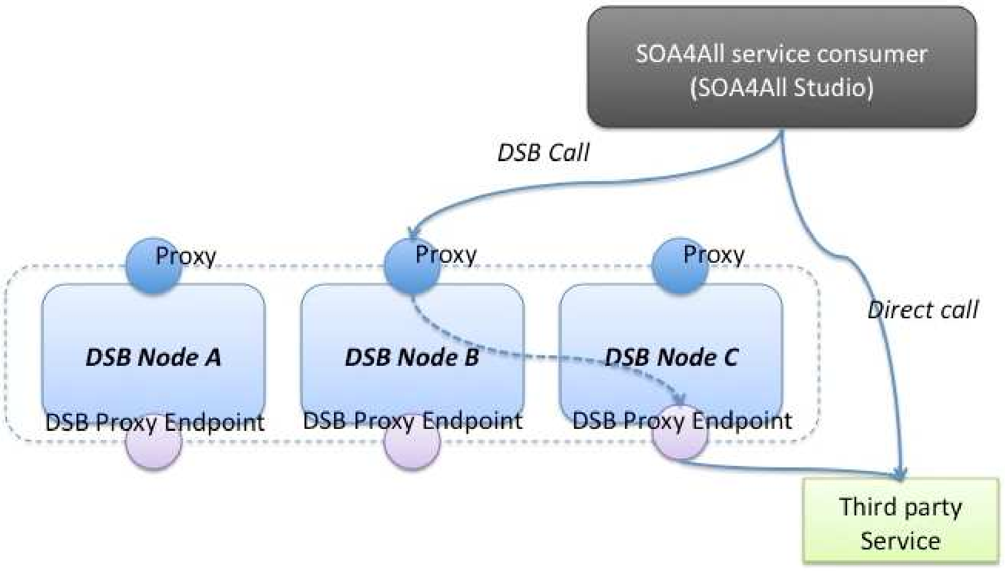

Figure 4: Services Proxy

In this section we will only focus on how the DSB provides access points to invoke these services.

3.2.2.1 SOAP Third-Party Services

The SOAP support for third party Web services is provided by the Petals ESB component named ‘petals-bc-soapproxy’. This component does not have the same behaviour than the ‘petals-bc-soap’ which is used to provide platform service support.

The SOAP proxy component exposes a Web service which is ‘WS-Addressing aware’ (more about Ws-Addressing is available at http://www.w3.org/Submission/ws-addressing/). It means that the WS-Addressing information which is available in the SOAP message header will be used by the DSB to invoke the final third party service. On the consumer side, using the DSB to invoke a third party service using the DSB means (Figure 4):

1. Adapt the client code to use WS-Addressing (if not already used). Standard WS stacks such as Apache Axis2 or Apache CXF provide a simple and clear API for that.
2. Inject the final third party service endpoint URL into the WSA-To value
3. Send the SOAP message to the DSB proxy endpoint

The DSB will get all the required information and will invoke the right DSB endpoint. This DSB endpoint is ‘SOAP and WS-Addressing aware’ and will call the final third party Web service. The response will be sent back to the primary consumer by the DSB.

Accessing a third-party SOAP service

If a Web service consumer wants to invoke a third-party service available at http://<HOST>:<PORT>/services/MyService, it must call the Web service http://<DSBHOST>:<SOAPPROXYPORT>/petals/services/ProxyService where:

- DSBHOST is the hostname of the DSB node
- SOAPPROXYPORT is the port which listens to SOAP calls. The default value is set to 8084

The SOAP message must have a SOAP message header with WS-Addressing section where the WSA-To value is equals to http://localhost:8080/services/MyService like in the following XML snippet:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <S:Header>
    ...
    <wsa:To S:mustUnderstand="0">
          http://<HOST>:<PORT>/services/MyService
    </wsa:To>
    ...
  </S:Header>
  <S:Body>
   ...
  </S:Body>
</S:Envelope>
```

Note that the mustUnderstand attribute must be set to 0/false in order to use the SOAP proxy. If not, the SOAP proxy will return a SOAP fault on each message call.

3.2.2.2 REST Third-Party Services

The REST support for third party services is provided by a Petals ESB component named ‘the rest-proxy component’. This component is deployed on each DSB node and starts an embedded HTTP server which will process and forward incoming REST requests to the right DSB endpoint. The DSB endpoint will get the message, invoke the REST service and send back the response to the DSB client ie the REST service consumer.

In the scenario described in Figure 4, the DSB chooses the best endpoint to send the DSB message to randomly. The default implementation of the DSB chooses the final DSB REST endpoint randomly from the list of DSB REST endpoints. A good evolution will be to add routing module to the DSB to select a better endpoint based on the final REST service to address. The criteria choices can be for example DSB node location (closest to the final third party service location), DSB load, DSB response time, amongst others.

Accessing a third-party REST service

If a REST service consumer wants to invoke a REST service at http://<HOST>:<PORT>/rest/services/RestService, it must call http://<DSBNODE>:<RESTPROXYPORT>/petals/rest/proxy/http://<HOST>:<PORT>/rest/services/RestService where:

- DSBNODE is hostname of the DSB node.
- RESTPROXYPORT is the port on which REST requests are accepted. The default value is set to 8989.

Note that this is the only thing to change from the standard way to call the third party service directly, i.e., the original HTTP operation (GET/POST/PUT/DELETE/…) and the HTTP message payload remain the same i.e. the client has just to update the service URL to call to the proxy one.

3.2.2.3Authentication

A majority of third-party business services will require a user authentication. This implies that the REST proxy service must also provide this authentication feature. The proxy service will handle authentication like:

- On REST message reception, get the http authentication related headers and assign authentication values to the DSB message
- Send the DSB message to the DSB REST proxy endpoint
- When the DSB REST proxy endpoint receives a message with authentication information, use this information to build the REST message.

The REST proxy component uses standard and well known and widely used libraries. To build a flexible REST proxy component, the Apache http-Component client library is used. This Library provides all the authentication schemes which are defined in HTTP RFCs. The DSB REST proxy endpoint will analyze the authentication headers of the incoming REST call and will use the right authentication classes to authenticate the user.

Important Note on HTTP Headers

HTTP response sent back from the service can contain headers that the service client can use (HTTP redirect for example). When the proxy component detects a header with an URL, it must rewrite the header value in order to use the proxy feature on next calls:

- Client send a REST request to the proxy
- The proxy send the REST request to the REST service
- The REST service sends back a response with HTTP redirect to http://host/foo/bar
- The REST proxy must return the HTTP redirect address in the HTTP header to the client after updating it like http://<DSBHOST>:<RESTPROXYPORT>/petals/rest/proxy/http://host/foo/bar
- The REST service client will handle the response and will call (probably) http://<DSBHOST>:<RESTPROXYPORT>/petals/rest/proxy/http://host/foo/bar instead of calling directly http://host/foo/bar

Such a mechanism on HTTP headers ensures that all the REST services calls are passing through the REST proxy.

3.3 Active QoS Monitoring Infrastructure

The monitoring infrastructure is designed to support both passive and active monitoring on SOAP and REST services. Passive monitoring, also known as real-user monitoring, refers to the approach that tracks the quality of services (QoS) as well as the end-user behaviours by capturing all the messages that go across the DSB as users invoke external services. Passive monitoring on Web services is described in previous deliverables [2][3]. On the other hand, active monitoring, also known as synthetic monitoring, is to test the performance of services by regularly sending faked requests that, to a certain extent, simulate the actions of actual users. Compared with passive monitoring, active monitoring can evaluate less QoS metrics and may cause measurement errors, because a faked request might be only able to trigger the exception handling rather than the normal business process of external services. However, active monitoring not only enables continuously observing on external services, but also provides up-to-date information about its QoS metrics such as availability, accessibility, latency, etc. These metrics can be used as criteria for service ranking, selection and recommendation. This section lists the definition of computable metrics by active monitoring and details measurement methods of metrics as well as the architecture of the active QoS monitor.

3.3.1 Metrics

A number of metrics have been defined to assess the quality of Web services [13][14][15], which also have been divided into two main categories: subjective and objective metrics [16]. Subjective metrics, e.g. reputation, satisfaction, are mainly dependent upon user experience and feedbacks, whereas objective metrics are those can be impersonally and quantitatively evaluated. It is manifested that an active QoS monitor can only measure some of the objective metrics, which are further divided into two categories: performance index and non-performance attribute. Performance indexes including availability, accessibility, response time and the lower bound of throughput indicate the run-time characteristics of Web services from the consumer's perspective. On the contrary, annotation qualities including interoperability and comprehensibility are the metrics can be computed based on service description and its semantic annotation.

1. Availability

In the context of service-oriented computing, availability means that a Web service is present on-line and is, similar to [13][14], defined as the ratio of the time period when a service is available and the total observation period.

$$Availability = T_{available} / T_{observation}$$

2. Accessibility

Like availability, accessibility also reflects the processing capability of a Web service, but it takes into account the delay in responding to requests. In other words, a Web service is accessible only if it can deliver the execution results within a certain time frame. Therefore, accessibility is the ratio of the time period when a service is accessible and the total observation period.

$$Accessibility = T_{accessible} / T_{observation}$$

3. Response time

Response time refers to the time between the QoS monitor sending the request and receiving the response from the service provider.

$$T_{response} = t_s - t_r$$

4. Lower Bound of Throughput

As stated in [13][15][17], throughput is the maximum of requests that a Web service handled during a certain period of time. Because the influence on QoS caused by active monitoring itself must be kept at an acceptable level, it is infeasible to test the actual throughput of a Web service using synthetic requests. However, the lower bound of throughput still can be estimated by counting the number of received responses.

$$LowerBound(Throughput) = N_{response} / T_{observation}$$

5. Interoperability

The interoperability of a Web service is perceived from both respects of syntax and semantics. Syntactic interoperability indicates whether a Web service is in the compliance with a standard [13][15]. Semantic interoperability is a new metrics introduced to reveal if a service is easy to be consumed by a client of semantic Web services. Semantic interoperability refers to the completeness and consistence of service annotations, i.e. the degree of obedience to five rules defined in [18].

6. Comprehensibility

Comprehensibility means the possibility of a semantic Web service understand by machines. In more detail, it contains two factors: the number of different annotations and the coverage rate of messages and operations in each annotation. Service annotations may change over time, thus monitoring on comprehensibility is helpful, especially for the discovery, selection and recommendation of services.

$$Coverage = \frac{N_{AnnotatedMessage}}{N_{Message}} + \frac{N_{AnnotatedOperation}}{N_{Operation}}$$

3.3.2 Measurement

This sub-section details the measurement of performance indexes and the annotation qualities respectively. Different methods are employed for evaluating the quality of SOAP and REST services.

Performance Index

Evaluation of performance of Web services in an active manner involves automatic generation of synthetic requests and analysis of responses. As for SOAP services, these two functionalities are implemented based on the semantic annotation of input and output messages as well as the lowering and lifting schemas. Whereas, the generation of requests for REST services is driven by hRESTS and URI templates [19], and the analysis of responses is based on the HTTP status and content of output messages. In addition, the exchanging messages captured by passive monitoring can also be used for active monitoring.

If an input message of a SOAP service is annotated with concepts of an ontology, and it has lowering schema, first, some instances of the concepts are created by setting the properties with random values. And then, using the lowering schema, the instances are transformed into messages that comprise the request to be sent to the service. If an input message unfortunately does not have semantic annotation or lowering schema, the generation of request will be carried out according to declaration of the message in the WSDL file. Similarly, if an output message is annotated and has lifting schema, it will be tried to transform the response into instances of the concepts used to annotate the output message.

In the case of a REST service, a request is generated by instantiating the URI template according to the semantic annotation or by using random values. Before the content of output message, HTTP status is first processed to analyze the response, because it provides an important basis for determining whether the service is available and accessible. For example, if HTTP status with code 404 implies that the service is currently unavailable.

Annotation Quality

As aforementioned, the evaluation of annotation quality is only based on analysis of the service description and its semantic annotation rather than the run-time characteristics. Counting the number of different annotation and calculating the coverage rate are all done by executing queries with the service repository. In order to get the up-to-date information, all the annotation qualities are recomputed every time a new annotation is stored into the service repository.

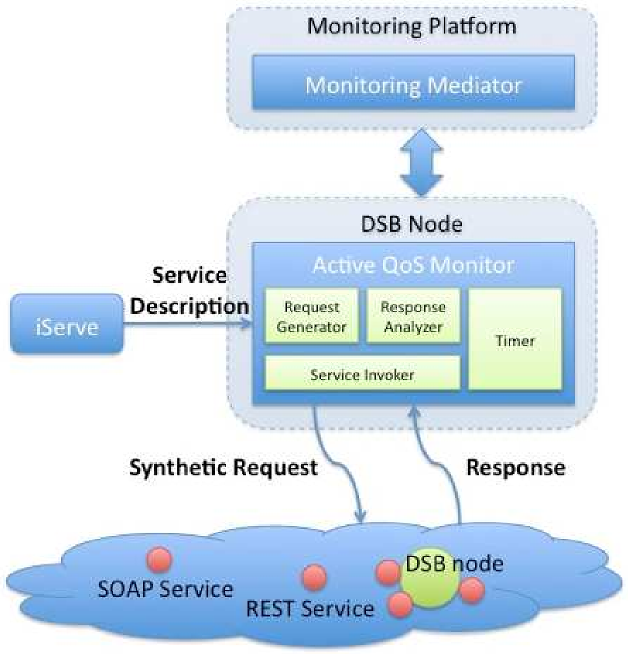

Figure 5. Active QoS Monitoring

3.3.3 Architecture

As shown in Figure 5, active QoS monitoring is composed of four key components: a request generator, a service invoker, a response analyzer and a timer. First, the QoS monitor retrieves service descriptions from iServe at a regular interval, and then the request generator carries out the method mentioned above to create synthetic requests. Note that the request generator will ignore the operations that have specific pre-conditions or effects on the real world. As a result, the overall performance of a Web service is measured not by active monitoring but by the aggregation of data from both the passive and active monitoring. The service invoker is in charge of sending requests and receiving the responses, which is implemented based on Axis 2 Client API [20]. The response analyzer sends the analysis results to the mediator of monitoring platform in the format of events, so that the data comes from active QoS monitor can be aggregated with those from the passive monitor. The Monitoring Mediator shown in Figure 7 serves as the interface to the monitoring platform. When Monitoring Mediator receives the events sent by the active monitor, it will filter, if nessary, and send them to Business Event Processor for further processing and analysis [21]. Finally, a timer triggers the invocation of services and computation of QoS metrics, and it is also configurable in order to keep under control the negative influences of active QoS monitoring on the performance of services.

3.3.4 Integration with SOA4All Studio

The active QoS monitor will also be exposed as RESTful services that are oriented towards the users as well as other components of SOA4All studio or third-party systems. As the Monitoring Mediator is the interface of the monitoring module of SOA4All Studio, it connects to active QoS monitor and transforms the data received into API calls to the Basic Event Processor (BEP). BEP is responsible for parsing the monitoring data received from the MM and to perform data processing in order to extracts derived information (such as computing averages and basic aggregated events). In addition, BEP also communicates raw and derived data to upper-level processing entities, the K-Analytics and SENTINEL engines [21], and updates of the graphical data structures used by the UI widgets of SOA4All.

4. Platform Services and Integration

4.1 Components

In this section we present eight platform services and three additional components which jointly provide the minimal functionality that is necessary for the realization of a delivery platform for Web services; recall the platform services at the bottom of Figure 1. Focus is set on updated component descriptions and a tabular specification of the access interfaces – some platform services offer a RESTful interface, while others are exposed via WSDL descriptions. The application of the platform services is explained in Section 4.2 by means of a consolidated discovering and constructing example. The platform services are grouped according to their area of application: service localization and service composition. The listing concludes with the presentation of the three support components for data grounding, reasoning and service description validation.

The category service location covers services that provide users with the possibility to find and rank services according to their needs and context. The category includes the crawler, the service repository, the discovery services, and the ranking and selection services.

Crawler: The crawler service offers a fundamental functionality for service discovery [22]. It deals with the collection of information related to services from the Web and the management of this data for enabling efficient and intelligent retrieval of service related artifacts. The crawler service takes thus care of searching for technical descriptions associated with services, also including related documents such as Web site, documentations, pricing and licensing information. The collected data is delivered either as RDF metadata, or as consolidated non-RDF archive files, and can be queried by means of a REST service endpoint. Furthermore, the crawler service's interface exposes interfaces to configure parameters and to define requirements in order to guide the crawler service in searching for desired service characteristics.

REST service	https://lanz.seekda.com/tomcat/crawldata/
Resource	GET
/sets	Description: list IDs of all crawl sets (i.e. a distinct crawl batch) Parameters: • *start* (optional): ordinal position of first result (in chronological order). Default value is 0 • *count* (optional): number of results to return. Default value is 100. Return: list of crawlset IDs (XML/RDF)
/sets/<*set-id*>/webservices/	Description: list IDs of all services in a specific crawlset Parameters: • *start* (optional): ordinal position of first result (in chronological order). Default value is 0 • *count* (optional): number of results to return. Default value is 100. Return: list of service IDs (XML/RDF)
/sets/<*set-id*>/webservices/<*service-id*>	Description: returns the service description documents (WSDL or HTML) Parameters: none Return: ZIP archive containing all the files (service descriptions)
/sets/<*set-id*>/webservices/<*service-*	Description: returns *all* documents related to a service (not service descriptions!)

id>/related	Parameters: none Return: ZIP archive containing all the files (related documents)
/sets/<*set-id*>/webservices/<*service-id*>/metadata	Description: returns *all* the metadata for the service (RDF triples according to the ServiceFinder ontology) Parameters: none Return: XML/RDF
/sets/<*set-id*>/query	Description: SPARQL query over the metadata for the services (in a specific crawlset) Parameters: q (mandatory): a valid SPARQL expression Return: XML/RDF

Service Repository: The service repository – in SOA4All released under the name iServe – is realized as public service and yields a central component of the service location category. iServe is used to store and maintain the collected service descriptions in RDF, and provides processing capabilities for SA-WSDL and MicroWSMO annotations that allow users and machines to upload their service annotations and have them automatically exposed as RDF [23]. The repository offers a Web interface allowing users to browse, query and upload annotations. Additionally, a RESTful API supports direct communications with machines for the very same purposes. Through iServe all service annotations provided through SOA4All become part of the global Linked Data cloud (www.linkeddata.org); more concretely they are stored in and queried through a repository-bound semantic space. The alignment with the Linked Data initiative increases public awareness and brings the service annotations into context. Moreover, potential service users have a more direct access to the descriptions to leverage them in locating service endpoints and in invoking and utilizing services within compositions.

REST service	http://iserve-dev.kmi.open.ac.uk:8080/iserve/data/
Resource	GET
/services	Description: returns the list of URI of services stored in the repository Return: SPARQL result set
/services/<serviceId>	Description: returns the description of the required service Return: service information as RDF/XML
/data/documents	Description: returns the list of URI of documents stored in the repository Return: SPARQL result set
/data/documents/<documentId>	Description: returns the contents of the required document Return: document (HTML, WSDL…)
/data/executequery	Description: returns the results of query execution Parameters: • query: the SPARQL SELECT or DESCRIBE query string Return: SPARQL result
Resource	POST
/services	Description: store a service in the repository Parameters:

	• format: Required, enumeration ("HTML", "WSDL", "OWLS"). The format of service description. • description: Required. The service description. • user: Required. The user name.
/data/documents	Description: store a HTML, WSDL or OWL-S file to the repository Parameters: • name: Required. The name of the file. • content: Required. The content of the file. • user: Required. The user name.

Discovery: Service discovery is all about the finding of services that match the need of a user. Traditionally, in Semantic Web services, the user perspective is specified by means of goals. SOA4All takes a more light-weight approach and offers the possibility to specify service templates that abstractly describe a user's objectives as a set of RDF triples. The service templates are designed in such ways that they are straightforwardly mappable into SPARQL queries to be resolved by repositories such as iServe. The service template RDF schema is shown in Table 1: the hasFunctionalCategory property shadows the functional classification reference that types services according to WSMO-Lite; the input and output properties are used to specify information about the input and the expected output of a service – in particular the latter is of interest to users; finally, requirements and preferences are properties that allow for further constraints in a language of choice, including SPARQL or WSML [25]. These values match roughly the specification of pre-conditions and effects as known from WSMO-Lite and are generally of more complex nature than RDF only; e.g., as stated SPARQL or WSML are prominent examples.

Table 1: RDF Schema for service templates

```
ServiceTemplate rdf:type rdfs:Class .
hasFunctionalCategory rdf:type rdf:Property .
hasInput rdf:type rdf:Property .
hasOutput rdf:type rdf:Property .
hasPreference rdf:type rdf:Property .
hasRequirement rdf:type rdf:Property .
```

The discovery service of SOA4All currently offers two types of search. The first one is referred to as full text-based discovery and mostly exploits keyword matching over the service descriptions and related documents as provided by the crawler. The second approach is referred to semantic discovery and leverages the aforementioned service templates. In the simplest case, discovery is reduced to the resolution of the derived SPARQL query. For more sophisticated searches, for example when using requirements and preferences, the discovery service makes use of the reasoning framework that is developed within SOA4All; see at the end of this section.

WSDL Service:	SemanticDiscovery		
	Operation	executePostQuery	
		Input	executePostQueryRequest
		Output	executePostQueryResponse
	Operation	executePreQuery	
		Input	executePreQueryRequest
		Output	executePreQueryResponse
	Operation	functionalityDiscovery	
		Input	functionalityDiscovery

		Output	functionalityDiscoveryResponse
	Operation	getClassificationRoot	
		Input	getClassificationRootRequest
		Output	getClassificationRootResponse
	Operation	getInputMessage	
		Input	getInputMessageRequest
		Output	getInputMessageResponse
	Operation	getLiftingSchemaMapping	
		Input	getLiftingSchemaMappingRequest
		Output	getLiftingSchemaMappingResponse
	Operation	getLoweringSchemaMapping	
		Input	getLoweringSchemaMappingRequest
		Output	getLoweringSchemaMappingResponse
	Operation	getMessageModelRefs	
		Input	getMessageModelRefsRequest
		Output	getMessageModelRefsResponse
	Operation	getOperationModelRefs	
		Input	getOperationModelRefsRequest
		Output	getOperationModelRefsResponse
	Operation	getOperations	
		Input	getOperationsRequest
		Output	getOperationsResponse
	Operation	getOutputMessage	
		Input	getOutputMessageRequest
		Output	getOutputMessageResponse
	Operation	getServiceClassification	
		Input	getServiceClassificationRequest
		Output	getServiceClassificationResponse
	Operation	getServicePostconditionString	
		Input	getServicePostconditionStringRequest
		Output	getServicePostconditionStringResponse
	Operation	getServicePreconditionString	
		Input	getServicePreconditionStringRequest
		Output	getServicePreconditionStringResponse
	Operation	getSubConcepts	
		Input	getSubConceptsRequest
		Output	getSubConceptsResponse
	Operation	getWsdlUri	

		Input	getWsdlUriRequest
		Output	getWsdlUriResponse

Ranking and Selection: The ranking and selection services offer the possibility to rank services according to given user preferences on the non-functional properties of the services [26][27]. Ranking and selection services integrate several ranking approaches that exploit the data gathered through crawling and monitoring. The first method computes global rank for services based on monitoring data or quality of documentation. A second ranking approach is using non-functional properties of services – e.g., liability or financing – that are given by means of logical rules. This ranking and selection service then uses logical reasoning to compute ranking scores for services based on aggregated non-functional values and their matching degree in terms of user requirements. A third approach is a fuzzy logic-based ranking mechanism that considers an extended model of preferences including vagueness information.

WSDL Service:	RankingService		
	Operation	Rank	
		Input	RankRequest
		Output	RankResponse

The second category of platform services yields the functionality to compose services, i.e. construct processes, and to execute the compositions. SOA4All particularly focuses on empowering non-technical users in constructing service compositions, and hence bases its work on process languages and process templates for lightweight modeling that foster re-usability and flexibility in design. The term lightweight explicitly refers to the usability of the language from a user perspective. Moreover, processes should be adaptable to specific contexts both at design-time and run-time, and optimized to the particular usage scenarios. The language that is common to all service construction components is referred to as Lightweight Process Modeling Language (LPML, [28]). LPML is a combination of established process modeling concepts mostly derived from BPMN or BPEL and SOA4All-specific extensions. On the one hand LPML simplifies BPEL by only considering relevant subsets of the business process modeling languages – in order to reduce the complexity of process models; and on the other, it extends these existing approaches with means to interleave goal specifications and process templates. The process specification is thus governed by reuse and requirements rather than specification of service-bindings. Furthermore, LPML defines how semantic annotations can be attached to various elements of processes such as activities, goals, input and output parameters, and introduces the concept of data connectors that specify the data flow in mash-up-style service compositions. A simple example of an LPML process specification with one goal is given in Table 2 of Section 4. In the following we present the service construction-related platform services that all operate over LPML process models.

Design-Time Composer: The design-time composer service provides semi-automatic assistance in resolving unbound activities within a process specification, given by service templates. As such, within SOA4All, the service is mainly supporting the activities of the process editor component that is part of the consumption platform of the studio. In other words, the design-time composer supports the entire life-cycle of service composition, from supporting the process specification through elaboration of process and template expansions to the discovery and binding of service endpoints as activities within the processes. Side-issues that are tackled by the composer service are data mediation and resolution of compatibility problems at the level of service inputs and outputs via so-called service connectors and semantic link operators.

WSDL Service:	DTComposer

	Operation	checkIOSemanticCompatibility	
		Input	checkIOSemanticCompatibilityRequest
		Output	checkIOSemanticCompatibilityResponse
		Fault	DTComposerException
	Operation	registerDesignAnalysisAgent	
		Input	registerDesignAnalysisAgentRequest
		Output	registerDesignAnalysisAgentResponse
		Fault	DTComposerException
	Operation	registerDesignModificationRuleAgent	
		Input	registerDesignModificationRuleAgentRequest
		Output	registerDesignModificationRuleAgentResponse
		Fault	DTComposerException
	Operation	registerDesignModificationSemanticAgent	
		Input	registerDesignModificationSemanticAgentRequest
		Output	registerDesignModificationSemanticAgentResponse
		Fault	DTComposerException
	Operation	registerSemanticLinkOperatorAgent	
		Input	registerSemanticLinkOperatorAgentRequest
		Fault	DTComposerException
	Operation	resolveGoal	
		Input	resolveGoalRequest
		Output	resolveGoalResponse
		Fault	DTComposerException
	Operation	resolveGoalMS	
		Input	resolveGoalMSRequest
		Output	resolveGoalMSResponse
		Fault	DTComposerException
	Operation	resolveGoalWithTemplate	
		Input	resolveGoalWithTemplateRequest
		Output	resolveGoalWithTemplateResponse
		Fault	DTComposerException
	Operation	resolveGoalWithTemplateMS	
		Input	resolveGoalWithTemplateMSRequest
		Output	resolveGoalWithTemplateMSResponse
		Fault	DTComposerException
	Operation	resolveGoalWithWS	
		Input	resolveGoalWithWSRequest
		Output	resolveGoalWithWSResponse

		Fault	DTComposerException
	Operation	resolveGoalWithWSMS	
		Input	resolveGoalWithWSMSRequest
		Output	resolveGoalWithWSMSResponse
		Fault	DTComposerException
	Operation	resolveProcess	
		Input	resolveProcessRequest
		Output	resolveProcessResponse
		Fault	DTComposerException
	Operation	resolveProcessMS	
		Input	resolveProcessMSRequest
		Output	resolveProcessMSResponse
		Fault	DTComposerException
	Operation	resolveProcessWithTemplate	
		Input	resolveProcessWithTemplateRequest
		Output	resolveProcessWithTemplateResponse
		Fault	DTComposerException
	Operation	resolveProcessWithTemplateMS	
		Input	resolveProcessWithTemplateMSRequest
		Output	resolveProcessWithTemplateMSResponse
		Fault	DTComposerException
	Operation	resolveGoalWithTemplateMS	
		Input	resolveGoalWithTemplateMSRequest
		Output	resolveGoalWithTemplateMSResponse
		Fault	DTComposerException
	Operation	resolveProcessWithWS	
		Input	resolveProcessWithWSRequest
		Output	resolveProcessWithWSResponse
		Fault	DTComposerException
	Operation	resolveProcessWithWSMS	
		Input	resolveProcessWithWSMSRequest
		Output	resolveProcessWithWSMSResponse
		Fault	DTComposerException
	Operation	unregisterDesignAnalysisAgent	
		Input	unregisterDesignAnalysisAgentRequest
		Output	unregisterDesignAnalysisAgentResponse
		Fault	DTComposerException
	Operation	unregisterDesignModificationRuleAgent	

		Input	unregisterDesignModificationRuleAgentRequest
		Output	unregisterDesignModificationRuleAgentResponse
		Fault	DTComposerException
	Operation	unregisterDesignModificationSemanticAgent	
		Input	unregisterDesignModificationSemanticAgentRequest
		Output	unregisterDesignModificationSemanticAgentResponse
		Fault	DTComposerException

Composition Optimizer: The input to the composition optimizer service is the definition of a service composition (as provided through the design-time composer) for which an optimized and executable process specification is sought [29]. Although the composer helps in binding service endpoints, there remain aspects in the process specification that cannot be treated at design-time. A core task of the composition optimizer is thus to assign open service templates to relevant and executable services in order to derive an executable process. To this end, the composition optimizer exploits reasoning support when necessary, and considers an extensible quality criteria model by coupling quality of service attributes (aka non-functional properties) and the semantic descriptions of the process. The non-functional properties of services are valued by means of Quality of Services measures (e.g., response time, reliability, availability) while the semantic elements are valued according to the data flow within the given executable process.

WSDL Service:	Optimizer		
	Operation	Optimize	
		Input	optimizeRequest
		Output	optimizeResponse

Execution Engine: The execution engine is the last service to be called in the service construction process. It offers operations to deploy executable processes in a dedicated execution environment and as such to expose entire processes as invokable Web services. Furthermore, the execution service provides tooling to transform light-weight process descriptions into standardized process modeling notations, such as for example BPEL. Lastly, this service is responsible for the execution of given processes and thus the invocation of atomic services that constitute a process. In SOA4All, the execution engine service includes a set of basic mechanisms for adaptive run-time reconfiguration of execution plans in order to react to changes in the execution environment.

WSDL Service:	LPMDeployer		
	Operation	deployServiceLPM	
		Input	deployServiceLPMRequest
		Output	deployServiceLPMResponse
		Fault	DeploymentException

Template Generator: The template generator service is to some degree a meta-component of the service construction suite. In principle, its functionality is not required for the realization of service compositions; however, it provides an important contribution in terms of facilitating the realization of executable processes. After all, enabling non-technical users in creating business value through services is one of the main drivers of the SOA4All project. A main task of the template generator is to analyze service execution logs and to generate hierarchies of process templates and a corresponding taxonomy; this process is motivated by work in [30]. Additionally, it allows users to refine and manually create templates. Process templates are abstract representations of processes that reflect typical workflows that are often too complex or too costly to be formalized as processes from scratch. To this end, the template generator service offers pre-formalized skeletons of service compositions that serve as input to both the process editor and the design-time composer. Note that consequently, the Template Generator is not deployed as a Web service, but it is deployed as a GWT server-side component as it is invoked only by the SOA4All Studio. For this reason there are no interface information given.

To conclude the presentation of the components that constitute the SOA4All service delivery platform, we present three additional software packages. Data grounding supports the invocation of service endpoints. The reasoning services support components that have to interpret and reason about semantic descriptions of services and process. Finally, the WSMO-Lite service validator is used to ensure that service annotations are valid formal descriptions, at least syntactically.

Data Grounding: As the Web services that are addressed by SOA4All are either WSDL-based and thus XML-aware services, or RESTful services that expose operations as URLs and often return XML-based data, there remains an extensive need for switching back and forth between the semantic layer data of SOA4All and the concrete service protocols. The relationships between these two types of data are defined at design-time in the form of bidirectional transformations, and bound to the semantic descriptions as lifting and lowering schemas. In SA-WSDL, the transformation schemas are linked to the service description via the schema mapping attributes liftingSchemaMapping and loweringSchemaMapping, respectively. The data grounding services uses the pre-defined transformation schemas to map from one model to the other whenever a lifting to the semantic level or lowering to the syntactical level is required during the invocation of a service endpoint [31].

REST service	
Resource	GET
/genOntology	Description: Generates an ontology from XML Schema (XSD) Parameters: • _xsd:_ the URL of the source XML Schema Header Parameters: • _xsd-url:_ URL of the XML Schema file alternative to 'xsd' param) • _format_ (optional): result ontology format, one of: *owl*, *rdf/xml*, *rdf/xml-abbrev*, *n-triple*, *n3*. Default is *rdf/xml*. • _xsd-data_:- XSD file content (alternative to 'xsd' and 'xsd-url') Return: Ontology definition in the selected representation format
/genLoweringSchema	Description: Generates a lowering schema between ontology and XML Schema instances. The mapping is provided as an input parameter in XML format. Parameters: • _mapping:_ the URL of the XML mapping descriptor file Header Parameters: • _mapping:_ the URL of the XML mapping descriptor file (alternative to ' mapping ' query parameter) • _format:_ input ontology format: *owl* or *rdf*. Default is *rdf*. • _xsd-url_ (optional): the URL of XML Schema file which should be used, instead of the one specified in the mapping file • _onto-url_ (optional): the URL of ontology file which should be used, instead of the one specified in the mapping file • _mapping-data:_ XML mapping file content (alternative to

	'mapping' parameters') Return: Lowering XSL Transformation file
/genLiftingSchema	Description: Generates a lifting schema between XML Schema and ontology instances. The mapping is provided as an input parameter in XML format. Parameters: • *mapping:* the URL of the XML mapping descriptor file Header Parameters: • *mapping:* the URL of the XML mapping descriptor file (alternative to ' mapping ' query param) • *xsd-url* (optional): the URL of XML Schema file which should be used, instead of the one specified in the mapping file • *onto-url* (optional): the URL of ontology file which should be used, instead of the one specified in the mapping file • *mapping-data:* XML mapping file content (alternative to 'mapping' parameters') Return: Lifting XSL Transformation file
/genLiftingMap	Description: Generates an ontology from XML Schema (XSD) and an XSL Transformation from XML data to OWL instances from the generated ontology Parameters: • *xsd:* the URL of the XML Schema file Header Parameters: • *xsd-url:* the URL of the XML Schema file (alternative to 'xsd' param) • *format:* (optional) result ontology format, one of: *owl*, *rdf/xml*, *rdf/xml-abbrev*, *n-triple*, *n3*. Default is: *rdf/xmlxsd-data* - XSD file content (alternative to 'xsd' and 'xsd-url') Return: ZIP archive containing the ontology, XML mapping descriptor and the XSL Transformation

Reasoner: Various platform services require reasoning support in matching services and service templates based on their semantic descriptions. The reasoning service exposes a corresponding framework of robust and scalable reasoning components that are tailored for each of the WSML language variants [32]. A variety of interfaces allow for schema/instance reasoning, satisfiability/entailment checking and query answering. The reasoning service has configurable links to service repositories to load service descriptions, and to public semantic spaces to get access to domain ontologies that are required to conclude the desired reasoning tasks.

WSDL Service:	ReasonerService			
	Operation	Query		
		Input	queryRequest	
		Output	queryResponse	

WSMO-Lite Service Validator: The WSMO-Lite Service Validator is an online support tool that can be used to confirm WSMO-Lite service annotations syntactically. The service is a typical meta-component of the SOA4All service delivery platform in that it supports any platform service that has to deal with service descriptions without providing a core functionality itself. In addition, the service can also be used to validate manually developed service annotations, and thus offers a stand-alone component that is intended to facilitate the adoption of the WSMO-Lite approach to Semantic Web services and to increase the quality of service annotations.

REST service	http://km.aifb.uni-karlsruhe.de/services/ServiceValidator/services/
Resource	POST
/	Description: accepts a WSMO-Lite based service description and validates it. Supported Content-Types are application/rdf+xml and text/n3 (restricted to Turtle syntax). Parameters: • *model* (optional): can be set to "MSM" to force validation against the currently used minimal service model Return: Standard HTTP response codes

4.2 Integrated Example

As the categorization in Section 3.1 has shown, service delivery in the context of SOA4All focuses on two separable areas: service location and service construction; although the latter at least partly subsumes the former in order to bind actual service endpoints to service templates within a process. In this section we exemplify how the different platform services coordinate in order to jointly realize SOA4All's global service delivery platform.

As a starting point, for our example, we assume that there are a relevant number of semantic service descriptions provided and stored in the service repository. The service descriptions were likely created by use of the provisioning tools such as SWEET or SOWER of the SOA4All Studio [33], which are used for annotating Web APIs or WSDL files with either MicroWSMO or SA-WSDL, respectively. In order to find the service endpoints and the corresponding documentation, a SOA4All user can profit from the crawler service that provides the background knowledge to formalize service annotations. In principle, it would also be possible that the crawler service detects not only service endpoints, but service descriptions directly, which then, without manual intervention, can be stored in the repository. Such SOA4All compatible service descriptions are however not yet publicly available, as the tools and languages are still in the process of being established and first have to penetrate the market. Our assumed starting point thus covers most of the actions for which the provisioning platform is developed: service endpoint selection, semantic description creation, annotation management and storage in the service repository.

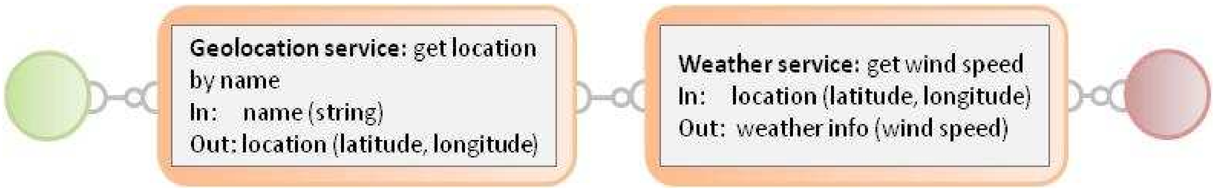
Geolocation service: get location by name
In: name (string)
Out: location (latitude, longitude)
Weather service: get wind speed
In: location (latitude, longitude)
Out: weather info (wind speed)

Figure 6: Example process with two services.

In terms of functional processes that are empowered by the service delivery platform, the actions triggered through the consumption platform are much more informative. A core component of the consumption platform is the mentioned process editor that yields a GUI for the modelling of processes. The graphical artefacts of a composition are specified by means of LPML, which is understood and further manipulated by all service construction-related platform services. An example process with two activities – and without flow information for clarity – is shown in Table 2 and Figure 3; note that the example shows the LPML model after the invocation of the design-time composer, and hence a potential Web service is already bound to each of the two activities. The first goal of the process is to determine the latitude and longitude of a city in order to retrieve information on the wind speed from the nearest weather station in a second step. Both services are offered by ws.geonames.org. As the output of the first activity maps directly onto the input of the second one (exact match), data mediation and I/O connector descriptions are skipped.

Table 2: LPML process description in XML

```
<org.soa4all.lpml.impl.ProcessImpl>
  …
      <activity class="org.soa4all.lpml.impl.ActivityImpl">
          <operation>getGeoLocationByName</operation>
          <conversation class="org.soa4all.lpml.impl.ConversationImpl">
            <goal class="org.soa4all.lpml.impl.GoalImpl">
              <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
                  <referenceURI>http://www.example.org/geolocation#location</referenceURI>
                  <type>FUNCTIONAL_CLASSIFICATION</type>
              </org.soa4all.lpml.impl.SemanticAnnotationImpl> <semanticAnnotations>
            </goal>
            <services>
              <serviceReference>http://ws.geonames.org/search</serviceReference>
            </services>
          </conversation>
          <inputParameters>
            <org.soa4all.lpml.impl.ParameterImpl>
              <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
                  <referenceURI>http://www.example.org/geo#locationString</referenceURI>
                  <type>META_DATA</type>
               </org.soa4all.lpml.impl.SemanticAnnotationImpl> </semanticAnnotations>
            </org.soa4all.lpml.impl.ParameterImpl>
          </inputParameters>
          <outputParameters>
            <org.soa4all.lpml.impl.ParameterImpl>
              <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
                  <referenceURI>http://www.w3.org/2003/01/geo/wgs84_pos#long</referenceURI>
                  <type>META_DATA</type>
              </org.soa4all.lpml.impl.SemanticAnnotationImpl> </semanticAnnotations>
            </org.soa4all.lpml.impl.ParameterImpl>
            <org.soa4all.lpml.impl.ParameterImpl>
              <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
                  <referenceURI>http://www.w3.org/2003/01/geo/wgs84_pos#lat</referenceURI>
                  …
            </org.soa4all.lpml.impl.ParameterImpl>
          </outputParameters>
          </activity>
          <activity class="org.soa4all.lpml.impl.ActivityImpl">
            <operation>getWindSpeed</operation>
            <conversation class="org.soa4all.lpml.impl.ConversationImpl">
              <goal class="org.soa4all.lpml.impl.GoalImpl">
                <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
                    <referenceURI>http://www.example.org/weather#WindSpeed</referenceURI>
                    …
            </goal>
            <services>
              <serviceReference>http://ws.geonames.org/findNearByWeatherXML</serviceReference>
            </services>
          </conversation>
          <inputParameters> … </inputParameters>
          <outputParameters>
            <org.soa4all.lpml.impl.ParameterImpl>
              <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
                  <referenceURI>http://www.geonames.org/ontology#windSpeed</referenceURI>
                  …
            </org.soa4all.lpml.impl.ParameterImpl>
          </outputParameters>
          </destination>
          ...
```

For applications that are based on prevalent workflows, a user can load a process template that was pre-designed by the template generator service in order to jump start the construction process. In our example, with a simple goal, this step is left out. A first contact point for our example is the design-time composer that allows for flexible and ad-hoc creation and adaptation of processes at design time. It is iteratively invoked during the process specification to assist in resolving goals, binding services, expanding templates, checking I/O compatibilities and creating data flow via connectors. In short, the design-time composer offers support in regards to all aspects of the LPML-based process specification and helps users in completing an eventually executable process model. These activities require service discovery support in finding adequate service bindings. Although matching user requirements and process-specific constraints, the outcome of this iterative collaboration between user, process editor and composer does not necessarily yield an optimized model. In fact, the design-time composer works mainly on local solutions only and does not care about the global optimization of a process specification – although some global process requirements and constraints are taken into account.

Mostly due to performance optimization, context adaptation or specific user preferences and constraints, it is necessary to optimize the completed compositions. While the input to the composer is generally a rather goal-heavy process specification, the optimizer only accepts complete process models for which it seeks a better global cost function in terms of functional (including semantic similarity of inputs and outputs) and non-functional qualities of services. The optimizer thus not only uses non-functional parameters such as well-known Quality of Service metrics (QoS) but also semantic similarity as a core indicator of functional quality. In summary, the optimizer – by exploiting Genetic Algorithms – transparently transforms compositions into their optimal versions by replacing service bindings and modifying the dataflow but without changing the workflow. Such transformations are heavily influenced by the context in which a service composition is to be used, and require reasoning support and service discovery for finding concrete and optimized service bindings. Again, as for the design-time composer, the optimizer service is optionally invoked to achieve better overall compositions.

Once a complete model is established that satisfies a user's preferences and requirements in terms of functionality and performance, the execution engine service is called in order to deploy the process in the execution engine and to expose it as a Web service. As described in Section 3, the execution engine transforms the complete LPML model into a BPEL model enhanced with some SOA4All-specific extensions. These extensions enable the use of semantic annotation for instructing the self-adaptation capabilities of the execution engine service. Additionally, a corresponding WSDL endpoint is specified to allow for public access. Consequently, the execution engine offers one more public service for any deployed process, in addition to the basic 'deployProcess' service that the engine hosts per default.

As the first part of our example has shown, discovery is an essential sub-task of the service construction process. To conclude this section, we thus enter more concretely into the domain of service discovery. The definition of activity, and in particular the goal element within the LPML process specification can be mapped straightforwardly onto the properties of the service templates (Table 1) which are at the basis of semantic discovery. The semantic annotations of type functional classification that are used to annotate a goal element are transferred to values of the hasFunctionalCategory property. The operations, given through inputOperation and outputOperation in LPML are converted to hasInput and hasOutput respectively. Last, there are other semantic annotations that can be attached to activities, which are of type requirement and non-functional property. These map to hasRequirement and hasPreference accordingly. Table 3 shows a concrete service template that was directly derived from mapping the LPML goal element in Table 2 according to the schema in Table 1. The 'st' prefix stands for the service template namespace http://cms-wg.sti2.org/ns/service-template#.

Table 3: Concrete service template for the wind speed service

```
windSpeedService rdf:type st:ServiceTemplate ;
    st:hasFunctionalCategory <http://www.example.org/weather#WindSpeed> ;
    st:hasInput rdf:type <http://www.w3.org/2003/01/geo/wgs84_pos#lat> ;
    st:hasInput rdf:type <http://www.w3.org/2003/01/geo/wgs84_pos#long> .
```

In a next step, SPARQL queries can be derived from the RDF-based service templates that can be executed against the semantic service descriptions in the service repository. A possible SPARQL query for the template in Table 3 is depicted in Table 4. Note that the classes and predicates are no longer taken from the service template schema but from the minimal service model (prefixed with msm) which borrow some elements from SA-WSDL, identified with the prefix sawsdl. The query selects services which match exactly the functional category that is searched, and that offer operations with the given input types. Had our service template contained hasOutput specifications, those would have appeared in the query in a similar way to the input types. More sophisticated query specifications that are investigated in the context of service location consider, in addition to the exact match achieved with the example in Table 4, plug-in, subsumes, and fail match degrees common in the literature. One approach to do this is to use subclass relations in the SPARQL query; i.e., the service classification does not reference wind speed directly but rather any subclass of the concept http://www.example.org/weather#WindSpeed. Executing the SPARQL query against the repository results in a collection of service endpoints that match the functional classification, and the input and output parameters respectively.

Table 4: SPARQL query to be executed against the service repository

```
SELECT ?service ?operation ?endpoint
WHERE {
    ?service rdf:type msm:Service ;
                rdfs:isDefinedBy ?endpoint ;
                msm:hasOperation ?operation ;
                sawsdl:modelReference <http://www.example.org/weather#WindSpeed> .
    ?operation msm:hasInputMessage ?input ;
                    msm:hasOutputMessage ?output .
    ?input sawsdl:modelReference <http://www.w3.org/2003/01/geo/wgs84_pos#long> ;
              sawsdl:modelReference <http://www.w3.org/2003/01/geo/wgs84_pos#lat> .
```

This passage from a goal specification to a set of possible service endpoints is followed for all goal elements in the LPML model. At the level of the design-time composer any of the discovered endpoints might be selectable, as they fulfill the basic requirements: typing, as well as input and output parameters. In order to optimize a process, it is however necessary to choose the service that best fits a user’s objective and expectations and ranking becomes important. The ranking and selection service is invoked with the set of possible service endpoints as input; i.e., ranking and selection depends on the discovery service too.

There are further, more sophisticated means defined in WSMO-Lite to annotate a service, namely pre-conditions, effects and non-functional parameters. These more complex elements of semantic service descriptions are in most cases given by WSML axioms that are persisted as RDF literals. The axioms are extracted from the service descriptions if needed and loaded into the reasoning service for query resolution in the scope of discovery, ranking and selection or process optimization.

4.3 SOA4All Functional Processes

In the previous sections we presented the SOA4All platform services and an example of the integrated usage of them. At this point it is important to address the integration perspective from a functional perspective, meaning the definition of a precise specification of functional processes involving the needs for platform services but particularly also infrastructure. The aim of this definition is twofold: i) define what SOA4All does and offers in terms of functionality and ii) check whether the platform services offer what is expected from them in terms of interfaces to complete the functional processes at hand.

Functional processes should include the service delivery platform functionality; i.e. functional processes operate over platform services. As a rule of thumb to define whether a process is functional or not, we could say that if no platform service is involved in a process, this process is not functional. Functional processes are grouped according to their area of application. To this extent we consider two main SOA4All functional areas: *service provisioning and service consumption*. These names are borrowed from the equivalent applications within the SOA4All Studio to facilitate the mapping between the user-oriented tools and the processes behind them. We do not consider any functional process coming from the analysis platform of the studio, because the analysis components of WP1 are considered to be part of the DSB infrastructure rather than platform services. Monitoring, as part of analysis, is a service of the runtime infrastructure rather than of the service delivery components; i.e., the platform services.

4.3.1 Service provisioning

The category service provisioning covers processes that allow the discovery, annotation and composition of services at design time, and the actual deployment of composed services to an execution engine. This category includes the service crawling, semantic annotation of WSDL and RESTful services and service composition.

Service Crawling: This process uses fundamentally the crawler service functionality for service discovery. As described for the crawler service, this process deals with the collection of information related to services from the Web and the management of this data for enabling efficient and intelligent retrieval of service related artefacts storing these descriptions either as RDF metadata (in the service repository platform service), or as consolidated non-RDF archive files (internal repository). As discussed in the integrated example before, the crawling process would also be able to detect semantic service descriptions directly, which then, without manual intervention, can be stored in the repository.

Hence the crawling process presents interaction between the following platform services: crawler and service repository. The crawler can be triggered by the SOA4All Studio.

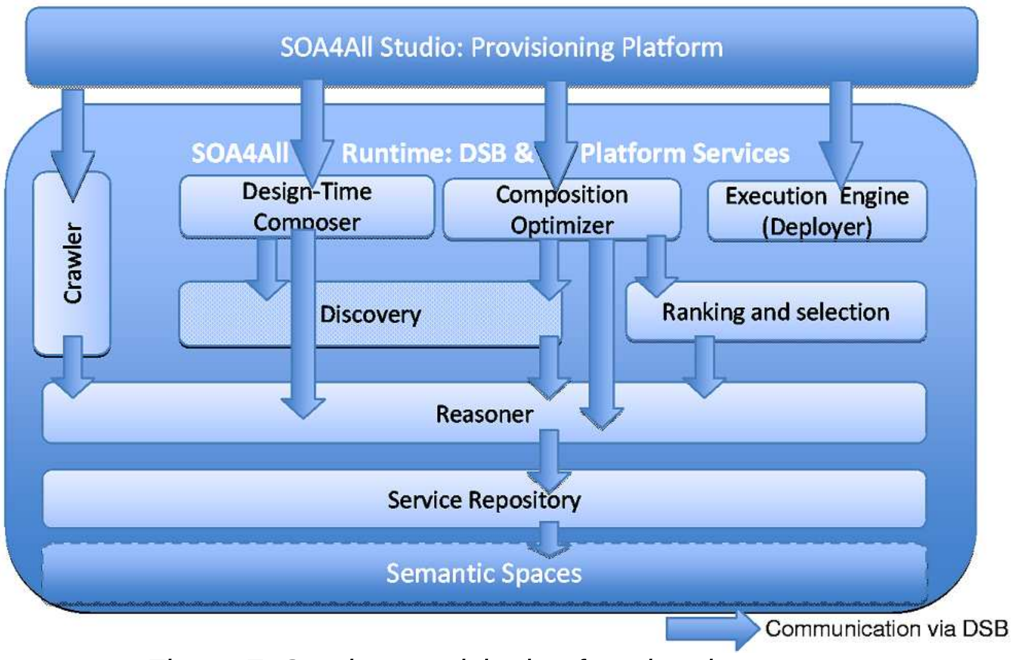

Figure 7: Service provisioning functional processes

Semantic Annotation of WSDL Services: The service descriptions are created by use of the provisioning tool SOWER of the SOA4All Studio, which is used for annotating WSDL files with SA-WSDL. The process assumes that we dispose of several service descriptions (provided for instance by the crawler service functionality for service discovery). The user proceeds to select the service endpoint and to semantically annotate the WSDL services using SOWER. The annotations are saved in the service repository. This process presents interaction only between the SOA4All Studio and one platform service, the service repository.

Semantic Annotation of RESTful Services: As in the previous process, the service descriptions are created by using SOA4All Studio functionality (in this case the SWEET tool), which is used for annotating RESTful services with Micro-WSMO. The annotations are saved in the service repository. This process presents interaction only between the SOA4All Studio (SWEET) and one platform service, the service repository.

Process Template Generation: Through this process the user is able to define new process templates using the template generator service. The templates are syntactic descriptions of a process saved as XML in a repository. This is intended for domain expert users that generate the templates for their usage in the process generation functional process. The process templates should be semantically annotated to allow their further discovery and reuse within the Process Management functional process. These templates use a similar set of annotations than other services, being the only difference that they are abstract in the sense that cannot be used to discover and invoke actual services

Service Composition: The Service Composition functional process involves all the steps of service construction in SOA4All in design time. It is related with multiple platform services and also to other SOA4All components, such as parts of the SOA4All Studio as can be seen in Figure 8, taken over from [33].

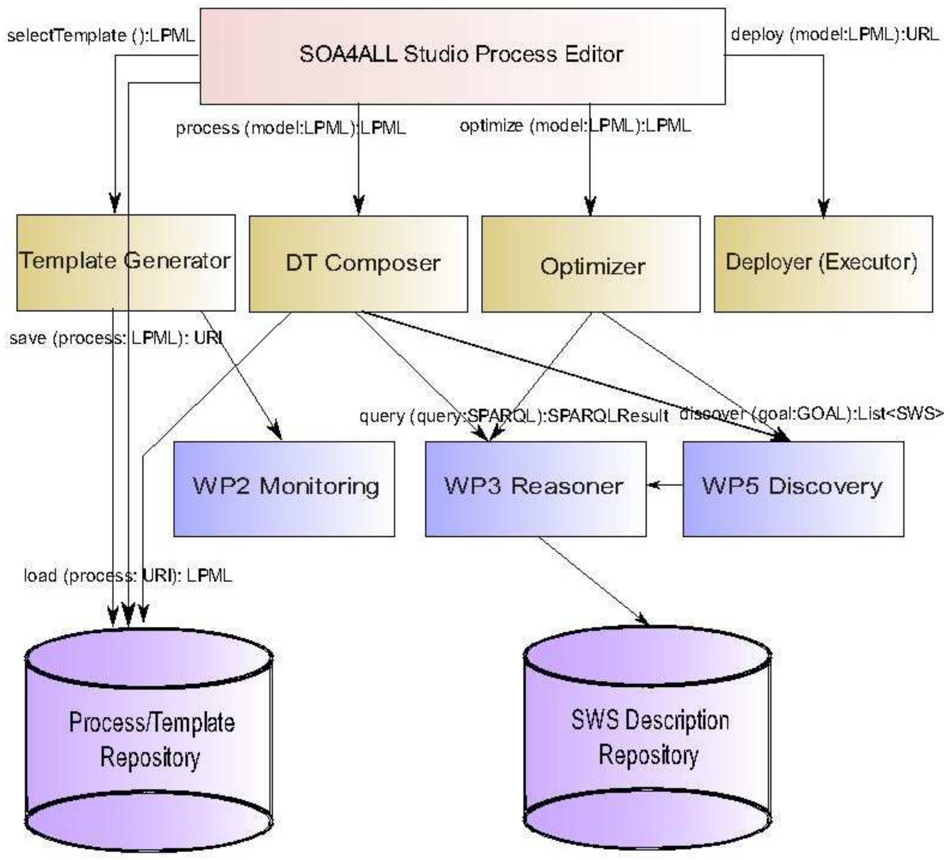
selectTemplate ():LPML
SOA4ALL Studio Process Editor
deploy (model:LPML):URL
process (model:LPML):LPML
optimize (model:LPML):LPML
Template Generator
DT Composer
Optimizer
Deployer (Executor)
save (process: LPML): URI
query (query:SPARQL):SPARQLResult
discover (goal:GOAL):List<SWS>
WP2 Monitoring
WP3 Reasoner
WP5 Discovery
load (process: URI): LPML
Process/Template Repository
SWS Description Repository

Figure 8: Design-time composition overview

This functional process has several steps:

- **Process Creation:** The Process Editor of the SOA4All Studio provides the process creation interface for the end users. As stated in the integration example before, the graphical artefacts of a service composition are specified by means of LPML. These LPML descriptions are saved in a syntactic repository. The user might start the definition of the process by loading a process template designed in the process template process described above or start drawing the process from scratch. Internally the design-time composer (DTC) platform service allows the creation of processes at design time. The DTC deals with tasks such as resolving goals, binding services, expanding templates, checking I/O compatibilities and creating data flow via connectors, helping users in completing an eventually executable process model. These activities require service discovery support in finding adequate service bindings. Although matching user requirements and process-specific constraints, the outcome of this iterative collaboration between user, process editor and composer does not necessarily yield an optimized model. In fact, the design-time composer works mainly on local solutions only and does not care about the global optimization of a process specification – although some global process requirements and constraints are taken into account.
- **Process Optimization:** Process optimization involves the usage of the process optimizer platform service. As it was stated before, this component might be used to further optimize the completed compositions created by the process creation functional process explained before. The optimizer seeks a better global cost function in terms of functional and non-functional qualities of services. The results of the optimizer are better service bindings and enhanced dataflow. In order to do that, the optimizer needs to use the discovery, ranking and selection and reasoning and services support. Hence, the process optimization involves the usage of the service repository, discovery, reasoning, ranking and selection and process optimizer platform services. It is optionally triggered by the process editor once complete service compositions are in place
- **Process Deployment:** Once a complete model is established that satisfies a user's preferences and requirements in terms of functionality and performance, the execution engine service is called in order to deploy the process in the execution engine and to expose it as a Web service. As described in Section 3, the execution engine transforms the complete LPML model into a BPEL model enhanced with some SOA4All-specific extensions. These extensions are heavily influenced by BPEL4SWS. Additionally, a corresponding WSDL endpoint is specified to allow for public access. Consequently, the execution engine offers one more public service for any deployed process, in addition to the basic 'deployProcess' service that the engine hosts per default.

This process presents interaction between the SOA4All Studio (Process Editor), and the following platform services: design-time composer, process optimizer discovery, reasoning, ranking and selection, the service repository and the deployment via execution engine (Figure 9). The process also makes use of the Template Generator component, which is not deployed as a platform service, but offers functionality in the process creation. The figure below shows a sequence diagram with a simplified view of the usage of the main components of the service composition functional process.

4.3.2 Service consumption

The category service consumption covers processes that allow the execution of semantically annotated services. These services can be single services (WSDL, RESTful or simple Web APIs) stored in the service repository, or composed services described using the process editor and deployed using the process deployment functional process described above.

These processes are deployed as BPEL services to the execution engine platform developed within SOA4All, and therefore exposing a WSDL interface. From the functional processes perspective, both services and processes are exactly the same and can be consumed as semantically annotated services using the same functionality.

Service Invocation: Service consumption includes the actual execution of semantically described services, thanks to the annotations performed with the Provisioning Platform tools and stored in the service repository. However, the actual execution of services still happens at syntactic level, requiring an interchange of messages, typically in XML in traditional WS services, but not restricted to those, for they might also be through invocation of URLs and involving other format of responses such as JSON in RESTful services. This means that it is necessary to ground information attached to the service. This information describes how the semantic data should be written in an XML form that can be sent to the service, and how XML data coming back from the service can be interpreted semantically by the platform. These two processes are respectively known as “lowering” and “lifting”. To perform these transformations the service invocation process uses the Service Grounding platform service.

It is important to notice that from the WP1 point of view, there is no difference between the services to be consumed from third-parties or services composed and deployed using the functional processes stated above. This is due to the fact that both types of services expose Web service interfaces and are annotated semantically in the same way. Hence the service consumption process presents interaction between the following platform services: data grounding service and service repository. The services are executed using the SOA4All Studio consumption platform.

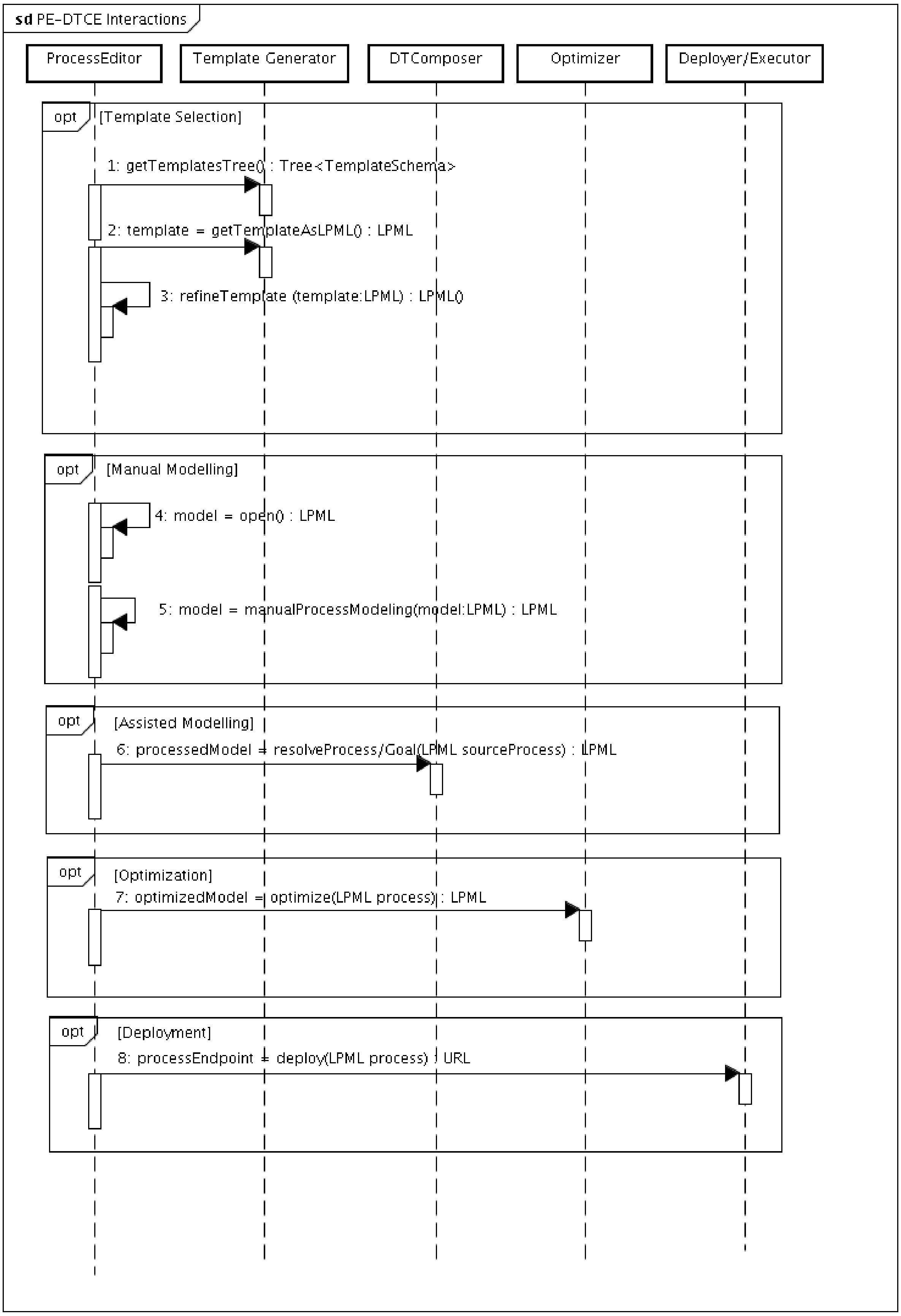
sd PE-DTCE Interactions
ProcessEditor
Template Generator
DTComposer
Optimizer
Deployer/Executor
opt
[Template Selection]
1: getTemplatesTree() : Tree<TemplateSchema>
2: template = getTemplateAsLPML() : LPML
3: refineTemplate (template:LPML) : LPML()
opt
[Manual Modelling]
4: model = open() : LPML
5: model = manualProcessModeling(model:LPML) : LPML
opt
[Assisted Modelling]
6: processedModel = resolveProcess/Goal(LPML sourceProcess) : LPML
opt
[Optimization]
7: optimizedModel = optimize(LPML process) : LPML
opt
[Deployment]
8: processEndpoint = deploy(LPML process) : URL

Figure 9: Service composition process sequence diagram

5. SOA4All Runtime Configurations and Use

5.1 Scenarios and Business Models

This section provides a quick overview of possible relevant business models for SOA4All infrastructure providers. We do not define which models should be followed, as this objective will be investigated within the context of WP10. Rather, the aim is to show the existing models that could be relevant, and provides advices for further research in WP10.

While trying to identify suitable business models for SOA4All infrastructure providers we should try to answer questions such as what we really mean with the term "infrastructure". This definition may be subject to different interpretations, and be reflected in different possible business models.

In a sense, the SOA4All infrastructure can be seen as the backbone to execute SOA4All Studio applications and where to execute third parties services and processes (along with support functionalities such as monitoring, QoS, etc.). It primarily refers to the federation of distributed service buses. In this sense, we can find several similarities with the Grid/Cloud paradigm, where different stakeholders are willing to offer/share their resources to run third party applications/services (in principle, regardless of the nature of such applications).

Furthermore, we may consider the SOA4All infrastructure so to also include SOA4All Studio components, thus covering two main categories of functionalities:

1. discovery of services (with all required functionalities to semantically annotate them, store annotations, reasoning, etc..)
2. provisioning of added-value services for service compositions and execution (i.e. process editor, execution engine, monitoring platform)

However, the incentives for offering platform services are much clearer as their direct benefit is more obvious, also in terms of financial revenues. Just as an example, we report here a short summary on what a project with similar objectives in terms of provisioning and consumption platforms is performing (see COIN IP at www.coin-ip.eu).

We may find a clear mapping of SOA4All architecture components with the XaaS stack:

a) SOA4All "hardware" infrastructure providers → PaaS (Platform-as-a-Service)
b) SOA4All "software" infrastructure providers (i.e. providers of the fDSB) → IaaS (Infrastructure-as-a-Service)
c) SOA4All Platform Services providers → SaaS (Software-as-a-Service)

It is unavoidable to investigate on possible similarities in terms of functionalities and business models with the most popular search engine on the Web: Google. Finally, we consider an infrastructure provider and associated business model which is under consideration within the SOA4All project i.e. the Telco 2.0 platform provider.

The following sections provide a quick reference and summary on how the role of an infrastructure provider is defined and which are the associated business models in different initiatives: Grid/Cloud projects, COIN IP, Google Inc., and Telco 2.0 Platform providers.

5.1.1 Grid/Cloud Infrastructures

A very comprehensive analysis on most recent R&D technical and business solutions in the area of Grid and Cloud computing can be found at: www.IT-Tude.com. The site analyses a wide number of business cases and business models, and derives a generic value chain and a taxonomy of business models which are relevant to SOA4All.

Generic Value Chain and Financial Flows: IT-Tude.com defines a "Generic Value Chain" (Figure 10). We identify two type of relevant actors for the role of SOA4All Infrastructure

Providers:

- *"Resource/Infrastructure" provider: provides equipment on which the Grid implementations run. This includes: hardware, network and system resources.*
- *"Resource/Infrastructure" operator: provides access to and use of the equipment that is owned by the resource or infrastructure provider.*

A further analysis (http://www.it-tude.com/financialflows.html) aims at identifying possible financial flows amongst these actors. Resource providers are classified with a pricing model involving an initial payment and a pricing-per-user.

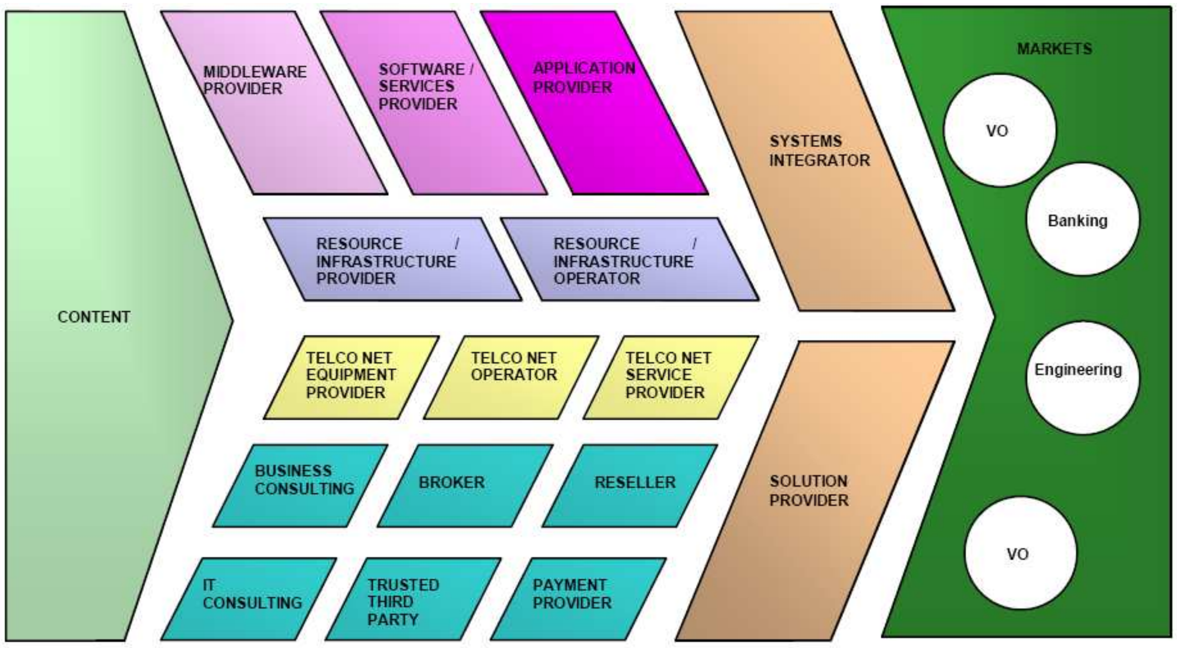

Figure 10: Generic Value Chain according to IT-Tude.com

Business Models: Several business models are analysed and classified by IT-Tude.com. These are grouped into three main categories:

- *Category 1: "Grid Business Cases with a clear performance-associated benefit",* addressing one of the following problems/limitations:
  - additional CPU power needed for executing a demanding application
  - huge amount of data storage/memory is required
  - access to heterogeneous, geographically distributed data resources is required.
- *Category 2: "Grid Business Cases with a highly collaborative benefit".* The benefit arises from sharing complementary resources among participating organizations, utilized for a common scope. Typical examples of this category are intra-organisational grids and Virtual Organisations and the expected economic benefit in this case could be shared among all participants in contrast to the first category where the main economic benefit is anticipated from the end-user where the service or application will be provided or sold. Also, the services of this category cannot be provided by a single provider since data or other resources are necessary to be obtained from other providers.
- *Category 3: "Grid Business Cases exploiting the component-based software paradigm"***.** This category comprising those business scenarios involving a service provider that offers applications on a pay-per-use basis rather than by means of licensing or long term

static agreements and thus exploiting to the most the concepts of the next generation Service Oriented Architectures

A more in-depth analysis of these three categories can be found at http://www.it-tude.com/gridclassification.html.

The question which model category is the most relevant one for SOA4ALL infrastructure providers could find several answers. Actually each of them seems to share relevant features with the SOA4All infrastructure:

- Category 1 fits with SOA4All infrastructure's need to access heterogeneous and distributed data sources.
- Category 2 is matching SOA4All infrastructure's need to share complementary resources which could not be provided by a single infrastructure provider. On the other hand this category seems also to be very organizational-oriented (either intra-company or virtual organization).
- Category 3 is explicitly addressing Service-Oriented paradigms - SaaS makes software accessible according to a service/utility model, but IT-Tude.com itself states clearly that further advances in research are still required: "...when resources are not required, they are not paid for, since computing power is purchased on a utility basis with operators only paying for the power that they use. This is a highly technically complex model to build and there are currently no SaaS services that underpinned by a true Grid hardware platform. However this is seen as the next stage of development in the SaaS market".

Taxonomy of Business Models: A further step is taken by the paper "A Taxonomy of Grid Business Models" [35] which analyses existing business models and on the basis of the result of the analysis: it formally defines a taxonomy of existing and future roles that a stakeholder can take on within the value chains of the Grid and gives examples of those roles. Finally, the paper applies the taxonomy to two reference business models: utility computing and software-as-a-service.

Their analysis takes into account the role of Hardware Resources Service Providers, as "*the lowest layer of the classification representing hardware providers. The hardware can belong to many different providers*". These include: Storage Resource Providers, Computing Resources Providers, Network Services Providers, Devices Service Providers.

In both of the business cases analyzed, Resource providers receive a benefit from the usage of their resources by end-users applications. Jobs monitoring is mandatory in order to quantify such usage and create a billing system.

5.1.2 COIN Integrated Project

COIN is an Integrating Project of ICT FP7 (www.coin-ip.eu), whose mission is to study, design, develop and prototype an open, self-adaptive, generic ICT integrated solution to support the 2020 vision "*by 2020 enterprise collaboration and interoperability services will become an invisible, pervasive and self-adaptive knowledge and business utility at disposal of the European networked enterprises from any industrial sector and domain in order to rapidly set-up, efficiently manage and effectively operate different forms of business collaborations, from the most traditionally supply chains to the most advanced and dynamic business ecosystems*", starting from notable existing research results in the field of Enterprise Interoperability and Enterprise Collaboration.

Software as a Service and Interoperability Service Utility Business Models: the research activity concerned with business models is driven by and executed with a business perspective in mind, aiming at answering a fundamental question: "*What is the value proposition for Enterprise Interoperability / Enterprise Collaboration in the forthcoming decade?"* Answering this question is a critical step for developing and ascertaining the (potential) business models for SaaS-U. A deliverable with first results is currently released as a public interim draft with an expected final release date in March 2010. Some insights of this interim draft that are of interest to the discussion of potential business models for the SOA4All Runtime are summarised below.

The business models research of COIN is concerned with business models for the Interoperability Service Utility (ISU) infrastructure in general, and SaaS-Us as specific business models leveraging the ISU capability with the focus on the combination of EI and EC utility and value added services for delivering value in targeted market segments.

We can clearly find several similarities between COIN Service Utility and SaaS Platforms and SOA4All infrastructure in terms of both functionalities offered (search, discovery, orchestration) and the way it is distributed (i.e. distributed/federated approach). We may then assume that business models that will be analyzed and defined within COIN could be relevant also for SOA4All infrastructure providers. Further collaboration activities, aiming at identifying synergies should be performed within the context of SOA4All WP10.

5.1.3 Google

If we consider SOA4All promised capability to discover “billions of services” on the Web, we cannot avoid taking a quick look at Google: the most popular search engine of the Internet represents a very interesting case in terms of business models.

It is easy to see how Google is sharing a number of common features and issues with SOA4All:

- Scalability: ability to easily grow at marginal costs, ability to adapt its size to high load and volumes, ability to monetize millions of users.
- Openness: content and services must be open and interoperable
- Co-creation: non-traditional actors become active part of the chain (Web 2.0 principles); users, content creators and external developers are given the tools to create new markets and enrich services
- Network effects: the reach of a critical mass of users constitutes a significant barrier

It is well recognised how Google’s main revenue source is advertising, like with AdSense and AdWords models (www.google.com/adsense; www.google.com/adwords):

  - “Google makes billions of dollars in revenue each fiscal quarter. Advertisers are Google's customer” [36].
  - “Google Business model: advertising is not a market but a business model; any market that attracts advertising is a target for Google”.[2]

While we may investigate how the Google business model could apply to the SOA4All front-end, a significant difference is constituted by the way Google deals with its resources: http://en.wikipedia.org/wiki/Google_platform#Current_hardware. While technically highly distributed, the Google infrastructure is totally owned or controlled by Google. On the one hand, this represents an initial difference versus SOA4All, where the infrastructure can be open to any provider; on the other, Google business model is also relying on advertising directly published on third party sites, like in the case of “AdSense”. These similarities and differences should be monitored and further analyzed within the context of SOA4All WP10.

5.1.4 Telco 2.0 Platform Provider

An infrastructure provider and associated business model which is under consideration within the SOA4All project is the Telco 2.0 Platform Provider. The platform is an essential part of

[2] http://www.slideshare.net/misteroo/all-about-google-presentation

the Telco 2.0 approach.

The Telco 2.0 initiative[3] introduces the notion that telecoms should use the opportunity provided by their position in the value chain in order to develop new "2-sided" business models. 2-sided business models exist where an organization is able to extract value from 2 sides of a value chain. In the telecommunications context, one revenue 'side' consists of essentially traditional revenues from core telco services such as voice and messaging; the other revenue side is made possible by the telco's position as a platform provider. This second revenue stream is derived from offering platform services to other businesses who then build on those services to offer services and applications to their own customers. An important aspect of this second 'side' is the leveraging of the telco's customer relationship to add value to the offerings of the upstream customer's offerings (e.g. by utilizing customer data to provide better targeting of adverts).

This upstream platform 'play' can be broadly divided into B2B value-added services (VAS) platforms and distribution platforms (as seen in Figure 11; Source: www.telco2.net).

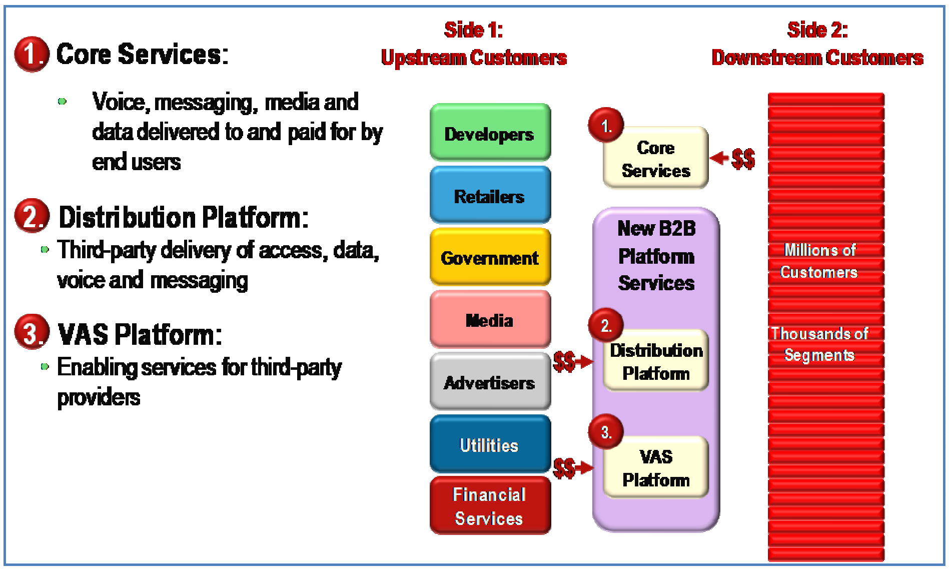
1. Core Services:
Voice, messaging, media and data delivered to and paid for by end users
2. Distribution Platform:
Third-party delivery of access, data, voice and messaging
3. VAS Platform:
Enabling services for third-party providers
Side 1: Upstream Customers
Side 2: Downstream Customers
Developers
Retailers
Government
Media
Advertisers
Utilities
Financial Services
1. Core Services
$$
New B2B Platform Services
2. Distribution Platform
3. VAS Platform
$$
$$
Millions of Customers
Thousands of Segments

Figure 11: Two-sided business model framework

Creating value-added services will enable third party organizations in multiple vertical sectors to become much more effective and efficient in their everyday interactions and business processes by allowing them to access and tailor services to their own specific needs and/or to build novel applications and services to meet the particular needs of their own customer base in a fast-changing technological environment.

Thus the platform provider is a key player in the Telco 2.0 approach. A service provider relies on the existence of a platform to leverage the relationship with its downstream customers to offer value added services to its upstream customers. The service provider may choose to operate its own platform infrastructure or choose to use that of another organization. A further possibility is that platform providers share their infrastructure with other platform providers, hosting services on behalf of them and vice versa.

Within the telecoms sector Service Delivery Platforms have emerged as a way to support a more personalized and instantaneous relationship between service providers and

[3] http://www.telco2.net/

consumers. Typically they provide a service control environment, a service creation environment, a service orchestration and execution environment, and abstractions for media control, presence/location, integration, and other low-level communications capabilities. Currently SDPs within telcos are optimized to handle their own specific services and architectures. However, there are moves to transform SDPs into frameworks that support the externalization of capabilities to third parties to build profitable services and to enable efficient service composition [37], that allow telcos to monetize the customer profile data and to stimulate service innovation by enabling mashups with Internet/Web 2.0 services and to support the B2B platform of Telco 2.0 approach allowing 2-sided business models. To support this there are standardization efforts in place to promote interoperability for such frameworks.

The DSB (distributed service bus) of the SOA4All project offers a large scale, open, distributed environment which can support the service platform required by the Telco 2.0 model. It is able to act as a shared repository and communication infrastructure and can host the SOA4All Platform Services that are made available to users via the SOA4All studio. Key requirements for a Telco 2.0 platform are scalability (i.e. able to support millions of simultaneous users and services), resilience and proximity to users (to reduce latency). Satisfying these requirements can improve the Quality of Service offered which is a key differentiator for telcos (compared to say webcos without the same level of infrastructure). Latency is a key issue in the telecoms domain due to the real-time requirements of services. Existing Service Delivery Platforms have been built to support such services and this capability has to be maintained in a platform supported by the DSB.

Clearly the reliance of the Telco 2.0 approach on the platform and the realization that SDPs must evolve to enable this creates an opportunity for SOA4All technology to support the additional requirements. A telco adopting the Telco 2.0 approach and acting as a platform provider can combine its Service Delivery Platform supporting its services with the features of a Distributed Service Bus. As such, using the categorization provided above, it is acting as a SOA4All hardware infrastructure provider or Platform-as a-Service provider.

5.2 Scenario-specific Configurations

The goal of this section is to define concrete but abstractly specified usage scenarios and to show how the Distributed Service Bus (DSB) and its particular features are needed and why. As a result, guidelines on infrastructure depending on scenario requirements will be provided so that each party which is interested in the SOA4All infrastructure layer can find the best way to use it.

5.2.1 Summary of SOA4All DSB Features

The SOA4All bus architecture is based on dynamically composed software components and can be adapted to fit everyone needs. A standard bus runtime contains only modules which are required to build a network of nodes which will provide the Distributed Service Bus. Depending on needs, the standard runtime can then be enriched by software components to provide customized runtimes. The software components and features are listed and described below:

- Optional modules :
    - SOAP platform services support. A software component is dedicated to provide SOAP based platform services binding to the DSB.
    - REST platform services support. A software component is dedicated to provide REST based platform services binding to the DSB.
    - SOAP third party services support. A software component is dedicated to provide SOAP based third party services binding to the DSB.

- REST third party services support. A software component is dedicated to provide REST based third party services binding to the DSB.
- Advanced monitoring feature based on WSDM. OASIS WSDM standard is used by the DSB monitoring layer to provide monitoring information to subscribers.
- Federation-level DSB transport support. A software component is dedicated to provide the capability to send/receive DSB messages whose destination/origin is one DSB node not belonging to the same DSB, but, to one of the DSBs it is linked to through the federation.

- Mandatory modules :
  - Management API. This API is used by SOA4All nodes managers to bind platform services to the DSB. A future version will also provide some “super-manager” API level. For example, a super-manager will have the capability to authorize nodes to connect to the core DSB network i.e. an unknown node will be rejected from the network.
  - Inter DSB node message transporter based on Internet compliant standards : HTTP, firewall friendly. This module is mandatory in order to provide the distributed service bus feature.
  - Load balancing/failover: The platform services can be replicated on multiple nodes if needed. The routing and endpoint choice strategy is the role of the DSB node consuming such a service. The DSB node consumer routing and the transport modules work together to finally access an endpoint which fits the requested QoS needs.

5.2.2 SOA4All Infrastructure Providers

SOA4All infrastructure providers can be defined as actors which will provide hardware, network or software resources to run the SOA4All DSB software. These providers can be classified in several groups depending on their interest and on their business model. The following sections will try to give an exhaustive list of providers’ types.

Hardware providers

Hardware providers will offer hardware resources on which SOA4All software will run. Since the nodes of a given DSB communicate freely using IP connections, all required ports must have been opened. However, in general, we can assume that one DSB is deployed upon a set of hardware resources provided by a same hardware provider, so, these ports opening requirements may not be a big problem, as the hardware resources may belong to a same network area, and so, be protected from uncontrolled access from the Internet by specific firewalls and NAT mechanisms. In case resources to host a given DSB are not provided by a same hardware provider, these hardware resources must be reachable from outside of each provider network area. This will only be possible by opening required network ports on the corresponding organizations firewalls.

Once a given DSB is deployed, the only requirement, in order for it to join a federation of DSBs is to select one specific resource, to act as a direct (or indirect, see below) gateway to/from the public network area, and configure it so that it can communicate with the other components of the federation (each component represents one DSB part of the federation). The only requirement for a given hardware resource to act as a peer of the federation layer, is to be capable to send and receive messages based upon the ProActive underlying communication substrate. This is not very constraining however, as ProActive communications can be handled by relying on only the http port, or better, on ssh-based connections, a mechanism that has proven to be secure because authentication is enforced. Even if the selected specific resource lies behind a firewall or belongs to a NAT, ProActive is able to route messages to/from it as soon as one machine of the network domain enables Internet access: this machine exchange messages with the specific resource through ssh tunnels, easily set up by configuring the ProActive runtime appropriately (see [38]).

As a hardware provider, there are two installation alternatives driven by security concerns. The first one is a total access to the machine. In this case, the SOA4All community will be able to deploy, install and configure the SOA4All DSB node directly. The second alternative, which will be potentially the most used, is the installation by the provider itself. In this case, the SOA4All consortium must provide a simple and automated DSB node installer (see a sketch of available deployment and installation facilities [2]) and the newly installed DSB must provide complete management API which is also accessible and exposed over the Internet as Web and REST services (see M&M API definition in [2]).

Software providers

The SOA4All software providers can be divided in three families. In all the cases, the software providers rely upon hardware providers (which might prove to be the same in practice!) since they need to have access to hardware resources to run the software on. Please refer to the previous section about hardware providers for more details.

1. DSB software providers: The main goal is to increase the number of DSB nodes. It is really important to keep a control over nodes which want to join the core DSB network, i.e. we cannot accept that someone who is installing a DSB node on its server will be connected to the Core DSB network without any control. By increasing the number of nodes connected to the core SOA4All node network, providers will give the opportunity to :
    a. Increase the number of access points. It means that a SOA4All consumer will potentially have the choice between several access points to the SOA4All network, and may prefer to chose the geographically closer access point
    b. Increase service availability by replicating some services. Adding a new node in the network can automatically replicate a selection of core platform services on the new node.
    c. Add the capability to bind platform service from a closer location. A best practice is always to bind a platform service on a node which is physically the closest one to reduce message transport delays and length of the communication path between the DSB node and the final service. The message transport on this physical link is not under control of the DSB itself, i.e. it may be unreliable, so the shorter it is, the better; on the contrary to DSB-level communication between DSB nodes can be considered as reliable enough and are under total DSB control.
2. Service providers: This type of provider can be described as an actor which has developed a set of services and who wants to provide them to the SOA4All core community (the SOA4All DSB developers and managers). This actor is not interested by the DSB side but only by binding its services to a DSB node. Several solutions are possible to bind the services of this provider :
    a. The provider sends a complete description of the service to the SOA4All managers. It is up to the managers to bind the service if they think that the service is mature enough and can really provide some added value to the core platform services.
    b. The provider uses a “software provider management API” to bind its service to the DSB. By using this API, the service becomes available and reachable from all the SOA4All consumers (modulo restrictions if the API provides a way to define them). The newly bound service is placed in what can be called a “service sandbox”. Once the service has been validated by the SOA4All managers and potentially by the SOA4All users (based on a rating feature), it can be moved from the “service sandbox” to the “community service” space.

3. DSB and service providers: A merge between the two previous software providers' views. It is the easiest way to enrich the DSB network and the core platform services. This type of providers can be defined as "premium partners" for whom all rights are granted. It means that the SOA4All consortium allows such a provider to easily provide a new set of services to all the SOA4All service consumers. This is only possible if the provider has installed one (or more) DSB node instance on his side and has bound its services to it (this DSB being part of the federation if any). The SOA4All management API needs to be invoked (invoker to be defined) to add this new node to the DSB. As a result, all the services which are bound to this node become available and accessible to all the SOA4All service consumers.

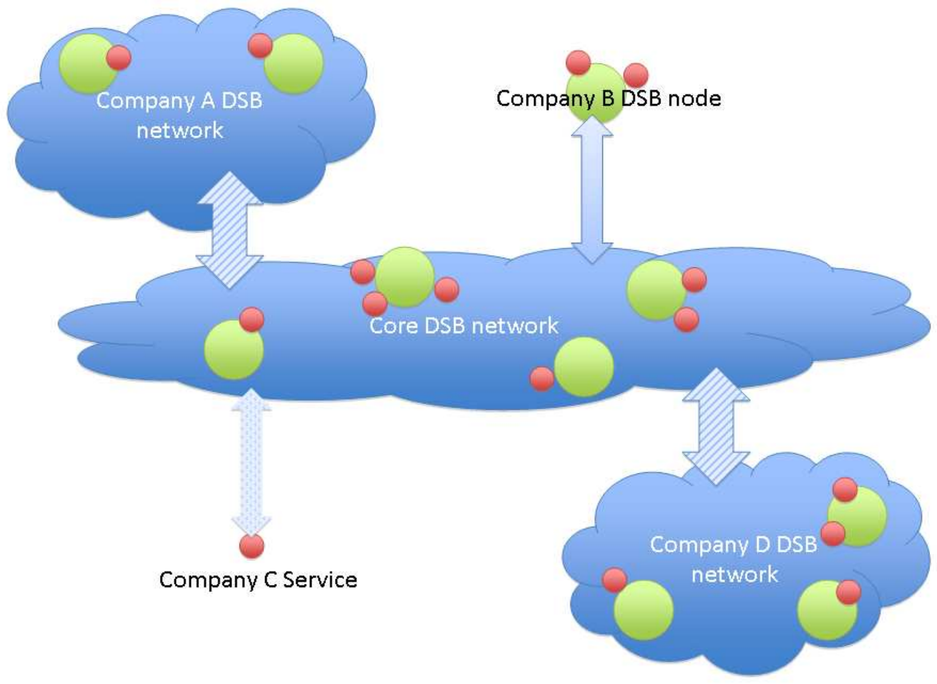

Figure 12: Different Provider roles in building the SOA4All services community

Building the Federated Distributed Service Bus with Providers

As a result, connecting all these provider types to the "Core DSB Network" is leading to a federation of DSB nodes: the Federated Distributed Service Bus aka fDSB. A summary view is available in Figure 12, and detailed below:

1. The "Core DSB Network" can be considered as the base infrastructure, nowadays built by the SOA4All project partners acting as hardware resource and software providers. This network of DSB nodes contains all the features and platform services needed to provide the minimal SOA4All infrastructure.
2. The companies A and D are connected to the Core DSB network by federation links (dashed big arrows on the figure). These companies both host internally a DSB network. The sum of these company networks and the Core DSB network complies to what is called the federation architecture pattern.
3. The company B hosts a single DSB node which is exposed and connected to the Core DSB network (meaning that this node had to open all needed ports to be part of the Core DSB network).

4. The company C provides a single service. This service is connected to the Core DSB network with the help of a binding component hosted by the Core DSB network (binding is very loose, compared to e.g., B being part of the Core DSB network, and as a consequence is depicted by a dotted arrow in the figure) .

It is important to understand that this DSB we name “Core DSB network” is behaving exactly the same as any other DSB (e.g. Company A in the figure) and plays the same role in the federated architecture of DSBs as any other. One major reason of its apparently specific status is that it has been deployed first, as a bootstrapping element for creating a bigger community, i.e. a federation. In parallel, imagine that other companies (e.g. A, D) have also decided to deploy their own SOA4All DSB. Then, knowing the address of the entry point to the Core DSB network, they decided to create a partnership relation with the Core, so joining the federation as illustrated in the figure.

The “Core DSB network” nodes are deployed on SOA4All partners resources for the time being, but in the future and as a successful sustainability result of SOA4All, we hope that the core be deployed on resources provided by one or several hardware providers, external to SOA4All, but wishing to set up collectively what is named a “Service Park” [12].

5.2.3 Use cases

There are many possible use cases scenarios of such flexible federation architecture. We devise a few of them below.

SOA4All Islands: Business or governmental entities, or spontaneous emerging user communities may decide to maintain one DSB through which access is granted to a bunch of thematic related services (i.e. in the telco domain, or banking, leisure, etc). Interest of relying upon a DSB is for obvious reasons as non-functional associated properties like monitoring, logging, etc. but also for more end-user oriented ones (i.e. accessing to a third-party service through the DSB lets the consumer gains some benefits, i.e. some miles or a cost reduction in case the service is not free, etc, and if he is a provider, lets him benefit from some advertisement because the service is deployed or at minimum proxy-fied onto the DSB). On exchange, the DSB hosting community gets some information about prosumer profiles, community size, and could have as goal to become a visible, popular, well-ranked service ecosystem.

Federation of SOA4All islands: Even with the presence of such SOA4All islands, it becomes possible to build up a federation along the same kind of architecture described in the above figure between Companies “A”, “D”, and “Core”.[4] Technically, the federation would be set up by using tools presented in Section 3.1. The main motivation would be to enlarge, extend the offered set of service thematic in a meaningful, coherent manner (e.g. as for a travel agency service infrastructure that federates several but complementing thematic services offered by flight companies, airports, car renting, hotel, sporting and leisure activities, etc). Here we not only federate services (this is a “simple” DSB duty), but we enable to federate several service marketplaces. Within the federation, SOA4All islands can be bound together, either in an all-to-all manner, or, not. Indeed, it might happen that island A subcontracts the execution of some required services to island B for any commercial reason, and not to island B’, because B and B’ even if both part of the federation act as competitors on the open market (i.e. B offers telcos services as SMS, conferencing, etc, as B’, but A has agreed to use only B). So in this case DSB A will never have to send messages to DSB B’ directly, so no need to have A be bound to B’. However if island A is partnering with island C, and island C has also partnered with island B’ in the context of setting up this service park, this still allows a client of DSB A to benefit, but indirectly, of services published on DSB B’ through DSB C intermediary.

[4] Federation of SOA4All islands = a service park.

5.2.4 Scenario-specific configurations

In the sections above, several DSB usage scenarios have been introduced in order to show all the features and possible installations that are offered by the SOA4All DSB. To conclude this section, we provide a detailed approach of the procedures to follow to deploy, install, configure and bind services to the DSB under different usage scenarios.

In the following scenarios, it is assumed that:

1. A DSB installer is provided. This installer will download the required modules selected by the user: SOAP support, REST support, SCA support, proxy support, etc... In the scenarios description, the installation task will only be described as “Install the SOA4All DSB binary package with support of …”.
2. A DSB configuration tool/script is provided. This tool will allow the manager to set the DSB properties such as name, network configuration (IP address, network ports …) and is also used to connect the local DSB node to the ‘Core DSB Network’. This tool is launched after installation step described in previous point.
3. A management tool is available (SOA4All Studio). This management tool provides :
    a. The list of nodes available in what is called ‘the Core DSB Network’
    b. A DSB management API client to easily call management operations on DSB nodes

Scenario 1 – Provide a platform service

In this scenario, a service provider wants to expose a WSDL/REST service to the SOA4All community without deploying any SOA4All software:

1. Get the SOA4All DSB node reference you want the service to be bound to.
2. Call the bindWSDL/bindREST operation of the ServiceBinder service of the DSB management API like bindWSDL(wsdlURI) where wsdlURI is the WSDL URI of the platform service or bindREST(restURI) where restURI is the REST service base URI.

Once the service is bound to the DSB node, it is automatically exposed by other nodes of the federation and so accessible to all external service consumers.

Scenario 2 – Install a DSB node

In this scenario, a provider wants to deploy a DSB node to join an existing DSB federation. The provider does not want to bind additional platform services and will only act as a new DSB access point. In order to act as a new access point, the DSB node must provide at least the SOAP and REST support.

1. Install the SOA4All DSB binary package with SOAP and REST support. This will install the DSB runtime plus the SOAP and REST JBI Binding Components. These components will expose the platform services which are deployed on other DSB nodes.
2. Configure the DSB node to join the ‘Core DSB Network’
3. Start the DSB node

Once the DSB node is started, it will detect the platform services which are already deployed/bound and will automatically expose them as SOAP/REST services. As a result, a new DSB access point is available and exposes all the other platform services to external service consumers.

Scenario 3 – Install a DSB node and provide platform services

This scenario is a mix of Scenario 1 ‘Provide a Platform service’ and of Scenario 2 ‘Install a DSB node’. The DSB platform provider has to follow the steps defined firstly in scenario 2 to install the DSB node and then he has to follow the Scenario 1 in order to bind and expose his platform services to the DSB network.

6. Conclusion

This deliverable was the second architecture documentation of SOA4All and provided an extended and more precise definition of the SOA4All architecture and in particular its components: service bus infrastructure and platform services.

An updated and extended specification of the distributed service bus infrastructure was given. The DSB federation architecture was introduced and the associated tools that enable the deployment of the federation, easy integration of PEtALS DSBs and management of the federation architecture was explain, in order to create partnerships among service providers in so-called service parks. Furthermore, extended support for invoking platform and third party services were added to the bus in order to allow for both RESTful services and traditional WS-* stack (WSDL/SOAP) services. Lastly, an updated specification of the monitoring infrastructure was given that supports both passive and active monitoring on SOAP and REST services. Passive monitoring, also known as real-user monitoring, refers to the approach that tracks the quality of services (QoS) as well as the end-user behaviours by capturing all the messages that go across the DSB as users invoke external services. Active monitoring, also known as synthetic monitoring, is to test the performance of services by triggering fake requests that simulate the actions of actual users.

In a second part, focus is set on specifying the platform services of their access interfaces. Some of the platform services are implemented as RESTful ones, while others are exposed via WSDL services. Moreover, the integration of the platform was exemplified first with a consolidated example of service location and construction, and then from a functional perspective by defining a precise specification of functional processes involving the needs for platform services but particularly also infrastructure. The aim of the functional processes section was to define what SOA4All does and offers in terms of functionality and to check whether the platform services offer what is expected from them in terms of interfaces to complete the functional processes at hand.

In a third and last part, this deliverable provided a quick excursion towards possible relevant business models for SOA4All infrastructure providers. An observable problem with the SOA4All approach was the focus on service providers and consumers only, and the negligence of the infrastructure provider role. This missing role is however crucial in particular in the scope of the service bus and hence the SOA4All Runtime realization. Although, this deliverable is not defining business models and exploitation plans, it is important to understand the different roles, application scenarios and the resulting platform needs and configurations. This last part thus outlines different usage scenarios from an infrastructure point of view, and how different runtime configurations allow for optimized infrastructure deployments that do not require heavy-weight service bus installations for all scenarios. Matching infrastructure requirements to bus configurations is important in keeping the infrastructural backbone as simple as possible.

Herewith, this second architecture deliverable concludes and in a next step the presented technicalities and tools will be implemented and integrated with the existing realization of the previous milestone. Furthermore, the SOA4All Runtime team will continue to provide the fundamental building blocks for the realization of an integrated SOA4All experience and the resulting Global Service Delivery Platform.

References

[1] D. Tapscott: The Digital Economy: Promise and Peril In The Age of Networked Intelligence, McGraw-Hill, 1997.

[2] R. Krummenacher, I. Toma, Ch. Hamerling, J.-P. Lorre, F. Baude, V. Legrand, Ph. Merle, C. Ruz, C. Pedrinaci, D. Liu and T. Pariente Lobo: SOA4All Reference Architecture Specification, SOA4All Project Deliverable D1.4.1A, March 2009.

[3] Ch. Hamerling, V. Legrand, F. Baude, E. Mathias, C. Ruz, M. Fried, R. Krummenacher, Ph. Merle and N. Dolet: SOA4All Runtime, SOA4All Project Deliverable D1.4.1B, September 2009.

[4] L. Nixon, E. Simperl, R. Krummenacher and F. Martin-Recuerda: Tuplespace-based computing for the Semantic Web: A survey of the state of the art, Knowledge Engineering Review 23(2), 2008, 181-212.

[5] R. Krummenacher, I. Filali, F. Huet and F. Baude: Distributed Semantic Spaces: A Scalable Approach to Coordination, SOA4All Project Deliverable D1.3.2A v1.1, August 2009.

[6] R. Krummenacher, M. Fried, F. Huet, I. Filali, L. Pellegrino and Ch. Hamerling: Distributed Semantic Spaces: A First Implementation, SOA4All Project Deliverable D1.3.2B, September 2009.

[7] R. Krummenacher, F. Huet, M. Fried, L. Pellegrino and B. Norton: A Distributed Semantic Marketplace, SOA4All Project Deliverable D1.3.3A, March 2010.

[8] T. Vitvar, J. Kopecky, J. Viskova, and D. Fensel: WSMO-Lite Annotations for Web services, 5th European Semantic Web Conference, June 2008: 674-689.

[9] J. Farrell and H. Lausen, Semantic Annotations for WSDL and XML Schema (SAWSDL), W3C Recommendation, August 2007.

[10] M. Maleshkova, J. Kopecky and C. Pedrinaci, Adapting SAWSDL for Semantic Annotations of RESTful Services, Beyond SAWSDL Workshop at OnTheMove Federated Conferences & Workshops, November 2009: 917-926.

[11] F. Baude, I. Filali, F. Huet, V. Legrand, E. Mathias, Ph. Merle, C. Ruz, R. Krummenacher, E. Simperl, Ch. Hamerling, and J.-P. Lorre: ESB Federation for Large-Scale SOA, 25th ACM Symposium On Applied Computing, 2010.

[12] C. Petrie and C. Bussler: The Myth of Open Web Services: The Rise of the Service Parks, IEEE Internet Computing 12(3), 2008: 96-95.

[13] E. Al-Masri and Q.H. Mahmoud: QoS-based Discovery and Ranking of Web Services, IEEE 16th International Conference on Computer Communications and Networks, 2007: 529-534.

[14] N. Artaiam and T. Senivongse: Enhancing Service-Side QoS Monitoring for Web Services, 9th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008.

[15] E.M. Maximilien and M.P. Singh: A framework and ontology for dynamic Web services selection, IEEE Internet Computing 8(5), 2004: 84-93.

[16] E. Al-Masri and Q.H. Mahmoud: Web Service Discovery and Client Goals, Computer 42(1), 2009: 104-107.

[17] D.A. Menasce: QoS issues in Web services, IEEE Internet Computing 6(6), 2002: 72-75.

[18] T. Vitvar, J. Kopecky, J. Viskova and D. Fensel: WSMO-Lite Annotations for Web Services, 5th European Semantic Web Conference, 2008: 674-689.

[19] J. Kopecky, K. Gomadam and T. Vitvar: hRESTS: An HTML Microformat for Describing RESTful Web Services. IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2008: 619-625.

[20] Apache Axis2/Java - Next Generation Web Services: http://ws.apache.org/axis2/

[21] A. Mos, C. Pedrinaci, G. Alvaro Rey, J. Manuel Gomez, D. Liu, G. Vaudaux-Ruth, and S. Quaireau: Multi-Level Monitoring and Analysis of Web-Scale Service Based Applications, 2nd Workshop on Monitoring, Adaptation and Beyond at ICSOC/ServiceWave, 2009.

[22] N. Steinmetz, H. Lausen and M. Brunner, 7th Web Service Search on Large Scale, International Conference on Service Oriented Computing, November 2009: 437-444.

[23] C. Pedrinaci, J. Domingue and R. Krummenacher, Services and the Web of Data: An Unexploited Symbiosis, AAAI Spring Symposia, March 2010.

[24] S. Agarwal, M. Junghans, O. Fabre, I. Toma and J.-P. Lorre: First Service Delivery Prototype, SOA4All Deliverable D5.3.1, September 2009.

[25] J. de Bruijn and H. Lausen and A. Polleres and D. Fensel, The Web Service Modeling Language WSML: An Overview, 3rd European Semantic Web Conference, June 2006: 590-604.

[26] I. Toma, D. Roman, D. Fensel, B. Sapkota and J.M. Gomez, A Multicriteria Service Ranking Approach Based on Non-Functional Properties Rules Evaluation, 5th International Conference on Service-Oriented Computing, November 2007: 435–441.

[27] I. Toma, N. Steinmetz, H. Lausen, S. Agarwal and M. Junghans: First Service Ranking Prototype, SOA4All Project Deliverable D5.4.1, September 2009.

[28] F. Schnabel, L. Xu, Y. Gorronogoitia, M. Radzimski, F. Lecue, G. Ripa, S. Abels, S. Blood, M. Junghans, A. Mos and N. Mehandjiev, Advanced Specification Of Lightweight, Context-aware Process Modelling Language, SOA4All Project Deliverable, September 2009.

[29] F. Lécué, Optimizing QoS-Aware Semantic Web Service Composition, International Semantic Web Conference, October 2009: 375-391.

[30] G. Greco, A. Guzzo, L. Pontieri and D. Saccà, Mining Expressive Process Models by Clustering Workflow Traces, 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, May 2004: 52-62.

[31] A. Simov: WSMO Data Grounding Component, SOA4All Project Deliverable D3.4.4, September 2008.

[32] F. Fischer and B. Bishop: Framework and API for Integrated Reasoning Support, SOA4All Project Deliverable D3.2.1, August 2008.

[33] M. Maleshkova, C. Pedrinaci and J. Domingue, Supporting the Creation of Semantic RESTful Service Descriptions, Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web at 8th International Semantic Web Conference, October 2009.

[34] Y. Gorroñogoitia, M. Villa, F. Lecue, M. Radzimski and G. di Matteo: Advanced Prototype For Service Composition and Adaptation Environment, SOA4All Project Deliverable D6.4.2, March 2010.

[35] J. Altmann, M. Ion, A. Adel and B. Mohammed: A Taxonomy of Grid Business Models, International Workshop on Grid Economics and Business Models, 2007: 29-43.

[36] M. Elgan: Google's Business Model: YOU Are the Product, on EarthWeb.com, February 2009.

[37] K. Kimbler: Evolving Service Delivery Platforms: Essential Plumbing For Smart Pipes, tmforum Insights Report, 2009.

[38] B. Amedro, F. Baude, D. Caromel, C. Delbé, I. Filali, F. Huet, E. Mathias and O. Smirnov: “An Efficient Framework for Running Applications on Clusters, Grids and Cloud” in Cloud Computing: Principles, Systems & Applications, Springer Verlag, to appear.