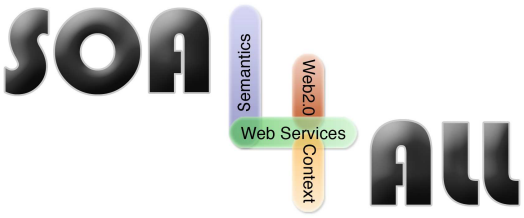

Project Number: **215219**
Project Acronym: **SOA4All**
Project Title: **Service Oriented Architectures for All**
Instrument: **Integrated Project**
Thematic Priority: **Information and Communication Technologies**

D6.5.1. Specification and first prototype of the composition framework

Activity:	Activity 2 - Core Research and Development Activities	
Work Package:	WP6 - Service Construction	
Due Date:		M12
Submission Date:		13/03/2009
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		CEFRIEL
Revision:		1.0
Authors:	Gianluca Ripa CEFRIEL Maurilio Zuccalà CEFRIEL Adrian Mos INRIA	
Reviewers:	Virginie Legrand INRIA Jesus Gorronogoitia ATOS	

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

Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	15/12/2008	TOC defined	Gianluca Ripa (CEFRIEL)
0.2	16/01/2009	First version	Gianluca Ripa, Maurilio Zuccalà (CEFRIEL)
0.3	23/01/2009	TOC modified	Maurilio Zuccalà (CEFRIEL)
0.4	31/01/2009	Revised content	Gianluca Ripa (CEFRIEL)
0.5	06/02/2009	Added section 3.3	Adrian Mos (INRIA)
1.0	09/02/2009	General revision	Gianluca Ripa (CEFRIEL)
1.01	20/02/2009	Internal Review	Virginie Legrand (INRIA)
1.02	04/03/209	Internal Review	Jesus Gorronogoitia (ATOS)
1.1	06/03/2009	Final version after internal review	Gianluca Ripa (CEFRIEL)
Final	13/03/2009	Overall format and quality revision	Malena Donato (ATOS)

Table of Contents

EXECUTIVE SUMMARY 7

1. INTRODUCTION 8

1.1 PURPOSE AND SCOPE OF THIS DELIVERABLE 8
1.2 STRUCTURE OF THE DOCUMENT 8

2. REQUIREMENTS FOR THE EXECUTION ENGINE FOR LIGHTWEIGHT PROCESSES 9

2.1 FUNCTIONAL REQUIREMENTS 9
2.2 NON FUNCTIONAL REQUIREMENTS 9
2.3 TECHNOLOGICAL AND SYSTEM REQUIREMENTS 10
2.4 SUMMARY OF REQUIREMENTS 11

3. SPECIFICATION OF THE EXECUTION INFRASTRUCTURE FOR ADAPTABLE SERVICE COMPOSITIONS 14

3.1 BASELINE FOR DEVELOPEMENT 14
3.1.1 Service Composition Execution Environment (SCENE) *14*
3.1.2 Semantic BPEL Execution Engine (SBPELEE) *16*
3.2 EXECUTION ENGINE DEVELOPMENT ROADMAP 16
3.3 A NEW APPROACH FOR ADAPTING SERVICE REQUESTS TO ACTUAL SERVICE INTERFACES THROUGH SEMANTIC ANNOTATIONS 17
3.3.1 The approach used in SCENE for adapting service requests to actual service interfaces *18*
3.3.2 The approach extended through the use of semantic annotations *19*
3.4 GENERATION OF RUNTIME ARTEFACTS 23
3.5 INTEGRATION IN SOA4ALL 24
3.5.1 Execution Engine Main Components Description *26*
3.6 SUMMARY OF FULFILLED REQUIREMENTS 27

4. CONCLUSIONS 29

5. REFERENCES 30

ANNEX A. COMPARISON BETWEEN SOA4ALL, SECSE AND SUPER APPROACHES 32

A.1. DIFFERENCES AND SIMILARITIES BETWEEN SOA4ALL EXECUTION ENGINE AND SBPELEE 32

A.2. DIFFERENCES AND SIMILARITIES BETWEEN SOA4ALL EXECUTION ENGINE AND SECSE 33

ANNEX B. THEWEATHER TO FORECAST MAPPING SCHEMA 35

ANNEX C. SAWSDLS SERVICE DESCRIPTION 38

C.1. THEWEATHER SERVICE SAWSDL DESCRIPTION SPECIFICATION 38

C.2. FORECAST SERVICE SAWSDL DESCRIPTION SPECIFICATION 41

ANNEX D. LIFTING SCHEMA MAPPING 44

ANNEX E. SAMPLE ONTOLOGIES 45

E.1. SAMPLE GPS ONTOLOGY 45

E.2. SAMPLE TIME ONTOLOGY 45

E.3. SAMPLE WEATHER FORECAST ONTOLOGY 45

Glossary of Acronyms

Acronym	Definition
BPEL	Business Process Executable Language
BPM	Business Process Management
BPML	Business Process Modelling Language
BPMN	Business Process Modelling Notation
BPMS	Business Process Management System
D	Deliverable
EC	European Commission
JBI	Java Business Integration
QoS	Quality of Service
SCA	Service Component Architecture
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SAWSDL	Semantic Annotated WSDL
T	Task
WP	Work Package
WSCI	Web Service Choreography Interface
WSDL	Web Service Description Language
XPDL	XML Process Definition Language

List of Figures

Figure 1 – Rule that establishes the binding to the most reliable service 14
Figure 2 – Architecture of SCENE 15
Figure 3 - Intermediating SOA Development Spaces 24
Figure 4 Service Construction Framework overall picture. 25

List of Tables

Table 1 – Summary of requirements.....13
Table 2 – Software development roadmap 17
Table 3- Interface of TheWeather service.....20

Table 4 -.Interface of the Forecast service.....20
Table 5 – Summary of fulfilled requirements (**-** no, **O** partially, **X** yes).....28
Table 6 Comparative analysis Super-IP vs SOA4All Execution Engine. 32
Table 7 Comparative analysis Super-IP vs SOA4All Execution Engine. 33

Executive summary

In the definition of the Execution Engine for Lightweight Processes, we have started from the requirements coming from the SOA4All use cases, from our experience and from the state of the art analysis.

A selection of most relevant requirements coming out from the SOA4All use cases are:

- supporting manual and automatic adaptation of specific services or entire processes to different consumption settings;
- enabling dynamic selection and composition services;
- supporting third party services freely available on the Internet as well as enterprise services;
- taking into account the unavailability and the possible faults of third party services;
- allowing non-technical users to compose services.

Some of the requirements listed above have been addressed in other projects before, see for instance SUPER [13], focusing on BPM, and SeCSE [12], focusing on adaptable service compositions, however no project has considered all of them as a whole. Furthermore the use of the solutions proposed by these projects requires high expertise in both business and IT while SOA4All has to enable non technical users for building and executing processes and service compositions.

The SOA4All Execution Engine is built on top of SCENE [8] that represents the baseline for development. One of the planned extensions is to evaluate the integration of SCENE with part of the SBPELEE engine [6] developed in SUPER.

In order to fulfill the SOA4All requirements some developments were realized and others are planned, they are:

- implementing new approaches that aim at simplifying and partially automating the work of a potential user that has to design and execute a lightweight process.
- defining and implementing an approach for the generation of the executable artifacts needed by the SCENE platform starting from the more abstract artifacts produced by the users (as described in [1]) and optimized by our adaptation framework (as described in [2]).
- integrating the Execution Engine in the SOA4All runtime (as described in [3]) for exploiting functionalities offered by other components of the SOA4All architecture.

We will conduct an experiment on the e-government scenario developed in the context of the WP7 of SOA4All.

1. Introduction

1.1 Purpose and scope of this deliverable

This deliverable contains the requirement specification and the design draft of the Execution Engine for Lightweight Processes as well as the description of the implementation of a first running prototype with basic functionality. The design draft is based on the state-of-the-art report, use case requirements as well as conceptual considerations.

The Execution Engine will exploit a set of basic mechanisms such as dynamic discovery, selection, adaptation, invocation, mediation, monitoring developed in the other work packages, for supporting the dynamic and adaptive reconfiguration in reaction to environmental changes. In the definition of the execution framework, we have started from the requirements coming from the SOA4All use cases and from the state of the art analysis.

From SOA4All use cases, and from our experience, we know that the high dynamics of businesses and the continuous evolution of end users expectations and needs lead to developing systems able to self-adapt to various kinds of changes, depending on:

- the specific capabilities of components services actually available,
- the environmental conditions in which the system is being executed, and
- the possible faults and unavailability of components services.

For enabling the runtime self reconfiguration of a service composition a set of constraint and rules has to be defined before execution. Since SOA4All targets non-technical users we are developing new approaches for allowing non–technical users to take advantage of the self-reconfiguration capabilities of a service composition. These approaches will exploit:

- the semantic description of the services;
- the classification of the contextual information and of the possible faults and unavailability of components services that can happen during execution.

1.2 Structure of the document

This deliverable is organised as follows. Section 1 gives an introduction to the scope and content of this deliverable. Section 2 summarizes the requirements for the *adaptive service compositions execution framework*. In Section 3 we present the specification of the execution infrastructure for adaptable service compositions and a new approach for allowing non–technical users to take advantage of the self reconfiguration capabilities of a service composition. Finally in section 4 we draw some conclusion.

A short paper is included as confidential Annex to this paper.

Other annexes to this document are:

- ANNEX A: Comparison between SOA4All, SeCSE and SUPER approaches;
- ANNEX B: theweather to forecast mapping schema;
- ANNEX C: sawsdls service description;
- ANNEX D: lifting schema mapping.

2. Requirements for the Execution Engine for Lightweight Processes

The requirements for the Execution Engine for Lightweight Processes result out of the SOA4All use cases as well as from the experiences from previous projects. In this section, we refer to an example already developed in the SeCSE project [12] and refined to meet the expectations of the SOA4All use cases. We do not refer directly to SOA4All use cases scenarios, as example, because they are defined in details in parallel to the writing of this document.

2.1 Functional Requirements

In order to better explain the functional requirements of our execution framework we refer to an example: a weather forecast service built to support car drivers exploiting a car navigation system in knowing the weather conditions at the destination place and at the estimated arrival time in a very user friendly way. The user must be only required to select the destination on the car navigation system and the weather forecast information is automatically displayed with the result of the trip planning.

In the example, the destination and the arrival time can only be known by the system at execution time. This is a common situation since many services depend on information known only during execution. Furthermore, from the experience, we know that, very often, the weather forecast services support only a certain geographical area or even a single country. Each of them can have a different granularity with respect to places (i.e. only big cities, all cities, by GPS coordinates,) and times (i.e. for a certain day of the week, for a certain hour of the day) of the forecast. In this hypothesis, we can only select the concrete service to invoke at execution time. **There is a need for runtime adaptation and dynamic binding mechanisms w.r.t.. user's context.**

Since the system will discover, select and bind the appropriate concrete service to invoke only at execution time, necessarily it will have a different interface or different protocol from those expected by the service requester. **The system must be able to adapt service requests to actual service interfaces at execution time**. This is also the situation of the WP7 scenario where, for instance for payment activities, depending on the consumers choices a completely different data should be transmitted to the payment broker.

In many situations, there is a probability to discover several candidate services able to return the weather forecasts for the same location and time. While these services could be considered equivalent for provided functionalities, they could provide very different results in term of :

- quality of service provided (e.g. availability, reliability, response time, reputation),
- underlying business models (from free-use to pay-per-invocation) and other non-functional characteristics (e.g. ability to negotiate or re-negotiate an SLA, ability to monitor the execution, requirements for security and identity management).

The execution infrastructure should support the service selection according to functional, non-functional and contextual information.

Almost every service can fail or malfunction in its execution for many reasons. Since we can rely on many candidate services, when a fault occurs the system must be able to be aware of this and when possible should try to recover the execution. **The execution infrastructure should provide fault handling and dynamic rebinding mechanisms.**

2.2 Non functional requirements

Within Web 2.0, active consumers (often referred as prosumers) become part of the content providing process and often in the form of a community of creators. One of the main objectives of SOA4All is to allow non-technical users to compose services, in a user friendly,

Web-2.0 way. Users should be able to discover, select and compose services publicly available on the Internet, not being faced to too high technological barriers. **The execution infrastructure should support service prosumers.**

In SOA4All, there is the assumption that in a near future, a suitably large collection of third party services will be deployed and will become available for discovery and consumption. In this context services can appear, be modified, or disappear in an ad hoc fashion. In this open world setting the idea of being able to self-manage is very important. As we already discussed in [4] the basic requisites for a self-manageable system are:

- being knowledge aware,
- ability to sense and analyze environmental conditions,
- ability to actuate on its environment.

The characteristics of autonomous systems are being applied today in four fundamental areas of self-management to drive significant operational improvements where traditional manual-based processes are neither efficient nor effective. These four areas are related to different attributes of autonomous systems, and they are self-configuring capabilities, self-healing capabilities, self-optimizing capabilities, and self-protecting capabilities.

For each of these main areas of applicability, a design principle can be extracted where it is believed it should be incorporated in SOA4All execution framework in order to leverage the construction, configuration and deployment of infrastructures that enable Web scale service-oriented environments:

1. ***Self-configuring*: The system should analyze monitoring information and react accordingly.** For instance, in the context of compositions of services when a service losts its quality, it should be replaced with another one.
2. ***Self-healing*: When a service is published and it becomes publicly available, it should announce its capacity and should be incorporated seamlessly.** The rest of the system should reconfigure itself to take advantage from the presence of the new service.
3. ***Self optimizing*: The system should be able to monitor itself and should be able to carry out actions to tune its resources**.
4. ***Self-protecting:* The runtime environment should be able to use information coming from other components of the architectures for anticipating problems and take steps to avoid or mitigate them.**

Another non-functional requirement of the SOA4All execution environment is **scalability**. Indeed Service-oriented computing at Internet scale raises scalability issues (e.g. performance management, replication and load balancing) not present in current intra-company solutions. In fact, the growing number of Web services and increased use of service infrastructure across business networks brings new challenges that are not addressed adequately today to master the very large and meeting the challenge of dealing with billions of services.

2.3 Technological and system requirements

SOA4All Runtime consists of an infrastructure that brings Web services and SOA to the Web scale. In particular, the SOA4All framework defines some facets from which result out a technological and a system requirement for the SOA4All service composition execution framework:

- A Web grounding for semantic descriptions.
- A Distributed Service Bus for accessing and co-ordinating services at Web scale.

WSMO will be an essential building block for the overall SOA4All architecture. As the project progresses, the current WSMO specification will be further developed and extended to enable further functionalities. WSMO Lite represents one of these extensions. WSMO Lite realizes the WSMO concepts in a lightweight manner using RDF/S and SAWSDL, established as parts of the Semantic Web and Web services language suite. **We assume that services involved in the composition are semantically annotated using the grounding schema described in [9].** This is related with the self-* characteristics of the execution environment since they are configured through the semantic descriptions of the component service available for use inside a service composition or inside a process. The execution environment relies on these descriptions for selecting services to invoke and for adapting requests and responses. A composition reconfigures itself by means of substituting a service implementation with one other.

If all the invocations go through the Distributed Service Bus (DSB), monitoring only needs to catch the messages sent to the services, otherwise the service composition execution engine will also have to notify which messages have been sent and received (this is different from the monitoring events it will have to send about the actual evolution of the process). Invoking through the DSB makes it simpler and more homogeneous.

2.4 Summary of requirements

In Table 1 the requirements for the execution infrastructure are summarized. The requirements are uniquely identified by a label for being properly referenced in the remainder of the document. The labels adopt the following syntax:

- functional requirements F plus number,
- non functional requirements NF plus number,
- system requirements S plus number.

N.	Requirement type	Requirement short description	Source of requirement	Explanation
F1	Functional	**Dynamicity** There is the need for runtime adaptation and dynamic binding mechanisms	Use cases	All our use cases require the support for manually and automatically adapting specific services or entire processes to different consumption settings.
F1.1	Functional	**Adaptation** The system must be able to adapt service requests to actual service interfaces at execution time	Use cases	All our use cases aim at allowing dynamic selection and composition of third party services.
F1.2	Functional	**Dynamic Binding** The execution infrastructure should support the service	Use cases	All our use cases require the management of the Quality of Service (QoS) and Service

		selection according to functional, non-functional and contextual information		Level Agreements (SLAs) in composed services
F2	Functional	**Functional fault handling** The execution infrastructure should provide fault handling and rebinding mechanisms w.r.t. availability and faults	Use cases	All our use cases aim at supporting freely available services on the Internet as well as enterprise services. Third party services are inherently unreliable and, for this reason, the unavailability and the possible faults have to be taken into account.
NF1	Non functional	**User-friendly composition** Enabling service prosumers	SOA4All Vision	One of the main objective of SOA4All is to allow non-technical users to compose services.
S1	Non functional / System requirement	**Adaptation to contextual changes** The system should analyze monitoring information and react accordingly	SOA4All Vision and Architecture	In a world of billions of services it is necessary to deal with timely and accurate information about the execution of applications and infrastructural services in order to govern their execution supporting the adaptation to contextual changes (e.g. availability of services, variations in their execution time, etc)
S2	Non functional / System requirement	**System Self-configuration** When a service is published, it should announce its capacity and the rest of the system should reconfigure itself to take advantage from the presence of the new service. In particular, the system should support WSMO Lite annotations.	SOA4All Vision and Architecture	Semantic service annotation is essential for service discovery and service composition and must go beyond the technical description of services such as provided by WSDL. To this extent we utilize and extend the WSML family of

				languages for the purpose of semantically annotating services.
S3	Non functional / System requirement	**Self-optimization** The system should be able to monitor itself and should be able to carry out actions to tune its resources.	General requirement	In order to increase performances, and to reduce human effort in managing the system
S4	System requirement	**Soa4all Integration** The runtime environment should be able to use information coming from other components of the architecture for anticipating problems and take steps to avoid or mitigate them	SOA4All Architecture	The runtime environment is a component of the SOA4All architecture and should cooperate with all the other components.
S5	Non functional/ System requirement	**Scalability** Taking advantage from the SOA4All Distributed Service Bus for addressing the scalability scalability requirement and for acheving coherence of the architecture.	SOA4All Vision and Architecture	The DSB enables for accessing services at Web scale, by seeking appropriate distribution techniques that evolve the traditional ESB techniques towards the fully Distributed Service Bus; without a priori altering the communication and interaction approaches of ESB.

Table 1 – Summary of requirements

3. Specification of the execution infrastructure for adaptable service compositions

Some of the requirements listed in section 2 have been addressed in other projects before, see for instance SUPER [13], focusing on BPM, and SeCSE [12], focusing on adaptable service compositions), however no project has considered all of them as a whole. Furthermore the use of these solutions requires high expertise in both business and IT while SOA4All has to enable non technical users for building and executing service process and service compositions.

The SOA4All execution infrastructure for adaptable service compositions is built on top of SCENE [8] that represents the baseline for development. The following subsections describe SCENE and the extensions already realized and envisioned for the second year of the project. For one of these extensions it is planned to evaluate the integration of SCENE with part of the SBPELEE engine [6] developed in SUPER. The SBPELEE engine is briefly described in section 3.1.2.

3.1 Baseline for developement

3.1.1 Service Composition Execution Environment (SCENE)

3.1.1.1 The SCENE language

In SCENE a service centric system is defined in terms of two main aspects: the application-dependent control logic that is expressed using BPEL, and the policies that enable self-adaptation at runtime. These policies, among other things, allow designers to postpone the binding to specific component services until runtime. At design time the designer associates to a BPEL service invocation an abstract service, e.g., a generic weather forecast service in the example mentioned in section 2, and defines a binding rule that defines how to determine the binding to a concrete service. In our specific example, this rule states the selection of the service with the best actual reliability.

```
Event: bindingEvent
Condition: action=getForecast
Action: bind getForecast, criteria: best candidateServiceList.QoS.reliability
```

Figure 1 – Rule that establishes the binding to the most reliable service

3.1.1.2 The SCENE platform

The SCENE platform provides the runtime execution environment for compositions written in the SCENE language. As mentioned above, the SCENE language extends the standard BPEL language with rules that are used to guide the execution of self-adaptation operations at runtime. Figure 1 shows an example of a rule, written in pseudo-code, referred to the example of the weather forecast we have introduced before. The rule allows the runtime environment to bind to the concrete service implementation that has proved to have the best reliability.

The SCENE prototype includes the following components depicted in Figure 2:

- a BPEL engine, Active BPEL [17], which is in charge of executing the process part of the service composition;
- an open source rule engine, Drools [18], responsible for running the rules;
- WSBinder [19], responsible for executing binding actions at runtime based on the directions defined in the rule language;

- a set of Proxies that decouple the BPEL process and the execution environment from the logic needed to support reconfiguration.

The components of SCENE interact through a *publish-subscribe* paradigm. At runtime, when the execution of the process reaches the invocation of an external service, a proxy operation is actually called. If the proxy does not refer to any concrete service, it emits a *bindingEvent*. The rule engine - that has subscribed to this event - receives it and activates a rule able to handle - possibly with the activation of *WSBinder* - the missing binding. The control is then passed to the proxy that, possibly activating an adapter, invokes the proper operation on the bound service, and then passes the control back to the BPEL execution environment.

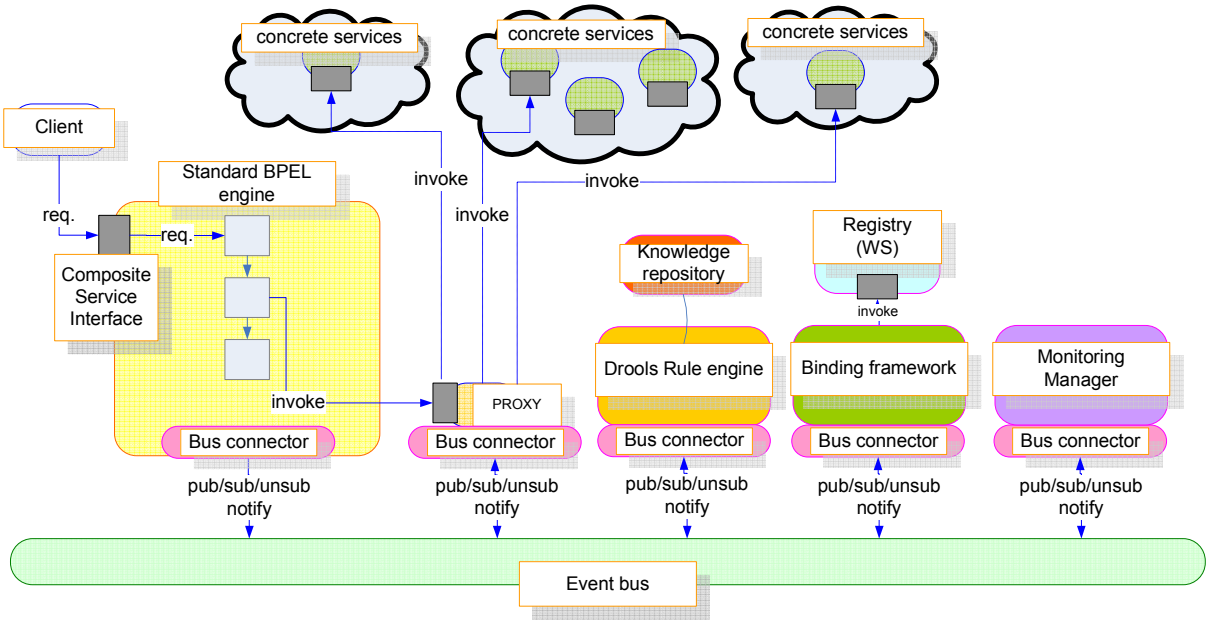

Figure 2 – Architecture of SCENE

3.1.1.3 The SCENE process deployer

An additional component of the SCENE platform is the *Process Deployer*. The Process Deployer is not meant to interact directly with a human user, it is meant to be called by a composition designer.

When a deploy process is launched a number of activities are performed:

- The WSDLs of the services used by the BPEL process are downloaded and weaved to produce the WSDL of the proxies used by the platform to enable dynamic binding.
- The BPEL process is weaved to produce the BPEL process using the proxies instead of the real services.
- The java code implementing the stubs, used by the proxies to access the real services is automatically generated.
- The java code implementing the proxies is automatically generated and it is packaged as axis2 services.
- The weaved BPEL process and the weaved WSDLs are packaged as an ActiveBpel deployable archive.
- The proxies are deployed under Axis2
- The weaved BPEL process is deployed under ActiveBpel

- The rules written in the high-level language are translated into the low-level language required by Drools and deployed.
- Some deployment descriptors of the SCENE platform are updated with the necessary information so that a new deployed process is available for invocation.

3.1.2 Semantic BPEL Execution Engine (SBPELEE)

The SUPER architecture [6] distinguishes two layers for Semantic Business Process (SBP) execution: the Semantic BPEL Execution Engine (SBPELEE) and the Semantic Execution Environment (SEE). The SBPELEE layer is responsible for orchestrating the execution of some activities according a the control and data flow.

3.1.2.1 The SBPELEE language

SBPELEE executes a process described in terms of a BPEL4SWS model, an SA-WSDL document describing the service defined by the process model and a WSDL containing the service descriptions to import and deploy in the process model.

As described in **Error! Reference source not found.** BPEL4SWS extends the BPEL language with the ability to use semantic information for describing activity implementations using semantics and thus independent of their interface descriptions. In addition, data models used in processes are represented semantically using ontologies, which enable the use of process relevant data for reasoning. Mismatches on the data and process level can also be resolved using mediation on the ontological level.

3.1.2.2 The SBPELEE architecture

The architecture of SBPELEE, as described in [6], has been centred on Apache ODE, which is the execution engine chosen by the SUPER consortium to be extended. Apache ODE was extended in SUPER for supporting BPEL4SWS.

3.1.2.3 The SBPELEE deployer

In order to be able to deploy a BPEL4SWS model into SBPELEE people from SUPER changed both the Parser and Compiler of Apache ODE for supporting some BPEL extensions they have defined.

3.2 Execution engine development roadmap

In SOA4All we are evolving the SCENE Platform and the SCENE Process Deployer in order to obtain an execution environment for lightweight processes as described in [1]. In order to achieve this objective we are implementing new approaches that aim at simplifying and partially automating the work of a potential user that has to design and execute a lightweight process. In particular:

- In section 3.3 we present in details a new approach for adapting service requests to actual service interfaces through semantic annotations. This is also the main achievement already realized during the first year of the SOA4All project and it is the main new functionality of the first prototype of the composition framework released at the time of writing.
- In section 3.4, we present an approach to the generation of artifacts. We are defining and implementing an approach for the generation of the executable artifacts needed by the SCENE platform starting from the more abstract artifacts produced by the user (as described in [1]) and optimized by our adaptation environment (as described in [2]).
- In section 3.5 we show how the extended SCENE platform will be integrated in the SOA4All runtime (as described in [3])

In this section, we provide, summarized in Table 2, the roadmap for development of the SOA4All execution engine software. It specifies the delivery dates of the software associated with task T6.5 and what will be the planned extensions for each milestone.

Component **Milestone**	***Lightweight Process* Executor**	***Lightweight Process Deployer***
M12 Prototype **Semantic Enabled Execution Engine**	Developed a new approach for adapting service requests to actual service interfaces at runtime Extended the SCENE platform integrating the new approach Conducted an experiment on a sample scenario	-
M18 Established **1st Prototype Demonstrator**	Experiments for substituting the BPEL engine used in SCENE with SBPELEE developed in SUPER New experiments based on the WP7 scenario (see [5])	Definition of the artefacts generation process
M24 Refined **Advanced Prototype For Adaptive Service Composition Execution**	Development of new approaches for exploiting semantic annotations in self-adaptation at runtime Development of an extended demonstrator implementing the WP7 scenario Partial integration in the SOA4All Runtime	Development of the 1st version of the Process Deployer Development of a demonstrator based on the scenario defined in the WP7 scenario Partial integration in the SOA4All Studio
M30 Matured **Final Prototype For Adaptive Service Composition Execution**	Refinement of the Executor Integration in the SOA4All Runtime	Refinement of the Deployer Integration in the SOA4All Studio

Table 2 – Software development roadmap

3.3 A new approach for adapting service requests to actual service interfaces through semantic annotations

An interesting challenge in the context of service oriented systems is the possibility of building applications where loosely coupled component services can be selected at run time

and can be replaced by other services when needed. Most of the research efforts aiming at supporting this dynamic selection and binding to services assume that the interface of the services to be composed are known at design time. In [7], based on our experience with industrial partners of the European integrated project SeCSE [12] we argued that this assumption was not totally realistic, as a consequence of the lack of standardization in service oriented systems. To solve this problem, in the same paper we presented an approach to allow invocation of services whose interface and behavior differs from one another. This approach is incorporated in the SCENE framework described in section 3.1.1 above. In that paper we identified a number of possible *mismatches* between services and some basic mapping functions that can be used to solve simple mismatches. Such mapping functions can be combined in a *script* to solve complex mismatches. Scripts can be executed by a mediator that receives an operation request, parses it, and eventually performs the needed adaptations.

Such approach requires the definition of scripts by some human being able to completely understand the mismatches and properly combine the mapping functions. In many situations this is not possible. Furthermore, such approach requires an intensive effort from a system integrator that, in the worst case in which a client can be bound to N different services, could be requested to specify N adapters for each client. In this document we describe a new approach that aims at solving these problems by exploiting semantic annotations of service interfaces and domain ontologies. Our approach extends the previous work and permits the automation of the definition of the adapting scripts.

3.3.1 The approach used in SCENE for adapting service requests to actual service interfaces

3.3.1.1 Mismatches definition

We say that a service consumer assumes to interact with some *abstract service* that can have various concrete realizations (the *concrete services*), all semantically equivalent to the abstract service, but that can show some differences with the abstract service in the way they need to be exploited by the consumer.

A service is described by an *interface* and a *protocol*. The former is defined as a tuple $I_s = O_s;D_s$, where O_s is the set of offered WSDL operations and D_s is a collection of data the service can understand. Each offered operation has a name, a set of input parameters and a set of output parameters. Each datum has a name, a type and a value.
A protocol is defined as a state machine characterized by tuple $P_s = (O_{names}; S_s; T_s)$, where O_{name} is the input alphabet of the state machine, containing names of operation associated to service transitions. S_s is the set of states the service can go through, and Ts is the set of transitions defined in the protocol state machine.
Given an abstract service S_{abs} and a concrete service S_{conc}, we say that a mismatch occurs when an operation request expressed in terms of the abstract interface cannot be understood by the concrete service that should handle it.

We distinguish the following two classes of mismatches:

- Interface-level mismatches: concern differences between names of operations exposed by an abstract and a concrete service and parameters of these operations.
- Protocol-level mismatches: concern differences in the order in which the operations offered by an abstract service and by its concrete representation are expected to be invoked. In this case we say that there is a mismatch between the abstract service protocol, P_{abs}, and the concrete service protocol, P_{conc}.

3.3.1.2 Mapping Functions and Mapping Language definition

To solve the mismatches listed above we defined a set of basic mapping functions. Since in real cases we often observe combination of mismatches, also our mapping functions can be

combined to provide a solution to complex mismatches.

We based our work on two simplifying hypotheses: there is no non-determinism in services protocols, and there is no operation names overloading. Under these hypotheses our basic mapping functions are defined as follows:

- ParameterMapping: maps abstract service input data on concrete service input data.
- ReturnParameterMapping: maps abstract service output data on concrete service output data.
- OperationMapping: maps abstract service operations on concrete service operations.
- StateMapping: maps an abstract service state on a concrete service state.
- TransitionMapping: maps an abstract service transition on a concrete service transition.

Basic mapping functions can be combined in adaptation scripts, defined in a domain specific executable language. The language is composed of rules structured in two parts:

- A *mismatch definition part* that specifies the type of the mismatch to be solved by the rule, and contains two sub-elements: *input*, specifying the elements of the abstract service that show the given mismatch, and *mapping*, specifying the elements of the concrete service the input elements have to be mapped on.
- A *mapping function part* that contains the name of the function to be used to solve the mismatch.

For an in depth treatment of mismatches, mapping functions and mapping language see [7].

3.3.1.3 Limitations of the previous approach

The approach described in this section shows three main limitations:

- The definition of the scripts requires the intervention of some human being able to completely understand the mismatches and properly combine the mapping functions.
- It requires an intensive effort from a system integrator that, in the worst case in which a client can be bound to N different services, could be requested to specify N adapters for each client.
- It is complicated to add new services during the application run time, since system integrators should develop a new adapter if they want to invoke a service showing mismatches in their composition.

In SOA4All we extended the previous work enabling the automation of the definition of the adapting scripts.

3.3.2 *The approach extended* through the use of semantic annotations

3.3.2.1 The example

In order to better explain our extended approach we refer to the weather forecast example mentioned in section 2.1. The system supports the selection of the concrete service to invoke at run time according to some rules. In particular, in this case the system selects the service with the best availability in the last hour. If the service fails to respond, the system tries with the second service with the best availability. The availability in the last hour of each candidate service is measured by a monitoring subsystem.

We make the hypothesis that the execution infrastructure exploits a set of basic mechanisms such as dynamic discovery, selection, adaptation, invocation, mediation and monitoring for supporting the dynamic and adaptive reconfiguration in reaction to environmental changes. In this perspective new services could be discovered and bound to the composition during

the system run time. Of course since there is lack of standardization these newly discovered services can show some mismatches. In case of mismatch it is necessary to define a new adaptation script in order to invoke the service in the composition.

We also assume that the system has to be designed to target non-technical users, which will not be able to add themselves adapters when a new service showing mismatches will be integrated into the composition.

On the base of the previous hypotheses it is not possible for a system integrator to develop new adaptation scripts for each newly discovered service, so the capability of automatically generate adapters is crucial for the system to work properly.

In Table 3 we summarize the interface of one of the candidate concrete services in terms of operation name, input parameters (type:name) and return value (type:name). In Table 4 we summarize the interface of another candidate concrete service. In both Table 3 and Table 4 complex types begin with uppercase while simple types begin in lowercase. We can see that there are some mismatches solvable by some mapping functions specified in section 3.3.1.2:

- ParameterMapping, e.g., int:hour → int:hours;
- ReturnParameterMapping, e.g., the simple type ForecastResponse → the complex type Forecast;
- OperationMapping, e.g., getForecats → Forecast.

Operation name	Parameters	Return value
getForecast	getForecast {	Forecast{
	double:latitude	string:utc
	double:longitude	string:wx
	int:hour	Weather:weather
	int:min	string:mslp
	}	string:temp
		string:relh
		string:prcp
		string:wind
		}

Table 3- Interface of TheWeather service

Operation name	Parameters	Return value
Forecast	Forecast {	ForecastResponse{
	double:latitude	string:return
	double:longitude	}
	int:hours	
	int:mins	
	}	

Table 4 -.Interface of the Forecast service

3.3.2.2 Adapting requests and responces for services in the example

In the example the first service interface (depicted in Table 3) is the one specified in the

abstract process. Thus in this case adapting from one service to another means allowing the process to invoke both services. In general it is always possibile to make an adaptation from an abstract interface specified in an abstract process to one or more interfaces specified for concrete services.

We make the hypothesis that the two services in the example do not have any internal or conversational state. Thanks to this hypothesis, we just need to build an adapter that provides a mapping between corresponding operations and their parameters. In the opposite situation we should also build an adapter that provides mapping for states and transitions.

Finally we assume that services involved in the composition are semantically annotated using the grounding schema described in [9]. Details of semantics annotation for the services in the example are reported in Annex C.

In order to bring the mapping language to a higher level of abstraction and overcome the limitations of the previous approach, we exploit in this section a semantics based approach to automate the adaptation script creation. In Annex A.1 we present the resulting mapping functions for the three kinds of mismatches as constructed and used in the implementation of our example. In the example there is a new kind of mapping not described in [4] that solves kind of mismatches not tested before.

In fact, the response message of the ForecastService service is a string concatenation of three elements:

- a value of temperature in degrees centigrade;
- the separator character ",";
- a short description of the weather conditions (e.g.,rainy, cloudy, sunny).

Such mismatch required the definition of two distinct mapping functions:

- a ReturnParameterMapping function for mapping the concrete service output to the abstract service output data.
- and a new kind of mapping function StringTokenizer for lifting the concrete service output comma separated string to a structured data type.

These mapping scripts solved all the mismatches in this experimental case study. The problem, as already summarized in section 2.3, is that in writing those mapping scripts there is the need for a complete understanding of the semantics of each operation and type described in the service interfaces.

In fact, in the resulting mapping script (e.g. the one showed in Annex B) there is more information than the original service descriptions. This information gap must be bridged by the system integrator. On the contrary, if the needed information about the semantics associated with the elements of the service descriptions is available, the problem would be almost completely manageable by a software agent.

Our hypothesis is to have for each candidate service a SAWSDL description based on some common domain ontologies. The ontology in our case study can be expressed in the OWL language or in WSML language as described in [9]. Compared to our previous approach in which the system integrator has to build a mapping from one service description to another, now each service provider is requested to annotate a service description using a common domain specific ontology.

In Annex C we report the semantically annotated WSDL of the "TheWeather" service and of the "ForecastService" service. The two SAWSDLs show that the two operations named respectively getForecast and Forecast were annotated using the same concept of the WeatherrForecast domain ontology *../weatherForecasts/#WeatherForecastByTimeLocation.*

This can be interpreted as semantic similarity between the two operations and as the fact that they can execute the same abstract task: return the forecast of the weather conditions given the location and time. Of course this is not enough for inferring that the getForecast and Forecast operations can substitute each other in a dynamic service composition. In both SAWSDLs descriptions all the data types used as input and output parameters of the operations are annotated once again using the concepts of the *WheaterForecast, GPS* and *Time* domain specific ontologies. These ontologies were developed expressly for the implementation of this example and are showed in Annex E.The input and output parameters (at least mandatory inputs and outputs) of two operations must be annotated to the same concept of the same ontology for inferring that the operation request message of one service can be mapped to the operation request message expected by the second service. The same happens for response messages.

Again this is not always enough for completing the mapping. In some cases the need of a lifting or lowering mapping schema to map an unstructured data to a structured one or vice versa is raised. Below we describe how to use the semantic annotations to the WSDLs for generating the various parts of the mapping script. This information can be used for solving the mismatch on the operation name. In our prototype, a software agent executed at design or publication time:

1. parses the SAWSDLs,
2. finds the concepts of the ontology related to the different operations,
3. establishes the mismatch on the operation name,
4. selects the corresponding mapping function Rename-Operation,
5. produces the mapping script.

In the same way, a software agent is able to parse the annotated schema types of the input parameters in the SAWSDLs and to find the relationships between the various elements:

- getForecast.latitude → .../GPS/#latitude → Forecast.latitude
- getForecast.longitude → .../GPS/#longitude → Forecast.longitude
- getForecast.hour → .../time/#timeDistance_hours → Forecast.hours
- getForecast.min → .../time/#timeDistance_minutes → Forecast.mins

Resolved the mismatch on the parameters names the mapping agent will look for mismatches in the input parameter types. In this case there are no mismatches on types.

The mismatch on the responses is more complex. In the SAWSDL of the Forecast service we can see that the response type is the complex type Forecast that is a sequence of simple elements; each element has its model reference in the ontology and from the annotations we infer that the *Forecast* complex type must be interpreted as an instance of the *WeatherForecast* concept.

In the SAWSDL of the TheWeather service again we can see that the response must be interpreted as an instance of the WeatherForecast concept but now the response message type is string. In this case we are able to solve the mismatch on the parameter name but not on the parameter type. Anyway, it has to be noted that in the SAWSDL of the TheWeather service, associated with the response message type there is a lifting mapping schema. In SAWSDL a lifting schema mapping lifts data from a structured data type to a semantic model. SAWSDL does not prescribe any particular mapping representation language. Here we used our own mapping language derived directly from our mapping functions already described in section 3.3.1.2. In Annex D there is the lifting schema mapping for the ForecastService response. Given this information again our software agent is able to infer the mapping function and generate the mapping script.

Our approach allows the invocation of services whose interface and behavior differ from each other by means of semantic annotations. The approach addresses the following requirements of SOA4All:

- **adaptation**, since it enables the automation of the adaptation scripts, overcoming some of the current limitations and issues connected to lack of standardization in service centric systems,
- **user-friendly composition**, since it allows developing new added-value services in a lightweight and effective manner and it moves some tasks, in the process of developing added-value services, from the system integrator (e.g., manual mapping for adapting service requests to actual service interfaces) to each service provider (e.g., annotation of service descriptions);
- **system self-reconfiguration**, since it eliminates the complexity of adding new mismatching services at run time to the service composition by developing service adapters;

The main advantage of our solution is the possibility to bring the service mapping language to a higher level of abstraction by means of semantic annotations of service interfaces: we achieve service composition by building service descriptions that exploit shared domain ontologies, instead of describing service mappings at a syntactical level. This approach greatly simplifies the work of SOA4All users, since it simplifies adaptive and dynamic service composition also in an open world setting. At the time of writing, the implementation of the first prototype implementing the new approach was released.

In the next section 3.4 we describe a new approach for the deployment of a lightweight process into the execution engine. This approach will be developed during the second year of the project and it will replace the process deployer component developed for SCENE.

3.4 Generation of runtime artefacts

Once a lightweight process has been designed (for details about lightweight process modelling see [1]), several deployable artefacts need to be generated in order to prepare it for execution: i.e. all the deployable elements listed in section 3.1.1.3 and the mapping scripts described in section 3.3.2.2.

Before generation can begin, abstract lightweight processes need to be translated into executable processes. We will evaluate the use of a meta-model based approach to generating artefacts, leveraging work done in the Eclipse SOA Tools (STP) Project as part of the STP-Intermediate Model component. The artefacts to be generated include the actual executable language (i.e. the SCENE language) to be deployed to the execution engine, as well as architectural artefacts required by the SOA4All Runtime, as described in [3].

The generative approach provided by STP-IM provides a means to move information from successive process design and architecture specification stages to the infrastructure development stages corresponding to technologies such as JBI, which is being targeted by SOA4All.

Figure 3 outlines the generic relationship between different possible SOA editors, the STP-IM and the SOA runtimes (such as Process Engines, ESBs or SCA platforms). Editors not using the STP-IM can directly generate deployment artifacts for the SOA runtime of choice. Naturally, the process editor in SOA4All will go through a transformation in order to generate executable processes to be deployed onto the process engine.

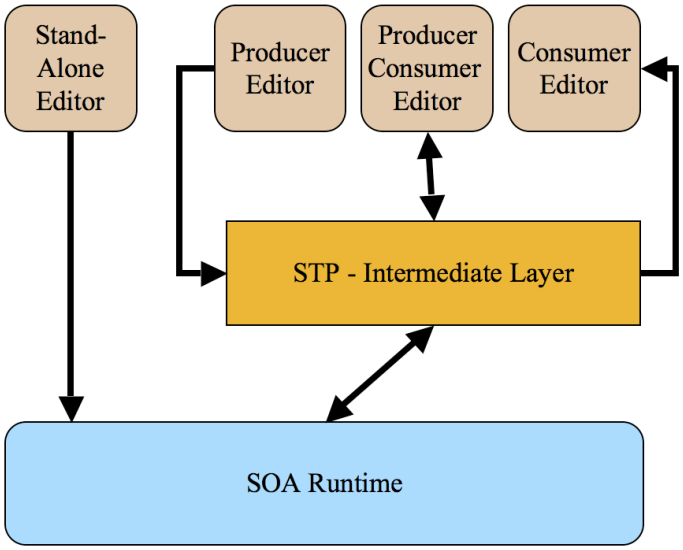

Figure 3 - Intermediating SOA Development Spaces

In addition, editors that target other editors for exporting artefacts must have one transformation for each such target editor. Using the Intermediate Model, editors must only have one transformation for exporting data and / or one transformation from importing data to / from the IM. This could be useful in SOA4All for transporting information between different Studio editors. The STP-IM contains elements that aim to cover some of the most common concepts found in SOA-related editors and deployment platforms. A detailed description of the meta-model can be found at http://www.eclipse.org/stp/im

Due to this “hybrid” nature of the meta-model, some of its concepts cannot map directly to the corresponding concepts in each of the editors it aims to unite. This is unavoidable and in fact desirable in order to attain a higher-level set of abstractions that can more easily map to different specifications. By using highly configurable elements, the IM facilitates the transport of platform-specific information between editors, while not directly supporting the full semantics of each element of source and target editors.

In its current implementation, the STP-IM has a number of plug-ins, one for each type of transformation (such as one plug-in from the transformation from BPMN to the IM, one for the transformation from the IM to SCA and so on). Each plug-in provides its own pop-up menu item registered for particular file types. In addition, the STP-IM must be made usable outside the Eclipse context. This is possible as the Eclipse Modelling Framework (EMF) can be extracted and used separately from the Eclipse platform. The STP-IM will be extended to include additional elements as required by the SOA4All lightweight process language, as well as the additional transformations necessary to target the SOA4All runtime.

3.5 Integration in SOA4All

In Figure 4 we depict the overall picture of WP6 and where we situate it inside the overall SOA4All architecture. Let us briefly describe the components of the Service Construction environment, in order to situate the software that we are going to describe in this deliverable. Users will use the user interface component to specify their required composite services and processes (part of the SOA4All Studio). Nevertheless, we need to define a graph-oriented lightweight process modelling language that we will use as specification language. To improve usability pre-designed and user-designed process templates are stored in the semantic service & template repository.

Once created and stored, in order to be usable and interpretable these lightweight processes have to be translated in to more complex processes that can be interpreted by an execution

in an effective fashion. We will create a **scalable design-time composer for the flexible and ad-hoc creation and adaptation of complex services at design time**. The system will transparently transform the aforementioned lightweight processes in to optimized complex services orchestrations; or already existing complex services processes could be adapted to a specific use. These activities will be heavily influenced by the context in which they will be carried out.

Finally, regarding the runtime phase of service construction, the outcome work package WP6 will be the **execution engine**. It will execute complex processes that represent orchestration of services. The execution engine is an execution infrastructure for lightweight processes, adaptive to environmental changes and flexible enough to allow its context-dependent self-reconfiguration. In SOA4All the term "execution infrastructure for lightweight processes" mainly refers to model and execute composite services and processes in a lightweight manner as described in [1]. This execution infrastructure will exploit a set of basic mechanisms such as dynamic discovery, selection, adaptation, invocation, monitoring developed in the work packages WP1, WP2 and WP5, for supporting the dynamic and adaptive reconfiguration in reaction to environmental changes.

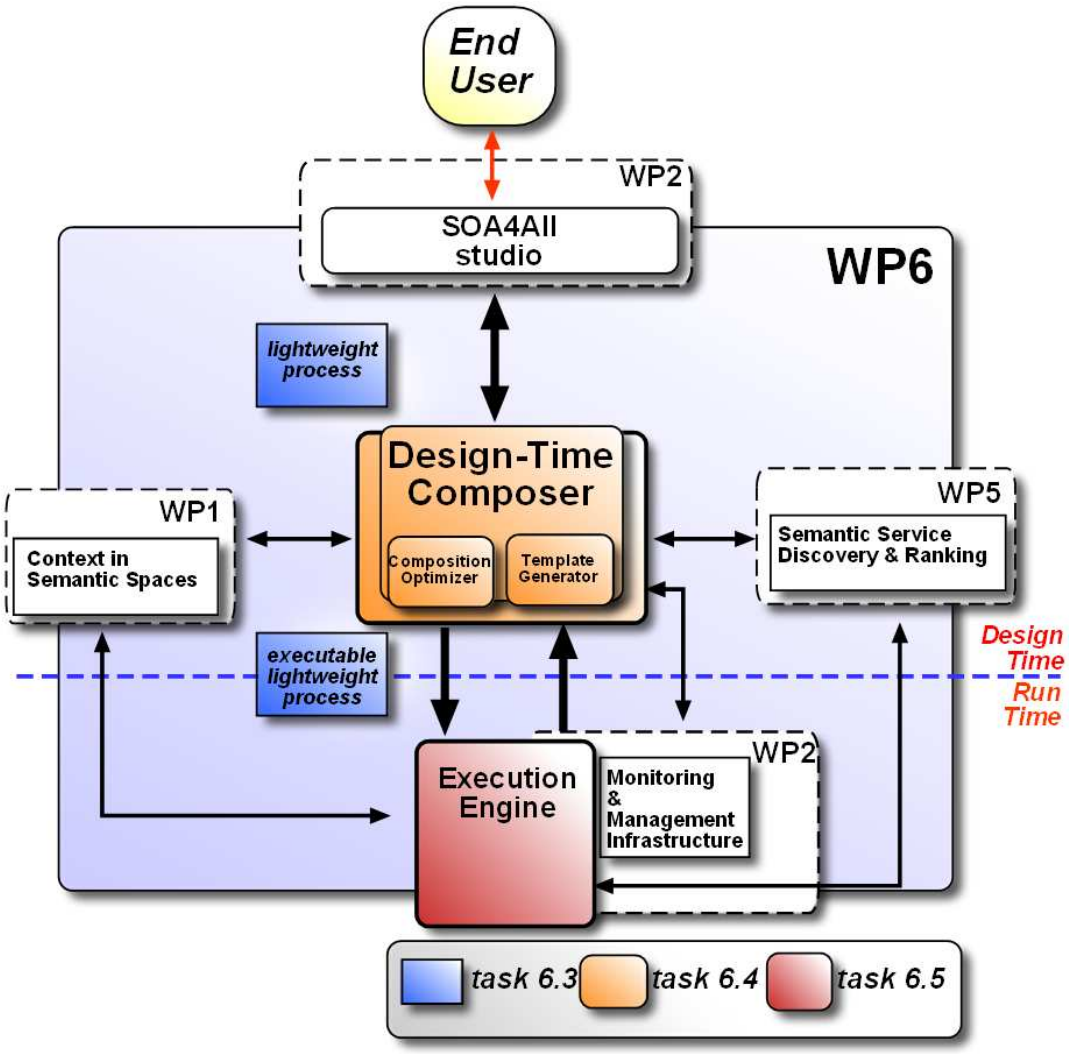

Figure 4 Service Construction Framework overall picture.

3.5.1 Execution Engine Main Components Description

3.5.1.1 Lightweight Process Executor

- **Description**: the Lightweight Process Executor component will carry out the adaptive and context aware execution of process and service composition, offering thus one external interface for each process deployed within the execution environment. During execution a lightweight process need to invoke concrete services trough the Distributed Service Bus and may need to discover new candidate services for an activity. The execution produces logs for the Template Generator component and produces and uses monitoring information.
- **Inputs**:
  - A set of user requirements (user constraints and preferences).
  - The execution values (the inputs-outputs of the process activities)
  - A set of monitoring data about previous executions
- **Outputs**:
  - Effect of the execution
  - Past processes and/or services execution logs (input for Template Generator, see [2])
- **Interfaces exposed**:
  - there is one external interface for each process deployed within the execution environment; in fact every process is exposed as a service

3.5.1.2 Lightweight Process Deployer

- **Description**:
- **Inputs**:
  - A concrete composition i.e., the output of the Composition Optimizer
  - A set of semantic annotated service descriptions
  - The ontologies used for annotating service descriptions
  - The classification of possible faults and unavailabilities of component services
- **Outputs**:
  - A process ready to be invoked and executed, exposed as a service
- **Interfaces exposed**:
  - IDeployer: the interface IDeployer

3.6 Summary of fulfilled requirements

In Table 5 we summarize the list of requirements already described in detail in section 2 and for each requirement we indicate if the requirement is fulfilled or will be fulfilled by Execution Environment.

N.	Requirement	Already (even partially) fulfilled in SeCSE or SUPER	Already fulfilled in the 1st protoype	Planned for the final prototype
F1.1	**Adaptation**	**O**	**X** Extended the SeCSE approach exploiting semantic annotations	**X** Refinements of the approach and new experiments are planned
F1.2	**Dynamic Binding**	**X**	**X**	**X**
F2	**Functional fault handling**	**O**	**O**	**X** Planned to extend the approach exploiting the classification of the possible faults developed in other tasks
NF1	**User-friendly composition**	**-**	**X** User-friendly composition is more related to T6.3 and T6.4. However our approach makes dynamic binding mechanisms transparent for the user	**X**
S1	**Adaptation to contextual changes**	**X**	**X**	**X** Planned to extend the approach exploiting the classification of contextual information developed in other tasks

S2	**System Self-configuration**	**O**	**X** The developed approach allows for self reconfiguring taking advantage from the presence of a new published service	**X** Planned to extend the Process Deployer and Executor with self-configuration capabilities exploiting the work of the Template Generator developed in T6.4
S3	**Self-optimization**	**-**	**-**	**X** Planned to extend the Process Deployer and Executor with self-optimization capabilities exploiting the work of the Process Optimizer developed in T6.4
S4	**SOA4All Integration**	**-**	**-**	**X** See section 3.2
S5	**Scalability**	**-**	**-**	**O** Scalability will be considered in the final evaluation criteria

*Table 5 – Summary of fulfilled requirements (**-** no, **O** partially, **X** yes)*

4. Conclusions

In the context of the SOA4All project, the execution framework for composite services and processes should enable different groups of end users to build new services and processes according to their specific needs in a lightweight manner.

In this deliverable, we have studied the requirements of lightweight, context-aware process execution from WP7, WP8, and WP9 and provided our methodology and approach for adaptive process execution and for the generation of runtime artefacts starting from the modelling language defined in D6.3.1.

At the time of writing, the implementation of our solution is under way. We released a first prototype on the end of February 2009. In this 1st prototype we developed a new approach for adapting service requests to actual service interfaces inside a dynamic composition of services exploiting semantic annotations. The developed approach allows for improving self-reconfiguring capabilities of the system and for taking advantage from the presence of a new published service. In fact, when a service is published, it could announce its capacity, described as WSMO Lite annotations, and then the system will be able to reconfigure itself by means of automatically developing a new mapping script. This enables the system for taking advantage from the presence of the new service without human intervention. Under this point of view, the approach helps also fulfilling the requirement of enabling service prosumers, since it simplify a lot the tasks of composing services.

We plan to develop a 1st demonstrator of the case study coming from the WP7 of SOA4All during the second year of the project.

5. References

[1] SOA4All deliverable D6.3.1 First Specification of Lightweight Process Modelling Language.

[2] SOA4All deliverable D6.4.1 Specification and First Prototype Of Service Composition and Adaptation Environment

[3] SOA4All deliverable D1.4.1 SOA4All Reference Architecture Specification

[4] SOA4All deliverable D1.1.1 Design Principles for a Service Web v1

[5] SOA4All deliverable D7.2 Scenario Definition

[6] SUPER deliverable D6.1 Execution Engine Design and Architecture

[7] L. Cavallaro and E. D. Nitto. An approach to adapt service requests to actual service interfaces. In SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, pages 129–136, New York, NY, USA, 2008. ACM.

[8] M. Colombo, E. Di Nitto, and M. Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In ICSOC, pages 191–202, 2006.

[9] Kopeck and A. Schtz. SOA4All - D1.2.1 WSMO grounding in SAWSDL. http://www.soa4all.eu/resources.html?func=startdown&id=14, 2008.

[10] X. Li, Y. Fan, and F. Jiang. A classification of service composition mismatches to support service mediation. In GCC '07: Proc. of the Sixth International Conference on Grid and Cooperative Computing (GCC 2007), pages 315–321, Washington, DC, USA, 2007. IEEE Computer Society.

[11] H. R. M. Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In WWW '07: Proc. of the 16th international conference on World Wide Web, pages 993–1002, New York, NY, USA, 2007. ACM Press.

[12] SeCSE (Service Centric System Engineering) Project. http://www.secse-project.eu.

[13] SUPER Project. http://www.ip-super.org

[14] World Wide Web Consortium (W3C). Semantic Annotations for WSDL and XML Schema.http://www.w3.org/TR/sawsdl/.

[15] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Version 1.1. http://www.w3.org/TR/wsdl.

[16] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Version 2.0. http://www.w3.org/TR/wsdl20-primer/.

[17] Active Endpoints: The ActiveBPEL Community Edition Engine, http://www.activevos.com/community-open-source.php

[18] JBoss: Drools, http://www.jboss.org/drools/

[19] Di Penta, M., Esposito, R., Villani, M.L., Codato, R., Colombo, M., Di Nitto, E.: Ws binder: a framework to enable dynamic binding of composite web services. In: ICSE Workshop on Service-Oriented Software Engineering (IW-SOSE 2006) (2006)

[20] Lessen, T., Nitzsche, J., Dimitrov, M., Konstantinov, M., Karastoyanova, D.,

Cekov, L., and Leymann, F. 2009. An Execution Engine for Semantic Business Processes. In Service-Oriented Computing - ICSOC 2007 Workshops: ICSOC 2007, international Workshops, Vienna, Austria, September 17, 2007, Revised Selected Papers, E. Di Nitto and M. Ripeanu, Eds. Lecture Notes In Computer Science, vol. 4907. Springer-Verlag, Berlin, Heidelberg, 200-211.

Annex A. Comparison between SOA4All, SeCSE and SUPER approaches

A.1. Differences and similarities between SOA4All Execution Engine and SBPELEE

In order to understand the differences / similarities with the approach followed by project Super-IP, we refer to Super Deliverable D6.2 – "Process Execution Engine First Prototype" and to [i]:

As a summary we include the following table with the main differences:

Table 6 Comparative analysis Super-IP vs SOA4All Execution Engine.

	Super-IP (SBPELEE):	**SOA4All:**
Main Purpose	BPEL for Semantic Web Services (BPEL4SWS) uses Semantic Web Service Frameworks to **define a communication channel between two partner services** instead of using the partner link which is based on WSDL 1.1.	SOA4All aims at making adaptive and dynamic service composition easier also for non-technical users.
Use of BPEL	BPEL4SWS extends the WS-BPEL 2.0 process model and uses existing WS-BPEL 2.0 capabilites**.**	Executes standard BPEL. The process model is expressed in terms of the lightweight process model defined in T6.3
Use of WSDL	WSDL-less interaction model. BPEL4SWS abstracts from interface definitions, i.e. port Types, and provides for a "WSDL-less interaction model". It is based on the newly introduced concept of a conversation.	WSDL-based interaction model.
Use of SAWSDL	BPEL4SWS uses SAWSDL to annotate data types of variables used in a process definition. The information given in SAWSDL/XSD documents is used to transform XML instance data into its ontological representation and vice versa. In case an error occurs during lifting or lowering of data, a liloFault has to be thrown.	SOA4All uses SAWSDL to help disambiguate the description of Web services. The information given in SAWSDL/XSD documents is used to create mapping scripts to solve complex mismatches. Scripts can be executed by a mediator that receives an operation request, parses it, and

[i] *Dimka Karastoyanova, Tammo van Lessen, Frank, Leymann, Jörg Nitzsche, Daniel Wutke: WS-BPEL Extension for Semantic Web Services (BPEL4SWS), Version 1.0, Technical Report No. 2008/03*

		eventually performs the needed adaptations.
Use of WSMO	Support WSMO service descriptions	Support for WSMO-lite service descriptions

A.2. Differences and similarities between SOA4All Execution Engine and SeCSE

In order to understand the differences / similarities with the approach followed by project SeCSE, we refer to SeCSE Deliverable A3.D5. "Prototype of service-centric runtime platform" and to [ii] and [iii]:

As a summary we include the following table with the main differences:

Table 7 Comparative analysis Super-IP vs SOA4All Execution Engine.

	SeCSE (SCENE):	**SOA4All:**
Main Purpose	SeCSE aims to offer to system integrators proper mechanisms for supporting dynamic changes and the explicit definition of self-configuration policies.	SOA4All aims at making adaptive and dynamic service composition easier also for non-technical users.
Use of BPEL	Executes standard BPEL. The process model is expressed in terms of an abstract BPEL.	Executes standard BPEL. The process model is expressed in terms of the lightweight process model defined in D6.3.1
Use of WSDL	WSDL-based interaction model. Abstract tasks are expressed as WSDL operations. Concrete services are selected and bound at runtime according to some rules.	WSDL-based interaction model. Abstract goals are concretized at design time (as defined in D6.4.1) and expressed as WSDL operations. In reaction to environmental changes or faults, according to some rules, new candidate concrete services can be selected and bound at runtime to the composition.
Use of SAWSDL	No use of SAWSDL. SeCSE specifies a solutions for possible mismatches between services interface building adaptation scripts,	SOA4All uses SAWSDL to help disambiguate the description of Web services. The information given in

[ii] *M. Colombo, E. Di Nitto, and M. Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In ICSOC, pages 191–202, 2006.*

[iii] *L. Cavallaro and E. Di Nitto. An approach to adapt service requests to actual service interfaces. In Proceedings of SEAMS '08, pages 129–136, New York, NY, USA, 2008.ACM.*

	defined in a **domain specific language**. The language is composed of rules structured in a mismatch definition part and a mapping function part. The adaptation scripts are developed by a system integrator and requires an intensive effort from a system integrator that, in the worst case in which a client can be bound to N different services, could be requested to specify N adapters for each client.	SAWSDL/XSD documents is used to create mapping scripts to solve complex mismatches. Scripts can be executed by a mediator that receives an operation request, parses it, and eventually performs the needed adaptations.
Use of WSMO-lite	No use of WSMO-Lite	Support for WSMO-lite service descriptions

Annex B. TheWeather to Forecast Mapping Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<ml:Mapping xmlns:ml="http://www.secse-project.eu/MappingLanguageSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.secse-project.eu/MappingLanguageSchema
MappingLanguageSchema.xsd ">
      <ml:RequestorNameSpace>http://it.crf.TheWeather/</ml:RequestorNameSpace
>
      <ml:ServiceNameSpace>http://zulu-53.nebula.fi/xsd</ml:ServiceNameSpace>
      <ml:StartEndpoint>http://inrete.dyndns.info/TheWeather/TheWeatherCRF.as
mx</ml:StartEndpoint>
      <ml:FinalEndpoint>http://zulu-
53.nebula.fi/ws/services/ForecastService</ml:FinalEndpoint>

      <ml:ProtocolRule >
      <ml:ReferenceRuleID>weather2forecasts</ml:ReferenceRuleID>
                  <ml:Input>
                        <ml:Name>getForecast</ml:Name>
                  </ml:Input>
                  <ml:Mapping>
                        <ml:Name>Forecast</ml:Name>
                  </ml:Mapping>
            </ml:OperationRename>
            <ml:MappingFunction>
                  <ml:Name>OperationMapping</ml:Name>
            </ml:MappingFunction>
      </ml:ProtocolRule>

      <ml:InterfaceRule ID="weather2forecasts">
      <ml:ReferenceRuleID>weather2forecasts</ml:ReferenceRuleID>
            <ml:DataBinding total="false">
                  <ml:Input>
                        <ml:Name>getForecast</ml:Name>
                        <ml:Side>ExpectedService</ml:Side>
                        <ml:type>getForecast</ml:type>
                  </ml:Input>
                  <ml:Mapping>
                        <ml:Name>Forecast</ml:Name>
                        <ml:Side>AvailableService</ml:Side>
                        <ml:type>Forecast</ml:type>
                  </ml:Mapping>

            <ml:Input>
                  <ml:Name>getForecast.latitude</ml:Name>
                  <ml:Side>ExpectedService</ml:Side>
                  <ml:type>double</ml:type>
            </ml:Input>
            <ml:Mapping>
                  <ml:Name>Forecast.latitude</ml:Name>
                  <ml:Side>AvailableService</ml:Side>
                  <ml:type>double</ml:type>
            </ml:Mapping>
```

```
            <ml:Input>
                  <ml:Name>getForecast.longitude</ml:Name>
                  <ml:Side>ExpectedService</ml:Side>
                  <ml:type>double</ml:type>
            </ml:Input>
            <ml:Mapping>
                  <ml:Name>Forecast.longitude</ml:Name>
                  <ml:Side>AvailableService</ml:Side>
                  <ml:type>double</ml:type>
            </ml:Mapping>

            <ml:Input>
                  <ml:Name>getForecast.hour</ml:Name>
                  <ml:Side>ExpectedService</ml:Side>
                  <ml:type>int</ml:type>
            </ml:Input>
            <ml:Mapping>
                  <ml:Name>Forecast.hours</ml:Name>
                  <ml:Side>AvailableService</ml:Side>
                  <ml:type>int</ml:type>
            </ml:Mapping>

            <ml:Input>
                  <ml:Name>getForecast.min</ml:Name>
                  <ml:Side>ExpectedService</ml:Side>
                  <ml:type>int</ml:type>
            </ml:Input>
            <ml:Mapping>
                  <ml:Name>Forecast.mins</ml:Name>
                  <ml:Side>AvailableService</ml:Side>
                  <ml:type>int</ml:type>
            </ml:Mapping>
      </ml:DataBinding>
      <ml:MappingFunction>
            <ml:Name>ParameterMapping</ml:Name>
      </ml:MappingFunction>
       <ml:ReturnMappingRuleID>weather2forecastsRet</ml:ReturnMappingRuleID>
</ml:InterfaceRule>

<ml:InterfaceRule ID="weather2forecastsRet">
<ml:ReferenceRuleID>weather2forecastsRet2</ml:ReferenceRuleID>
<ml:DataBinding total="false">
      <ml:Input>
            <ml:Name>ForecastResponse</ml:Name>
            <ml:Side>AvailableService</ml:Side>
            <ml:type>ForecastResponse</ml:type>
      </ml:Input>
      <ml:Mapping>
            <ml:Name>getForecastResponse</ml:Name>
            <ml:Side>ExpectedService</ml:Side>
            <ml:type>getForecastResponse</ml:type>
      </ml:Mapping>
</ml:DataBinding>
```

```
<ml:MappingFunction>
      <ml:Name>ReturnParameterMapping</ml:Name>
</ml:MappingFunction>
</ml:InterfaceRule >

<ml:InterfaceRule ID="weather2forecastsRet2">
<ml:DataBinding total="false">
      <ml:Input>
            <ml:Name>return</ml:Name>
            <ml:Side>AvailableService</ml:Side>
            <ml:type>string</ml:type>
      </ml:Input>
      <ml:MappingFunction>
            <ml:Name>UnformattedStringSplit</ml:Name>
            <ml:StringSplitter>, </ml:StringSplitter>
            <ml:Mapping>
                  <ml:Name>getForecastResult.temp</ml:Name>
                  <ml:Side>ExpectedService</ml:Side>
                  <ml:type>Forecast.string</ml:type>
            </ml:Mapping>
            <ml:Mapping>
                  <ml:Name>getForecastResult.weather</ml:Name>
                  <ml:Side>ExpectedService</ml:Side>
                  <ml:type>Forecast.Weather</ml:type>
            </ml:Mapping>
      </ml:MappingFunction>
</ml:DataBinding>
</ml:InterfaceRule>
```

Annex C. SAWSDLs service description

C.1. TheWeather service SAWSDL description specification

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://it.crf.TheWeather/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://it.crf.TheWeather/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
  <wsdl:types>
    <xs:schema elementFormDefault="qualified"
targetNamespace="http://it.crf.TheWeather/">
      <xs:element name="getForecast"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#weat
herForecastRequest">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="1" maxOccurs="1" name="latitude"
type="xs:double"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/GPS#latitude" />
            <xs:element minOccurs="1" maxOccurs="1" name="longitude"
type="xs:double"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/GPS#longitude" />
            <xs:element minOccurs="1" maxOccurs="1" name="hour" type="xs:int"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/time#timeDistance_hou
rs" />
            <xs:element minOccurs="1" maxOccurs="1" name="min" type="xs:int"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/time#timeDistance_min
utes" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getForecastResponse"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#weat
herForecastResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" name="getForecastResult"
type="tns:Forecast" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="Forecast"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#weat
```

```
herForecastResponse">
        <xs:sequence>
          <xs:element minOccurs="0" maxOccurs="1" name="utc" type="xs:string"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/time#UTCTime" />
          <xs:element minOccurs="1" maxOccurs="1" name="weather"
type="tns:Weather"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Weat
herConditionsShortDescription" />
          <xs:element minOccurs="0" maxOccurs="1" name="mslp"
type="xs:string"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Pres
sure_hPa" />
          <xs:element minOccurs="0" maxOccurs="1" name="temp"
type="xs:string"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Temp
erature_Centigrades" />
          <xs:element minOccurs="0" maxOccurs="1" name="relh"
type="xs:string"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Humi
dity_Percentage" />
          <xs:element minOccurs="0" maxOccurs="1" name="prcp"
type="xs:string"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Prec
ipitation_mm" />
          <xs:element minOccurs="0" maxOccurs="1" name="wind"
type="xs:string"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Wind
_Knots/>
        </xs:sequence>
      </xs:complexType>
      <xs:simpleType name="Weather">
        <xs:restriction base="xs:string">
          <xs:enumeration value="Unavailable" />
          <xs:enumeration value="Clear" />
          <xs:enumeration value="Cloudy" />
          <xs:enumeration value="Overcast" />
          <xs:enumeration value="Rain" />
          <xs:enumeration value="Shower" />
          <xs:enumeration value="Thunderstorm" />
          <xs:enumeration value="Variable_with_rain" />
          <xs:enumeration value="Variable_with_showers" />
          <xs:enumeration value="Variable_with_thunderstorms" />
          <xs:enumeration value="Sleet" />
          <xs:enumeration value="Snow" />
          <xs:enumeration value="Fog" />
        </xs:restriction>
      </xs:simpleType>
      <xs:element name="getWeatherStation">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="1" maxOccurs="1" name="latitude"
type="xs:double" />
            <xs:element minOccurs="1" maxOccurs="1" name="longitude"
type="xs:double" />
```

```
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getWeatherStationResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1"
name="getWeatherStationResult" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="getForecastSoapIn">
    <wsdl:part name="parameters" element="tns:getForecast" />
  </wsdl:message>
  <wsdl:message name="getForecastSoapOut">
    <wsdl:part name="parameters" element="tns:getForecastResponse" />
  </wsdl:message>
  <wsdl:message name="getWeatherStationSoapIn">
    <wsdl:part name="parameters" element="tns:getWeatherStation" />
  </wsdl:message>
  <wsdl:message name="getWeatherStationSoapOut">
    <wsdl:part name="parameters" element="tns:getWeatherStationResponse" />
  </wsdl:message>
  <wsdl:portType name="TheWeatherCRFSoap">
    <wsdl:operation name="getForecast">
      <wsdl:input message="tns:getForecastSoapIn" />
      <wsdl:output message="tns:getForecastSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="getWeatherStation">
      <wsdl:input message="tns:getWeatherStationSoapIn" />
      <wsdl:output message="tns:getWeatherStationSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="TheWeatherCRFSoap" type="tns:TheWeatherCRFSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getForecast"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Weat
herForecastByTimeLocation" >
      <soap:operation soapAction="http://it.crf.TheWeather/getForecast"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getWeatherStation">
      <soap:operation soapAction="http://it.crf.TheWeather/getWeatherStation"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
```

```
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="TheWeatherCRFSoap12" type="tns:TheWeatherCRFSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getForecast">
      <soap12:operation soapAction="http://it.crf.TheWeather/getForecast"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getWeatherStation">
      <soap12:operation
soapAction="http://it.crf.TheWeather/getWeatherStation" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="TheWeatherCRF">
    <wsdl:port name="TheWeatherCRFSoap" binding="tns:TheWeatherCRFSoap">
      <soap:address
location="http://inrete.dyndns.info/TheWeather/TheWeatherCRF.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
     </FacetSpecificationData>
</LanguageSpecificSpecification>
```

C.2. Forecast service SAWSDL description specification

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:axis2="http://zulu-53.nebula.fi"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:ns0="http://zulu-
53.nebula.fi/xsd" xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://zulu-
53.nebula.fi" xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
           <wsdl:documentation>ForecastService</wsdl:documentation>
```

```
            <wsdl:types>
                <xs:schema xmlns:ns="http://zulu-53.nebula.fi/xsd"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://zulu-53.nebula.fi/xsd">
                    <xs:element name="Forecast"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Weat
herForecastRequest">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element minOccurs="0" name="latitude"
type="xs:double"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/GPS#latitude" />
                                <xs:element minOccurs="0" name="longitude"
type="xs:double"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/GPS#longitude"/>
                                <xs:element minOccurs="0" name="hours"
type="xs:int"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/time#timeDistance_hou
rs" />
                                <xs:element minOccurs="0" name="mins"
type="xs:int"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/time#timeDistance_min
utes" />
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="ForecastResponse"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Weat
herForecastResponse">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element minOccurs="0" name="return"
nillable="true" type="xs:string"
sawsdl:liftingSchemaMapping="http://www.soa4all.eu/ontologies/mapping/Forecas
tResponse2WheaterOntology.ml" />
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:schema>
            </wsdl:types>
            <wsdl:message name="ForecastRequest">
                <wsdl:part element="ns0:Forecast" name="parameters"/>
            </wsdl:message>
            <wsdl:message name="ForecastResponse">
                <wsdl:part element="ns0:ForecastResponse" name="parameters"/>
            </wsdl:message>
            <wsdl:portType name="ForecastServicePortType">
                <wsdl:operation name="Forecast"
sawsdl:modelReference="http://www.soa4all.eu/ontologies/weatherForecasts#Weat
herForecastByTimeLocation">
                    <wsdl:input message="axis2:ForecastRequest"
wsaw:Action="urn:Forecast"/>
                    <wsdl:output message="axis2:ForecastResponse"
wsaw:Action="urn:ForecastResponse"/>
```

```
                </wsdl:operation>
            </wsdl:portType>
            <wsdl:binding name="ForecastServiceSOAP11Binding"
type="axis2:ForecastServicePortType">
                <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
                <wsdl:operation name="Forecast">
                    <soap:operation soapAction="urn:Forecast"
style="document"/>
                    <wsdl:input>
                        <soap:body use="literal"/>
                    </wsdl:input>
                    <wsdl:output>
                        <soap:body use="literal"/>
                    </wsdl:output>
                </wsdl:operation>
            </wsdl:binding>
            <wsdl:binding name="ForecastServiceSOAP12Binding"
type="axis2:ForecastServicePortType">
                <soap12:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
                <wsdl:operation name="Forecast">
                    <soap12:operation soapAction="urn:Forecast"
style="document"/>
                    <wsdl:input>
                        <soap12:body use="literal"/>
                    </wsdl:input>
                    <wsdl:output>
                        <soap12:body use="literal"/>
                    </wsdl:output>
                </wsdl:operation>
            </wsdl:binding>
            <wsdl:binding name="ForecastServiceHttpBinding"
type="axis2:ForecastServicePortType">
                <http:binding verb="POST"/>
                <wsdl:operation name="Forecast">
                    <http:operation location="ForecastService/Forecast"/>
                    <wsdl:input>
                        <mime:content part="Forecast" type="text/xml"/>
                    </wsdl:input>
                    <wsdl:output>
                        <mime:content part="Forecast" type="text/xml"/>
                    </wsdl:output>
                </wsdl:operation>
            </wsdl:binding>
            <wsdl:service name="ForecastService">
                <wsdl:port binding="axis2:ForecastServiceSOAP11Binding"
name="ForecastServiceSOAP11port_http">
                    <soap:address location="http://zulu-
53.nebula.fi/ws/services/ForecastService"/>
                </wsdl:port>

            </wsdl:service>
</wsdl:definitions>
```

Annex D. Lifting schema mapping

```
<ml:DataBinding total="false">
      <ml:Input>
            <ml:Name>return</ml:Name>
            <ml:Side>AvailableService</ml:Side>
            <ml:type>string</ml:type>
      </ml:Input>
      <ml:MappingFunction>
            <ml:Name>UnformattedStringSplit</ml:Name>
            <ml:StringSplitter>, </ml:StringSplitter>
            <ml:Mapping>

      <ml:Name>http://www.soa4all.eu/ontologies/weatherForecasts#Temperature_
Centigrades</ml:Name>
                  <ml:Side>ExpectedService</ml:Side>
                  <ml:type>Forecast.string</ml:type>
            </ml:Mapping>
            <ml:Mapping>

      <ml:Name>http://www.soa4all.eu/ontologies/weatherForecasts#WeatherCondi
tionsShortDescription</ml:Name>
                  <ml:Side>ExpectedService</ml:Side>
                  <ml:type>string</ml:type>
            </ml:Mapping>
      </ml:MappingFunction>
</ml:DataBinding>
```

Annex E. Sample ontologies

E.1. Sample GPS ontology

```
namespace { _"http://www.soa4all.eu/ontologies/GPS#"
,
     wsmostudio _"http://www.wsmostudio.org#" }

ontology _"http://www.soa4all.eu/ontologies/GPS"
     nonFunctionalProperties
          wsmostudio#version hasValue "0.7.3"
     endNonFunctionalProperties

concept latitude

concept longitude
```

E.2. Sample time ontology

```
namespace { _"http://www.soa4all.eu/ontologies/time#"
,
     wsmostudio _"http://www.wsmostudio.org#" }

ontology _"http://www.soa4all.eu/ontologies/time"
     nonFunctionalProperties
          wsmostudio#version hasValue "0.7.3"
     endNonFunctionalProperties

concept minute

concept hour
     hour impliesType  (0 24) _decimal

concept UTCTime
```

E.3. Sample weather forecast ontology

```
namespace { _"http://www.soa4all.eu/ontologies/weatherForecast#"
,
     wsmostudio _"http://www.wsmostudio.org#" }

ontology _"http://www.soa4all.eu/ontologies/weatherForecast"
     nonFunctionalProperties
          wsmostudio#version hasValue "0.7.3"
     endNonFunctionalProperties

     importsOntology
          { _"http://www.soa4all.eu/ontologies/coordinate",
            _"http://www.soa4all.eu/ontologies/time"}
```

```
concept weatherForecastConditionsShortDescription

concept Pressure_hPa

concept Temperature_Centigrades

concept Humidity_Percentage

concept Wind_Knots
     direction impliesType _string
     speed impliesType _string

concept Precipitation_mm

concept getweatherForecast
     latitude impliesType
_"http://www.soa4all.eu/ontologies/coordinate#latitude"
     longitude impliesType
_"http://www.soa4all.eu/ontologies/coordinate#longitude"
     hour impliesType _"http://www.soa4all.eu/ontologies/time#hour"
     minute impliesType _"http://www.soa4all.eu/ontologies/time#minute"

concept weatherForecastResponse
     utc impliesType _"http://www.soa4all.eu/ontologies/time#UTCTime"
     weatherForecastConditionsShortDescription impliesType
_"http://www.soa4all.eu/ontologies/weatherForecast#weatherForecastConditionsS
hortDescription"
     mslp impliesType
_"http://www.soa4all.eu/ontologies/weatherForecast#Pressure_hPa"
     temperature impliesType
_"http://www.soa4all.eu/ontologies/meteorology#Temperature_Centigrades"
     relh impliesType
_"http://www.soa4all.eu/ontologies/weatherForecast#Humidity_Percentage"
     prcp impliesType
_"http://www.soa4all.eu/ontologies/weatherForecast#Precipitation_mm"
     wind impliesType
_"http://www.soa4all.eu/ontologies/weatherForecast#Wind_Knots"
     summary impliesType _string

concept endpoint
```