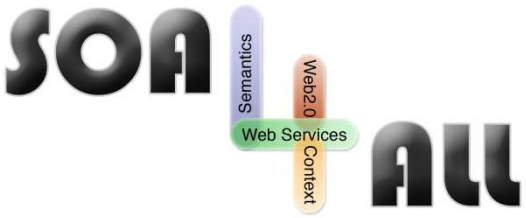

Project Number: **215219**

Project Acronym: **SOA4ALL**

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

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D6.4.2 Advanced Prototype For Service Composition and Adaptation Environment

Activity 2:	Core Research and Development
Work Package 6:	Service Construction
Due Date:	M24
Submission Date:	28/02/2010
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	ATOS
Revision:	1.0
Author(s):	Yosu Gorroñogoitia, Matteusz Radzimski (Atos), Freddy Lecue (UNIMAN), Matteo Villa, Giovanni di Matteo (TXT)
Reviewers:	Sven Abels (TIE), Gianluca Ripa (CEFRIEL)

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	25/11/2009	ToC	Yosu Gorroñogoitia (ATOS)
0.2	03/02/2010	First contributions: section 2, Annexes	Yosu Gorroñogoitia, Mateusz Radzimski (ATOS), Matteo Villa, Giovanni di Matteo (TXT), Freddy Lecue (UNIMAN)
0.3	08/02/2010	Common sections. Main contributions. Internal review comments	Yosu Gorroñogoitia, Mateusz Radzimski (ATOS), Matteo Villa, Giovanni di Matteo (TXT), Freddy Lecue (UNIMAN)
0.4	15/02/2010	Final contributions. All sections completed.	Yosu Gorroñogoitia, Mateusz Radzimski (ATOS), Matteo Villa, Giovanni di Matteo (TXT), Freddy Lecue (UNIMAN)
0.5	22/02/2010	Peer Review	Sven Abels
0.6	24/02/2010	Peer Review	Gianluca Ripa
1.0	24/02/2010	Final editing	Yosu Gorroñogoitia

Table of Contents

EXECUTIVE SUMMARY 7

1. INTRODUCTION 8

1.1 PURPOSE AND SCOPE 8
1.2 STRUCTURE OF THE DOCUMENT 9

2. DESIGN TIME COMPOSITION ENVIRONMENT. 10

2.1 COMMON ARCHITECTURE AND VIEW 10
2.2 TEMPLATE GENERATOR 14
2.2.1 Requirements and Functional Specifications 14
2.2.2 Theoretical grounding 17
2.2.3 Implementation architecture 22
2.3 DESIGN TIME COMPOSER 24
2.3.1 Requirements and Functional design. 24
2.3.2 Theoretical grounding 27
2.3.3 Implementation architecture. 29
2.4 OPTIMIZER 37
2.4.1 Requirements and Functional Specifications. 37
2.4.2 Theoretical grounding 40
2.4.3 Implementation architecture. 42
2.5 DTCE PROTOTYPES IMPLEMENTATION ROADMAP. 45
2.5.1 Template Generator Implementation roadmap 45
2.5.2 Design Time Composer Implementation roadmap 46
2.5.3 Optimizer Implementation roadmap 47

3. CONCLUSIONS 48

4. REFERENCES 49

ANNEX A. INSTALLATION AND CONFIGURATION 52

ANNEX B. DTCE EXPERIMENTS 58

List of Figures

Figure 1Architecture view for Design Time Composition Environment 10
Figure 2 PE-DTCE-EE Interactions 13
Figure 3: functional integration of the Template Generator 16
Figure 4: an example of a schema generated by the TG from the Process Editor 17
Figure 5: The mining approach 19
Figure 6: standardised representation of the context framework 20
Figure 7: context-driven logs filtering 21

Figure 8: Template Generator internal architecture 22
Figure 9: Integration of the TG with SOA4All Process Editor 24
Figure 10 DTC Architecture 30
Figure 11 DTC design models taxonomy 37
Figure 12 A Composition with no Service Binding 41
Figure 13 - Template Generator GUI invocation 58
Figure 14 - MXML input ready to be analysed 59
Figure 15 - Generated template schema tree and selected schema representation 60
Figure 16 – WP7 input registration process for unrestricted payment. 61
Figure 17 – WP7 output registration process for unrestricted payment. 61
Figure 18 – WP7 input registration process for credit card payment. 62
Figure 19 – WP7 output registration process for credit card payment. 62
Figure 20 – WP9 input update catalogue process. 63
Figure 21 – WP9 output update catalogue process. 63
Figure 22 – WP9 manual amended update catalogue process. 63
Figure 23 – WP9 output update catalogue process. 64
Figure 24 – WP9 output update catalogue process. 64
Figure 25 – WP9 manual amended update catalogue process. 65
Figure 26 – WP9 update catalogue process with some data flow connectors. 65
Figure 27 Illustration of a WP9 Domain Ontology 66
Figure 29 Non-Optimal Composition 67
Figure 28 Illustration of two WP9 WSML services 67
Figure 30 Initials Services and Semantic Links Binding 68
Figure 31 Final and Optimal Services and Semantic Links Binding 68
Figure 32 Optimal Composition 68

List of Tables

Table 1Template Generator implementation roadmap 45
Table 2 DTC implementation roadmap 47
Table 3 Optimizer implementation roadmap 47

Glossary of Acronyms

Acronym	Definition
DTCE	Design Time Composition Environment
LPML	Lightweight Process Modelling Language
SWS	Semantic Web Service
BP	Business Process
PE	Process Editor
GUI	Graphical User Interface
HCI	Human Computer Interaction
LPM	Lightweight Process Modelling
TG	Template Generator
DTC	Design Time Composer
EE	Execution Environment
DSB	Distribute Service Bus
DSL	Domain Specific Language
I/O	Input/Output
MXML	ProM eXtensible Markup Language
API	Application Programming Interface
GWT	Google Web Toolkit
SOA	Service Oriented Architecture.
WS	Web Service
BPM	Business Process Modelling
WSML	Web Service Modelling Language
WSMO	Web Service Modelling Ontology
NFP	Non Functional Properties
DMA	Design Modification Agent
DAA	Design Analysis Agent
SD	Service Discovery
SLO	Semantic Link Operator
BBCA	BlackBoard Control Agent
IoC	Inversion of Control

DRL	Drools Rule Language
SAWSDL	Semantic Annotations for WSDL
WSDL	Web Services Description Language
URL	Universal Resource Location
FC	Functional Classification
URI	Universal Resource Identifier
JAX-WS	Java API for WS
QoS	Quality of Service
IP	Integer linear Programming
GA	Genetic Algorithm
IT	Information Technologies
C2C	Customer to Customer
CSOP	Constraint Satisfaction Optimisation Problem
CSP	Constraint Satisfaction Problem
NP	Non-Deterministic Polynomial-time
DL	Description Logic

Executive summary

Modelling context-aware adaptive business processes by reusing and aggregating Semantic Web Services (SWS) comprises a very complex engineering life cycle. Most of the business analysts and modellers have not previous background and enough expertise in SWS composition techniques, whereby they must rely on ICT skilled service modellers or integrators. This approach impedes business modellers and other average Web end-users to create directly optimal SWS compositions that reify their own business processes. However, the inherent complexity of the business process modelling (using SWS) cannot be easily suppressed, even by using front-end editors[1].

In SOA4All, we are aiming to move this complexity from the front-end SOA4All Studio used by business modellers to the back-end SOA4All platform services, which will take care of the modelling complexity, discharging modellers from complex modelling tasks, extracting domain specific and contextual knowledge from different sources.

This document describes the technical implementation of the advance Design Time Composition Environment (DTCE) prototype, which comprises those aforementioned SOA4All platform services that assist business modellers who are using, at design time, the SOA4All Studio to created adaptive, context-aware, optimized business process, by reusing existing process knowledge acquired from the analysis of previous process executions.

The DTCE consists of the following tools. **Template Generator** infers abstract BP model templates from the analysis of process past executions and lets modellers start from a BP template, but not from scratch. **Design Time Composer** assists end-users of the Process Editor to complete BP models with further details and helps in solving complex modelling tasks. **Optimizer** computes optimal BP models by suggesting optimal (concerning QoS and semantic quality metrics) set of bound services to BP activities. The DTCE tools are seamlessly integrated through a common Lightweight Process Modelling Language (LPML) [12] and associated methodology, and through a common access point, the Studio PE.

DTCE tools provide implementation support to some of those methodological ideas underlying the LPML that were gathered in [12]. By contrast, the implementation support for LPML methodology at runtime is provided by the Execution Environment, which is reported in [14].

This document focuses on the description of the implementation of DTCE tools, which motivation, functional analysis and theoretical background was introduced in [13]. Additionally, this document provides testing and experimental examples, aiming to validate their usage.

[1] All industrial available BPEL4WS Editors and Suites requires a deep WS/WSDL knowledge.

1. Introduction

1.1 Purpose and Scope

This document provides a detailed functional, technical and implementation description of the prototype tools constituting the Design Time Composition Environment (DTCE). The motivation, theoretical background and preliminary analysis for DTCE were introduced in [13]. DTCE tools provide implementation support, during the design time phase, to the methodological ideas underlying the Lightweight Process Modelling Language (LPML) gathered in [12]. The implementation support for LPML methodology at runtime is introduced in [14].

The DTCE tools are implemented as platform services within the SOA4All infrastructure. They work, seamlessly integrated with the SOA4All Studio, concretely, the Process Editor and other platform services (e.g. Service Discovery, Reasoner, Execution Environment, SWS registry, etc). The DTCE tools aims at assisting human modellers when creating business process (BP) models using the SOA4All Studio (i.e. Process Editor), providing a comprehensive and coherent support at design time. BP modelling is a complex task that cannot be undertaken by non-skilled modellers without some semi-automatic assistance. DTCE tools assist modellers using Studio Process Editor and simplify some time-consuming and error-prone modelling tasks during the complete modelling phase and more importantly, it supports the main lightweight modelling principles. DTCE features techniques such as [12]:

- Intensive reuse of process templates and fragments gathered from the analysis of previous process executions.
- Coarse-grain description of activities and other process elements (e.g. gateways and flow conditions) based on light semantic annotations.
- Adaptable (based on context) set of services bound to activities, which can be selected at runtime.
- Process knowledge extraction from domain-specific and contextual model sources
- Full optimization of process global response function based on non-functional properties and semantic quality metrics, etc.

The DTCE consists of the following tools. **Template Generator** infers abstract BP model templates from the analysis of process past executions and lets modellers start from a BP template, but not from scratch. **Design Time Composer** assists end-users of the Process Editor to complete BP models with further details and helps in solving complex modelling tasks. **Optimizer** computes optimal BP models by suggesting optimal (concerning QoS and semantic quality metrics) set of bound services to BP activities.

This document focuses on the description of the implementation of DTCE tools, which initial functional analysis was introduced in [13]. Nonetheless, it complements it, introducing additional functional and theoretical analysis of new prototype features where necessary, in order to improve the understanding. Additionally, this document provides testing and experimental examples, aiming to validate their usage.

1.2 Structure of the document

The rest of this document is structured as follows. Section 2.1 introduce structural and behavioural views of the DTCE and shows an integrated and overall picture of DTCE tools in the wider SOA4All context. Section 2.2 describes the functional, technical design and implementation details of Template Generator tool prototype. Section 2.3 describes the functional, technical design and implementation details of Design Time Composer tool prototype. Section 2.4 describes the functional, technical design and implementation details of Optimizer tool prototype. Section 0 summaries the implementation roadmap for each DTCE prototype. Section 0 outlines the main deliverable conclusions. Annex A provides the installation and configuration guide for each DTCE tool prototype. Annex B describes some experiments performed using DTCE tools in the context of SOA4All case studies scenarios.

2. Design Time Composition Environment.

2.1 Common Architecture and View

This section outlines the common architecture and views (structural, behavioural) of DTCE, comprised of **Template Generator, Design Time Composer and Optimizer**. DTCE architecture was depicted within the overall context of SOA4All architecture in [13] and [1]. A more developed description is given in this section. For completeness and coherence, this view also includes other SOA4All Studio components and SOA4All platform services that participates in the common SOA4All methodology concerning to service construction.

The next figure shows the Design Time Composition Environment architecture view.

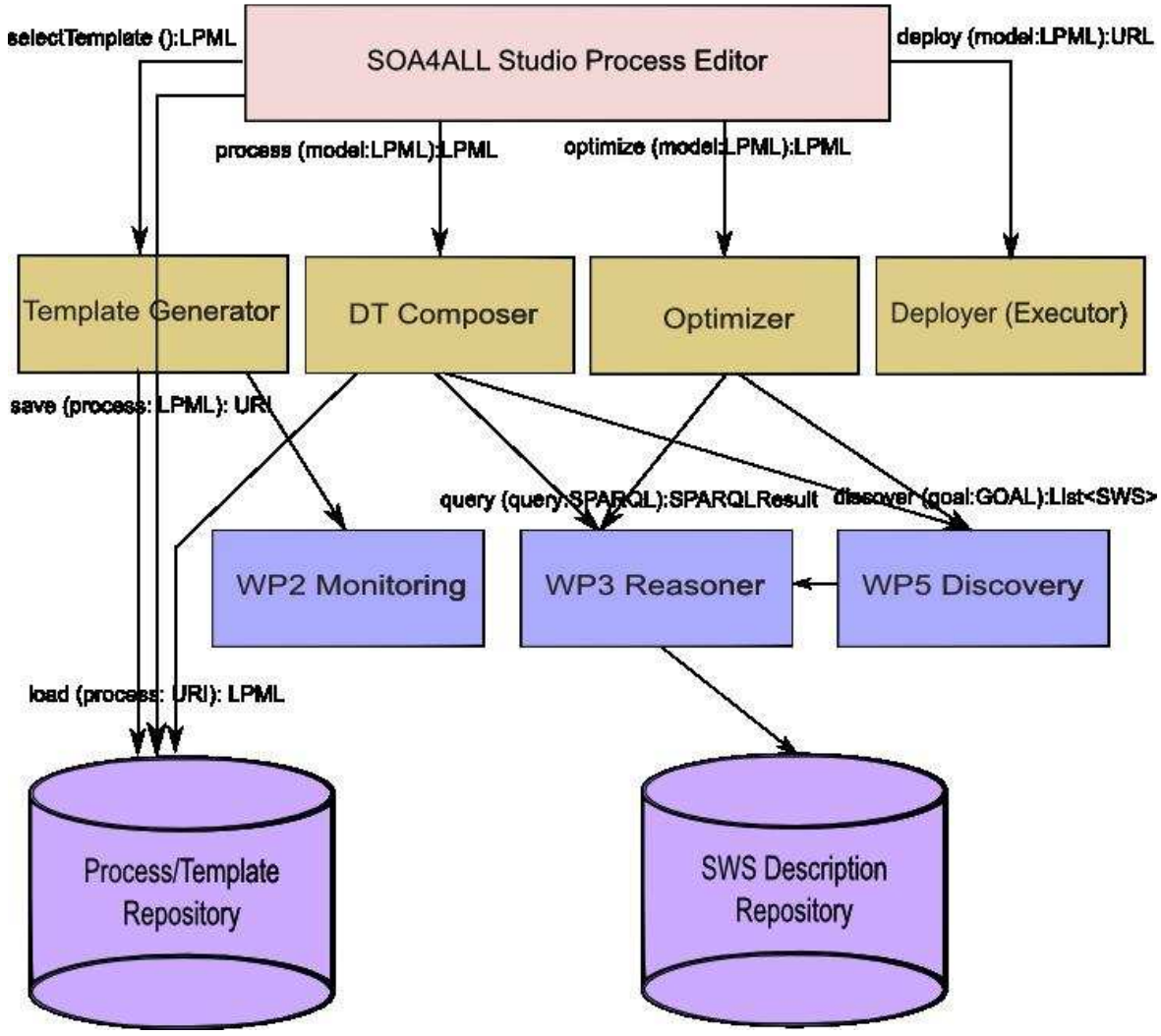

Figure 1Architecture view for Design Time Composition Environment

In this architecture view, the SOA4All Studio Process Editor (PE) provides a common Graphical User Interface (GUI) that supports the whole SOA4All lightweight process modelling lifecycle. PE provides an interface between human modellers and the SOA4All

Design Time Composition Environment (DTCE) tools described in this deliverable. Other platform services involved are Service Discovery, Reasoning, Monitoring and Management Infrastructure, Process/Template repository, SWS description repository.

TG, DTC and Optimizer are developed within T6.4 and reported in this document. PE is developed in T2.6 and reported in [4]. Deployer is developed in T6.5 and reported in [14]. Service Discovery is developed in T5.3 and reported in [10]. Reasoner is developed in T3.2 and reported in [7]. Monitoring is developed in T2.3 and reported in [6]. Repositories are implemented by T1.3 and reported in [1].

The role of each component depicted in Figure 1 and participating in the common processes of the Design Time Composition Environment is the following:

The Process Editor (PE) supports the main Human-computer Interaction (HCI) between modellers and the SOA4All Design Time Composition Environment. Besides, PE supports the complete manual LPM modelling life cycle.

The Template Generator (TG) generates process abstractions (templates) from the analysis of past execution logs and populates the Process/Template repository.

The Design Time Composer (DTC) provides partial automatic process model generation, completing missing LPML information that is obtained from different knowledge sources.

The Optimizer improves the global cost function of a LPML process model by replacing the non-optimal (according to some NF properties) bound service set by an optimal one.

The Deployer[2], that is part of the Execution Environment (EE), receives the final LPML process model created in the PE and deploys it within the EE [14].

The Service Discovery provides lookup facilities to some DTCE tools, returning a set of SWS (out of those stored within the SWS repository) that match particular search criteria (goal).

The Reasoning provides some query facilities to some DTCE tools to inspect SWS descriptions.

The Monitoring and Management infrastructure provides past-execution logs to the TG.

The Process/Template repository is a common repository of LPML processes and templates and some DTCE tools use it indirectly to interoperate each other.

The SWS repository is a common SWS description repository that contains the SWS descriptions used extensively by some DTCE tools.

TG provides a GUI embedded within the PE. TG is used by PE modellers to select concrete past-execution logs, generate process abstractions (templates) and taxonomies, refine manually those templates and use them during the process-modelling phase, usually as initial draft process models. Past-execution logs are provided by the Monitoring and Management infrastructure. LPML templates created by TG are stored within the templates repository. Modellers can retrieve them from the same template repository using the PE.

[2] Deployer is developed by T6.5 and described in [14]. Conceptually can be considered as part of the DTCE since it is invoked by the PE at design time, but considering that is reported as part of the Execution Environment, we keep this approach.

Normally, domain-experts modellers open them and refine them for further use. All process and templates models created or edited by the PE are stored/retrieved into/from that process/template repository.

PE interacts iteratively with DTC. Human modellers mediate in this iterative procedure by validating and amending the LPML process models returned by DTC. Amended LPML processes can optionally be sent again to the DTC for further process model refinement, until the process model is considered completed. DTC reuses intensively when possible the process templates stored within the common process/template repository and the SWS descriptions kept within the SWS repository.

PE interacts with Optimizer once the process model has been completed in the PE and it requires to be optimized with respect to some NF properties and functional qualities of semantic connections between services. The returned optimized model is the ultimate outcome artefact of the DTCE lifecycle.

All interactions between PE and DTCE tools (excepting TG) are mediated through the Distribute Service Bus (DSB) [3]. PE exchanges directly LPML models with DTC and Optimizer. Therefore, the process/template repository is not used to mediate the interactions between those components.

PE interacts with the Deployer in order to install and activate a LPML process model within the EE. Once deployed, the process model is ready to receive invocations.

LPML process models comprise activities described by annotations. DTCE tools, such as DTC and Optimizer, use these annotations to create querying criteria that can be matched against some knowledge sources, such as the SWS and process templates descriptions, in order to select suitable ones that could resolve/bound LPML process activities. SWS descriptions are created using SOA4All Editors. Process template descriptions are implemented as instances of the same SOA4All Service Model as well. Furthermore, they can be created and stored within the repository of SWS descriptions by: a) the SOA4All Process Editor, b) manually or by using any third party tool capable of creating SOA4All SWS descriptions. Additionally, the DTC may support other domain specific template knowledge descriptions (for instance the Domain Specific Language (DSL) format described in section 2.2.3.1). Those are the template sources used by DTC to resolve activities. Additionally, DTCE tools can further inspect SWS and process templates descriptions. The Service Discovery and the Reasoner platform services support these procedures performed by DTCE tools.

The Figure 2 provides a behavioural (sequential) description of the interactions between the PE, DTCE platform services and the EE. To simplify the picture, only interactions between directly involved SOA4All service construction components (PE, DTC, Optimizer, Deployer/Executor) are depicted.

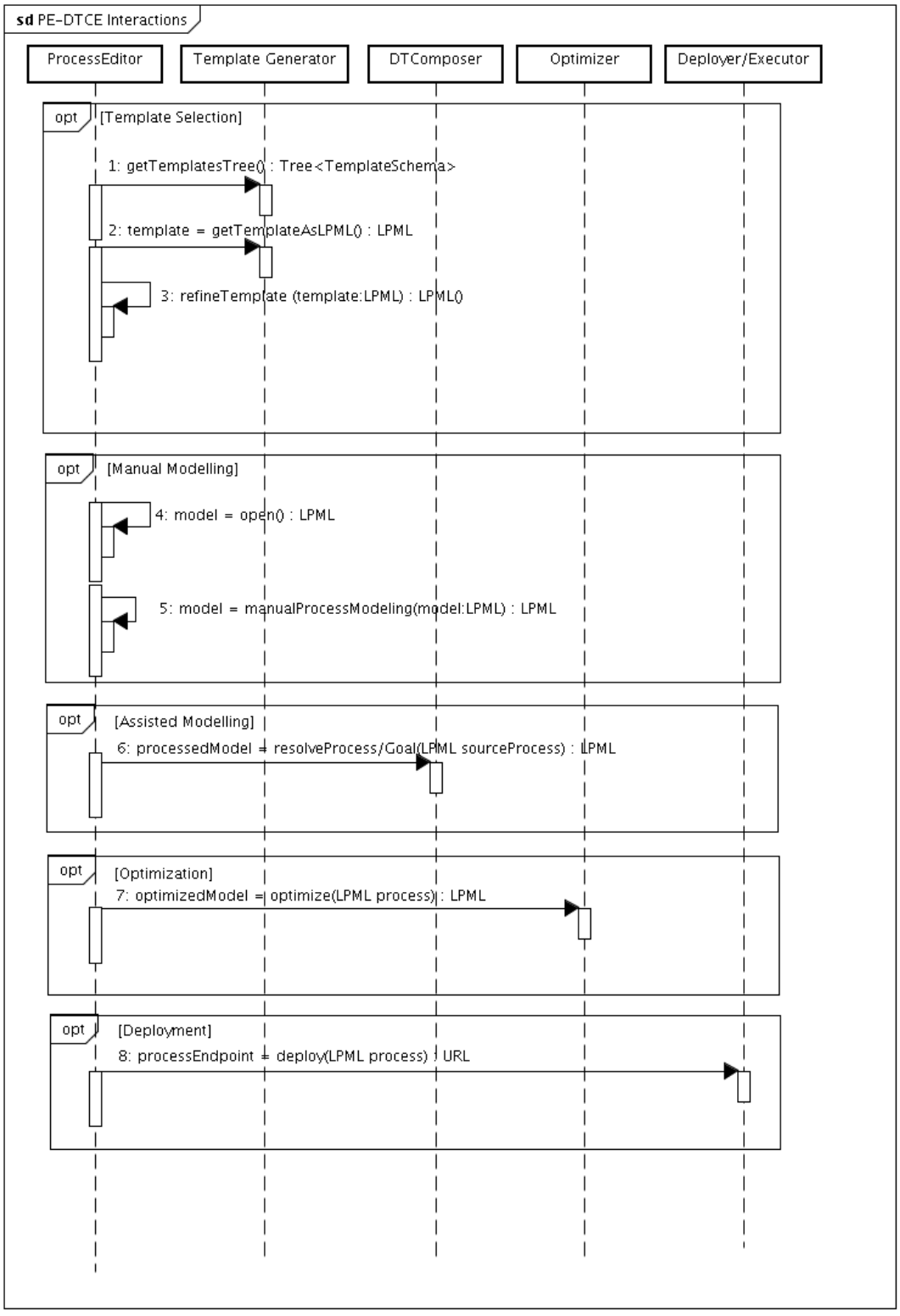

Figure 2 PE-DTCE-EE Interactions

Summarizing, PE can optionally interact with any SOA4All service construction platform service, normally following this sequence:

- Modellers can use the PE to browse and open templates stored within the Template Repository. Modellers can optionally use the TG in order to generates a draft process template. Once the process template opens in the PE, the modeller can: a) refine it as a template, b) start modelling a business process out of it.
- The DTC is optionally and iteratively invoked to assist users to model business processes, either binding concrete SWS to unbound activities, expanding unbound activities with process templates, check I/O semantic compatibility, creating Data Connectors, etc.
- The Optimizer is optionally invoked to optimize executable process models (all activities bound to SWS), replacing underperforming SWS with optimal ones.

The Deployer is optionally invoked to deploy a final executable process model within the EE.

2.2 Template Generator

In this section, we describe the advanced prototype of Template Generator component, extending the initial description provided into [13].

2.2.1 Requirements and Functional Specifications

2.2.1.1 Motivations and Requirements

Explicit process modeling can be a difficult and expensive task, especially in those situations where either an a-priori model is unknown or the effort to create the model is too complex

Process Mining techniques aim at automatically discovering a process model, based on data gathered during its past executions (logs). These techniques help users to:

- design and optimize concrete workflow models
- better comprehend the process behavior – by deriving the real run-time schema of a previously designed process

Most of existing state-of-the-art approaches are devoted to identify a single process formalisation, often resulting in particularly complicated schemas and not very accurate (single schema for all possible executions). The resulting schema, even if formally complete and adequate to support a process execution, turns out to provide little help to let end-users understand what the hidden process schema.

This side effect can be further amplified in the context of SOA4All*, due to the large number of potential users, services and process executions*

On the other hand, SOA4All is not targeting only modeling specialists: users are not expected to have in-depth modelling capabilities, so they need to be supported by a simpler and more abstract schemas.

This effect is again amplified when the schema is formed by services whose names may not be self-explicative, or at least out of the end-users context: a more intuitive and self-

explanatory information is required.

Finally, most of mining approaches are suitable for a typical intra-company scenario, where processes, activities, user roles are quite well defined. Within SOA4All users will belong to different environments (including but limited to business-like environment), and they will be accessing a much wider range of services. Thus the problem of which logs should be analysed in order to derive process schemas should be addressed by the Template Generator.

2.2.1.2 Functional Specifications

Due to such motivations, the Template Generator aims at supporting end-users in both the initial design phase of a process and in the phase of the analysis of its run-time behaviour after execution by:

1. deriving not a single complex schema, but rather an hierarchy of process schemas at different levels of complexity and completeness;
2. deriving a taxonomy of possible process templates at different level of abstraction. Such abstract activities will be mapped to SOA4All LPML abstract activities (cfr. [12]);
3. presenting such taxonomies of schemas to end-users in an interactive graphical way, and let them choose the most suitable one;
4. exploiting SOA4All semantic service descriptions (whenever available) in order to facilitate users' understanding on the process schemas: instead of the service URI, which may be not self-explanatory, annotations on service name will be shown;
5. exploiting SOA4All context information (whenever available) in order to select the most suitable logs sets to be analysed;
6. integrating it within the SOA4All Process Editor.

An example of this approach is reported in Annex B.

The TG is intended to be part of the typical process of designing and running SOA4All processes. The following picture illustrates how it logically relates to the other SOA4All components of WP6 and WP2:

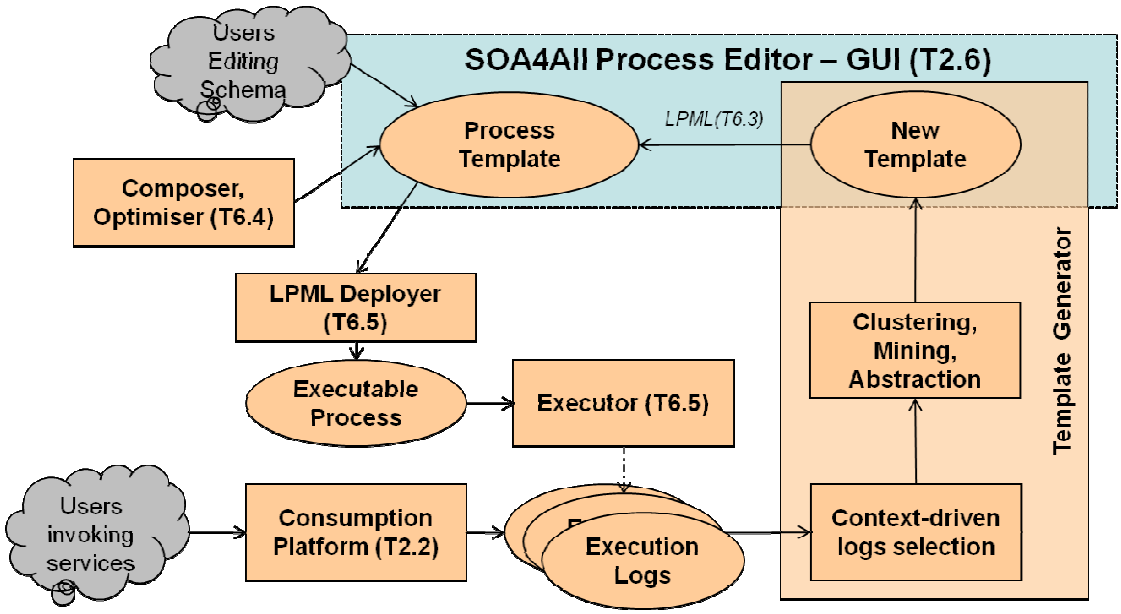

Figure 3: functional integration of the Template Generator

1. The TG provides a user-interface accessible via the SOA4All Process Editor (task T2.6).
2. Execution logs to be analyzed are generated either by the Consumption Platform (T2.2), in case of users invoking single services, or by the Execution Engine (T6.5), in case of a process being executed.
3. Process templates generated by the TG are made available in LPML format (task T6.3), so that they can be further refined or modified by Process Editor users or optimized by the other T6.4 tools (Composer, Optimizer).
4. Process schemas can then be transformed into the executable format (as described in [14]) and be executed by the executor. This closes the loop.

Next picture shows an example of the output of the Template Generator from the Process Editor environment:

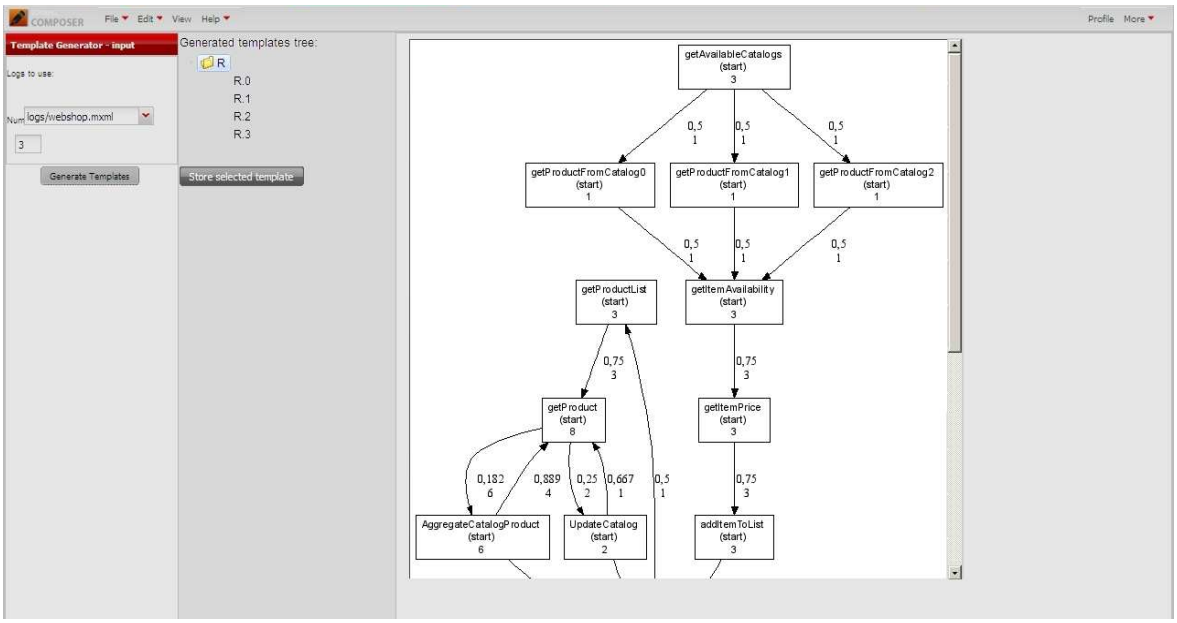

Figure 4: an example of a schema generated by the TG from the Process Editor

2.2.1.3 Expected Benefits in use Cases

As shown, the TG can be used either to generate an initial schema or to verify the actual run-time behavior of an existing process.

Both situations may occur in the SOA4All use-cases:

- **WP7 (process composition):** to support process analyst in the formalisation of the Public Administration processes. An initial schema can include those activities that are currently being used by the PA. Modellers can then complete the process schema by adding new required services.
- **WP9 (process intelligence):** through the use of *Template Generator*, the Web shop owner (*Nada*) can study and analyse the actual behavior of buyers, by abstracting a typical purchase process out of several interactions. She can use this information to change the design of her processes, and make them closer to real selling processes.

2.2.2 Theoretical grounding

This section summarizes theoretical grounding implemented by the advanced Template Generator prototype.

Template Generator is built upon state-of-the-art mining techniques, facing new challenges when applied to the web of services world. More in particular, three main technical and theoretical challenges are addressed:

1. How to transfer the hierarchy mining approach, based on the use of the ProM suite[3] and two additional plug-in[4], to a web-based environment like the SOA4All Process

[3] http://prom.win.tue.nl/tools/prom/

[4] http://staff.icar.cnr.it/wfmining/tools.htm

Editor and integrated with SOA4All technologies and standards.

2. How to exploit context information to select/filter the input log set.
3. How to exploit the availability of semantic annotations on services, and the possibility to define “abstract” activities, eventually resulting in a specific process template described with SOA4All LPML language.

The following paragraphs provide some more insight on these elements.

2.2.2.1 Hierarchy based mining

Hierarchy-based mining approach was described into [13] at a very detailed level, and is based on the work of [15]. Here we present a summary of the most relevant features.

The mining approach is composed of the following steps:

1. ***Preliminary Workflow Discovery*.** As the first step, we generate a preliminary schema for the initial set of logs, using algorithms such as Alpha, Multi-phase, Genetic, Social Network..
2. **Logs Clustering thanks to Discriminant Rule Extraction methods**. The preliminary schema generated in this way is based on the whole set of logs. In order to detect and to separate meaningful execution scenarios into meaningful set (cluster), we will exploit Discriminant Rule Extraction and Log Clustering techniques (see [16]).
3. **Derivation of Process Schemas for the Clusters.** We have a new set of logs arranged into clusters, which can be interpreted as a possible different execution of the same process – in this way we are reducing the degree of complexity in the schema, but also the degree of completeness of the schema. Once again, such log clusters can be modelled with a specific workflow schema, using the same Process Mining techniques described in bullet 1)
4. **Nodes Refinement.** We can iterate this process and repeat steps 2 and 3 for each node we wish to refine, in order to obtain a final hierarchy of workflow schemas.
5. **Final Hierarchy Selection**. We present to the end-user the final hierarchy of schemas: leaves will constitute a disjoint set, which represents the initial log set in a more accurate and expressive way rather than the preliminary schema (root). It is up to the user to select the most suitable schema, based on the number of possible situations (i.e. different possible executions) he wants to take into account and based on the complexity of the schema.
6. **Process Abstraction**. In order to further improve our process we apply a process abstraction methodology, in order to re-structure the knowledge embedded in the various schemas of the hierarchy in a taxonomy of schemas at different level of abstraction. The resulting taxonomy is a tree where leaves describe real process instances and higher-level nodes represent an abstract view on heterogeneous process instances schemas. The abstraction technique replaces groups of homogeneous activities with a single, abstract activity (thanks to “is-a” and “part-of” relationships). Finally, end-users will be able to navigate such taxonomy and to visualise the various schemas, based on the abstraction degree they need

The following picture shows the whole process: from the initial set of logs, the schema

complexity is reduced by logs clustering into disjoint set (thus increasing the number of possible schemas) – as shown in the "Process Single schema" box in the picture. Then, abstraction techniques allow producing a taxonomy of schemas at different level of abstractions (upper part of the picture):

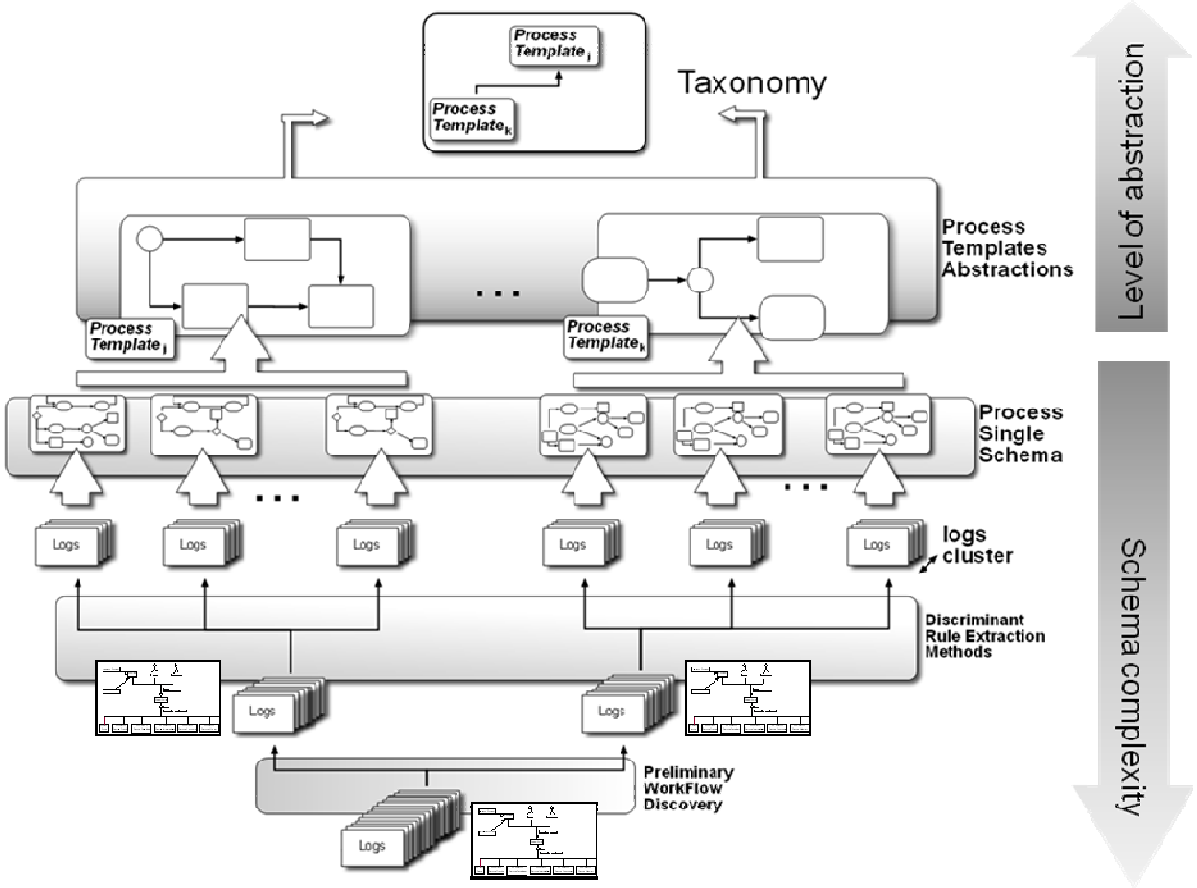

Figure 5: The mining approach

2.2.2.2 Context-driven logs selection

Process mining techniques, like the ones described in the previous section, are typically used in intra-company scenarios, where the notion of context is quite well established (i.e. a certain taxonomy on user, user-roles, enterprise processes and procedures).

SOA4All is anyway addressing also common internet-users, not necessarily belonging to specific organizations: the Consumption Platform (see [5]) allows for single services invocations, to both registered and anonymous users. Indeed these users could be actually following some process while invoking service in a logical sequence, or be part of a collaborative process with other internet-users, without even being aware of this fact, or without having the process structure formalized somewhere. So the problem of deriving relevant and useful process templates out of their services execution logs can still be applied.

On the other hand, due to the expected large amount of users and services invoked, and the fact they belong to different contexts, the problem of which logs the Template Generator

should analyse as input is much more complex. It is not possible to build a single hierarchy of schemas from all possible logs, as these include activities performed in different processes by different users, who may have no real connections amongst them.

Due to such reasons it appears necessary to adopt some contextual-driven filtering of input logs, so that only contextually coherent logs can be processed together by the TG. Unfortunately it is not possible to define an a-priori schema of such context, for the very reason that SOA4All is open to all users.

The approach followed by the Template Generator is therefore to exploit a flexible structure to represent context, like the framework introduced into deliverable [9] (par. 3.1.2) where "*contextual information is structured along a number of aspects or dimensions"*

Error! Reference source not found. provide a graphical representation of it.

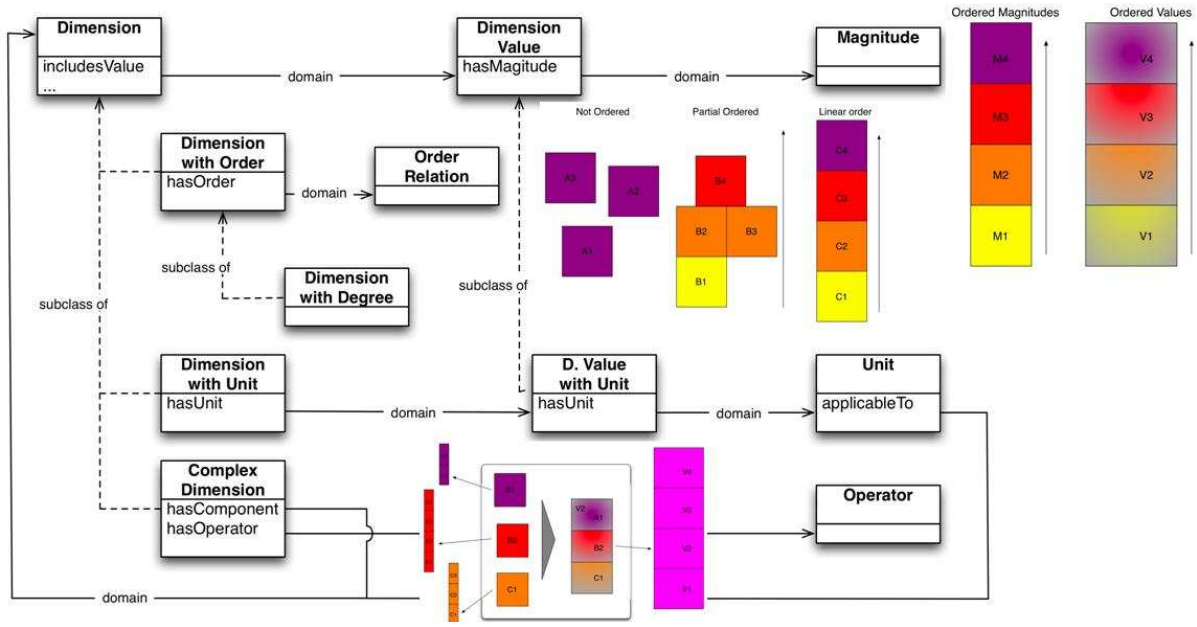

Figure 6: standardised representation of the context framework

Two things should be reminded (see [9], par. 3.3):

1. *"...a generic framework for the expression of contextual information has to be specified that provides a sketch and guidelines for specification of in application contexts"*

2. "...there needs to be a generic way of indicating contextually relevant information"

According to the framework applications have to play an active role in indicating the modelling resource used for capturing contextual information as well as directing the way this information is kept and managed.

This is precisely the approach that the Template Generator will follow, but with the specificity that now all dimensions of the framework could be defined a-priori, nor specific filtering criteria across each dimension can be pre-assigned. The Template Generator will rather allow to dynamically configure/customise such framework based on more specific "vertical" scenarios. TG users will be facilitated by a friendly user interface in the process of configuring the various dimensions and setting their units and thresholds.

The following picture summarizes the approach proposed:

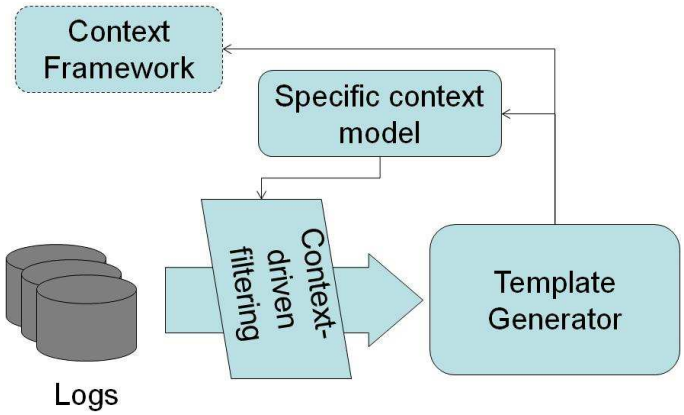

Figure 7: context-driven logs filtering

Indeed the operation of identifying most relevant dimensions is not expected to be simple or straight-forward, as it is a typical business intelligence task: this is why the approach proposed offers to the users all the necessary flexibility to dynamically study and extract the most relevant logs set, in order to eventually derive relevant process schemas.

2.2.2.3 Transformation to LPML

The third problem faced by the Template Generator is how to map the schema chosen by the end-user into a template describe with the SOA4All LPML. This in fact necessary in order to allow users to re-use or to edit the schema from within the Process Editor

More in particular, two facts should be considered:

1. TG users can select a schema which may contain abstract activities;
2. TG schemas include a semantic element (Service name), in order to facilitate users in the understanding of the process: this information should not be lost while translating to LPML.

LPML APIs, defined into [12], allow to manage both aspects, so the challenge is to define a mapping schema from one of the ProM formats to LPML elements.

The following table shows the basic mappings between the main TG data types and LPML elements.

TG local data format element	LPML element	Notes:
DependencyHeuristicsNet	Process	Hierarchy algorithm
LogEvent	Service	Hierarchy algorithm
WorkFlowSchema	Process	Abstraction algorithm

In TG local data format, each *LogEvent* object contains an array of entries including the identifiers of the following activities. Each one of these links is mapped in a LPML *Flow*

object.

Semantic annotations related to the single services of the process will be stored in the LPML *SemanticAnnotation* class and bound to the single *Service* elements.

In case of the abstraction algorithm, the *DependencyHeuristicsNet* object is transformed into a more complex structure: the *WorkFlowSchema* object, where abstraction mechanism is performed. Abstracted activities are stored as LPML *Activity* objects.

2.2.3 Implementation architecture

2.2.3.1 Internal Architecture

The Template Generator code is structured in two sections: **server-side** modules, implemented as GWT server-side services, responsible of the analysis and computation, and a visual **client-side module**, implemented as a Widget, responsible for user interaction: collecting user input, retrieving the selected set of logs for the analysis, displaying the results and storing them to the SOA4All common repository.

The TG core components have been implemented and deployed as **GWT server-side service**

The following picture shows the sub-modules of the TG:

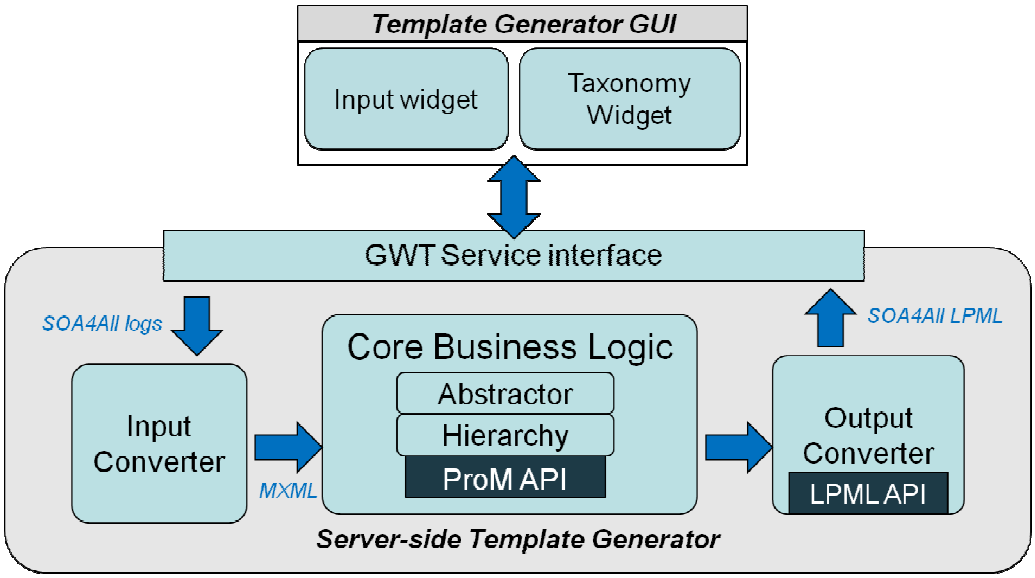

Figure 8: Template Generator internal architecture

Client-side components:

- **Input Widget** (GUI based): this widget is responsible for collecting user's input choices, such as context-driven selection of logs, mining algorithms input parameters as well as decision to store selected templates
- **Taxonomy Widget** (GUI based): this widget is responsible for visualizing the hierarchy of schemas that are produced by the TG

Server-side components:

- **Input Converter:** this module is responsible for retrieving the required logs from the SOA4All platform services, and converting them into ProM input format (MXML)
- **Core Business Logic**: this module is responsible for the main business logic of the schema generation process. It is built on top of ProM APIs, and it includes the Abstractor and Hierarchy plug-ins
- **Output Converter**: this module is responsible for converting the selected schema into LPML format, by exploiting LPML APIs. It is also responsible for storing the selected template in the SOA4All platform repositories (via LPML API)
- **GWT Service interface**: this layer makes available as GWT services the functionalities exposed by the other server-side components

2.2.3.2 API

The TG widgets are integrated in the SOA4All Process Editor. The main service interface class is:

org.SOA4All.processeditor.templategenerator.server.TemplateGeneratorService

Its implementation is:

org.SOA4All.processeditor.templategenerator.server.impl.TemplateGeneratorServiceImpl.

The API exposed are:

Tree<SchemaTemplate> **getTemplatesTree** (int algorithm, *Set<Log>* logs, int numChildren, int maxDepth)
Returns a org.SOA4All.processeditor.templategenerator.server.Tree of
org.SOA4All.processeditor.templategenerator.server.SchemaTemplate
@param algorithm internal algorithm id (0 = clustering; 1 = abstraction)
@param logs logset to be analysed
@param numChildren max number of children per node
@param maxDepth max depth of the generated tree
LPMLSchema **getTemplateAsLPML**(String templateId)
Returns the schema of a specified template as LPML schema
@param templateId id of the bewished template

2.2.3.3 Use of SOA4All Platform Services

The Template Generator needs to access various SOA4All Platform Services in order to:

- retrieve logs (via Monitoring Platform Service)
- retrieve semantic elements for services (via Reasoning Engine Service)
- map to LPML (via LPML API)
- store selected schemas (via LPML API)

2.2.3.4 Integrated View

The following picture shows how the Template Generator is integrated into SOA4All architecture:

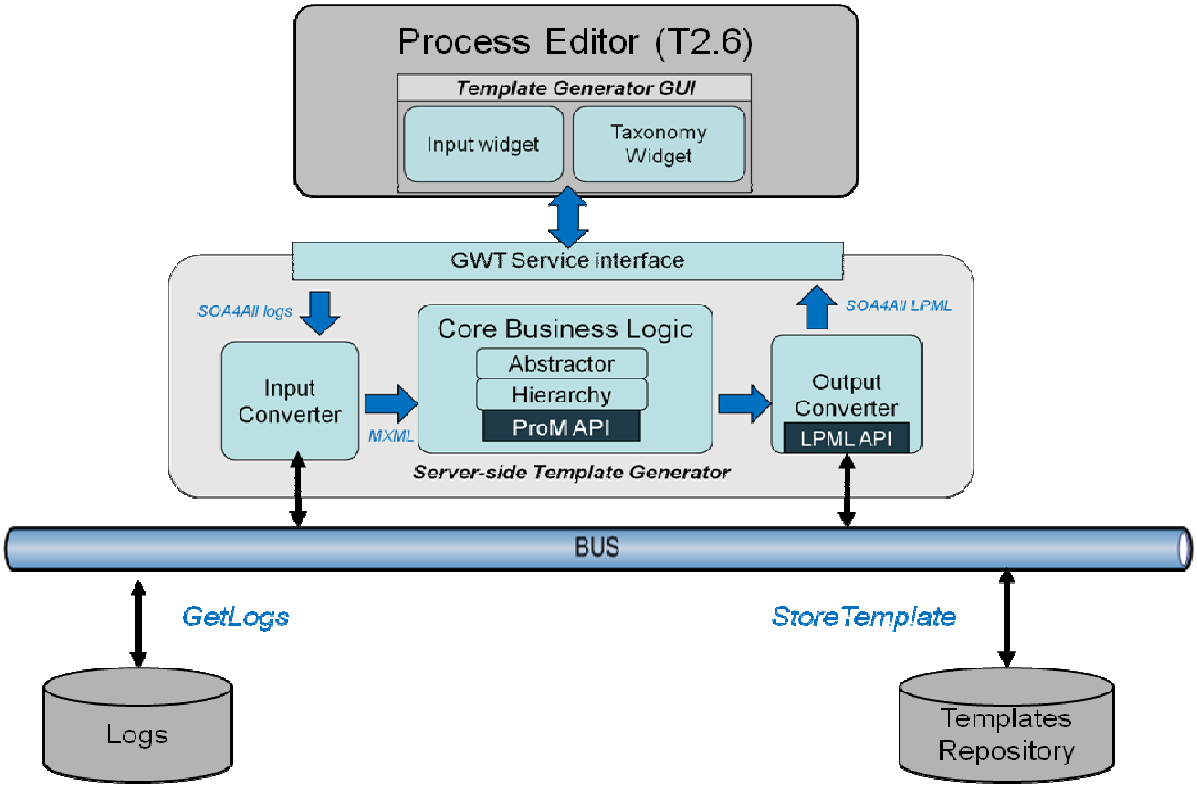

Figure 9: Integration of the TG with SOA4All Process Editor

2.2.3.5 Experiments and Testing.

TG has been tested in the modelling of WP9 scenario. These experiments are detailed in Annex B.

2.3 Design Time Composer

In this section, we describe the advanced prototype of Design Time Composer (DTC) platform service component, as an improvement of the first prototype described in [13].

2.3.1 Requirements and Functional design.

This subsection describes the features implemented in the advance prototype of DTC. Based on these features and others described in [13], but not implemented in the first prototype, the functional description of DTC is explained in some detail. Furthermore, the public DTC interface exposed to be consumed by the SOA4ALL Studio is described.

2.3.1.1 Motivations and requirements

The LPML methodology [11], [12] describes a new approach for the design-time phase of the BP modelling lifecycle in the SOA domain. LPML promotes some modelling principles: i) iterative, incremental, easy to use, semi-assisted BP modelling, ii) coarse-grained goal based activity-centric description of BP models, as opposed to the service-centric SOA composition approach, iii) semantically annotated activity descriptions (goals), iv) intensive reuse and customization of pre-existing domain specific process templates and fragments, v) context-aware BP model composition and adaptation. SOA4ALL tooling, concreted SOA4ALL Process Editor [4] and the DTCE platform services described in this document aim at supporting modellers to apply this methodology. In particular, the DTC platform service works integrated with the Process Editor, aiming at assisting iteratively modellers to complete executable BP LPML models, by fulfilling required model elements they left unfulfilled, with information extracted from different knowledge sources. DTC aims at featuring most of the LPML modelling principles.

2.3.1.2 Functional specification

DTC starts from an incomplete LPML model. It can include a set of activities, logically linked by a draft workflow. Each activity is coarse-grained described by its requirements (or by goals) that are expressed as a set of LPML annotations (i.e. references to semantic concepts defined within some shared domain specific ontologies). DTC returns a more elaborated LPML model which has resolved some of its information gaps: e.g. activities are bound to concrete WS, or expanded with BP templates or fragments, data flow is populated with connectors mediating between input/output (I/O) parameters, semantic compatibility between subsequent I/O parameters is checked, etc. The iterative LPML modelling approach, that DTC fosters, increases the level of concreteness of LPML models, closer to executable as opposed to abstract.

DTC applies registered knowledge about:

- Domain specific Process templates, fragments, available within a process model repository (see sections 2.1, 2.3.3)
- Web-scale SWS, accessible through SOA4ALL Service Discovery, (see [10] and section 2.1).
- Domain specific SWS descriptions, available within the common SOA4ALL SWS registry (see section 2.1 and [2][2]) or other sources.
- Context models, available within the common SOA4ALL space storage (see [2]) or other sources.
- Domain specific models, available within the common SOA4ALL space storage (see [2]) or other sources.

in order to resolve unassigned process model parameters, such as, unbound process activities, data flow connectors and so on:

- Replacing activities by matching process fragments and templates
- Binding them to concrete matching SWSs.
- Ensuring the semantic compatibility between subsequent (along the workflow) process activities in terms of I/O

- Creating Data Connectors between subsequent process activities: dataflow.

DTC platform service provides a public interface intended to be consumed mainly by the SOA4ALL Process Editor, but also compatible with any third party BPM editor capable to exchange LPML models.

DTC interface consists of three main groups of methods:

- Methods to resolve LPML models, which are processed with all DTC available knowledge. These methods receive one input LPML process model serialized as XML and return only one output processed LPML model, serialized in the same XML format (this XML serialization is supported by the LPML API, see [12]). Those methods can operate upon the whole process (resolveProcess<WithXXX>) or upon a concrete target process activity (resolveGoal<WithXXX>).

```
String resolveGoal (String sourceProcessModel, String activityTargetURI);
String resolveGoalWithTemplate(String sourceProcessModel, String activityTargetURI);
String resolveGoalWithWS(String sourceProcessModel, String activityTargetURI);
String resolveProcess(String sourceProcessModel);
String resolveProcessWithTemplate(String sourceProcessModel);
String resolveProcessWithWS(String sourceProcessModel);
```

- A similar set of methods, but returning a requested number of found process model solutions (see [13] for the parametric design taxonomy of process models):

```
String resolveGoalMS (String sourceProcessModel, String activityTargetURI, int
        numberRequestedSolutions);
String resolveGoalWithTemplateMS (String sourceProcessModel, String
        activityTargetURI, int numberRequestedSolutions );
String resolveGoalWithWSMS(String sourceProcessModel, String activityTargetURI, int
        numberRequestedSolutions);
String resolveProcessMS (String sourceProcessModel, int numberRequestedSolutions);
String resolveProcessWithTemplateMS (String sourceProcessModel, int
        numberRequestedSolutions);
String resolveProcessWithWSMS (String sourceProcessModel, int
        numberRequestedSolutions);
```

- A set of methods for supporting dynamic registering (hot-deployment) within the DTC service of additional agents and knowledge (see section 2.3.3 for DTC agents description), for concrete domains, in order to extend the DTC knowledge:

```
void registerDesignModificationRuleAgent (String uri, String knowledgePath);
void registerDesignModificationSemanticAgent(String identifier, String[] knowledgeOntologies,
String[] knownServices, String[] contextOntologies, String[] goals);
void registerSemanticLinkOperatorAgent (URI identifier);
void unregisterDesignModificationRuleAgent(String identifier);
void unregisterDesignModificationSemanticAgent (String identifier);
void unregisterSemanticLinkOperatorAgent (String identifier);
```

2.3.1.3 Expected benefits in use cases.

The combined assistance of Process Editor and DTC should simplify substantially the

modelling of executable business processes of the WP7-WP9 use case scenarios, starting either from scratch or from a process template generated by the Template Generator. We expect improvements in: a) reducing model development time, b) increasing model quality, c) increasing model completeness, d) simplifying the modelling process. Model development time is decreased since some modelling tasks are performed automatically by the DTC, therefore reducing dramatically the modelling time. Model quality is expected to be improved since DTC reuses existing well-refined process templates and fragments in order to complete the process model. Models are supposed to be more complete after each modelling iteration, since DTC can automate the fulfilment of some information gaps in model elements and data flow, dispensing modellers of providing that complex data. In general, the modellers perceive the modelling methodology easier, since it is guided by the Process Editor interface and by the iterative DTC support.

2.3.2 Theoretical grounding

This section complements the theoretical grounding implemented by the advanced DTC prototype, but not included in [13]. In particular, this section describes the mapping between the parametric design method described in [13] and the LPML methodology. This section also introduces the agents developed within current DTC prototype, although concrete details will provide in section 2.3.3.

DTC implements LPML assisted modelling features using a knowledge-intensive configuration process, more precisely a parametric design procedure [17]. In order to increase the scalability of this procedure, we extend the classical approach to its synthesis task by using an opportunistic approach, based on blackboard-based multi-agent system [18] [19]. Multi-agent architecture allows also for extra flexibility and extensibility with regard to management of knowledge bases used by DTC, by allowing for hot-plugging or updating knowledge while the system is operating. This can be done by adding new ontologies, services and templates descriptions, when registering new agents. Agents are autonomous, work in a collaborative manner on the common blackboards and each one is containing its own knowledge base (KB). The knowledge can be expressed using, for instance, WSML ontologies and WSMO-lite service models or rules with forward-chaining inference. These agents work altogether on resolving model activities binding them to appropriate services or by performing template expansion. In both cases, agents are using their own domain-specific knowledge for selecting the best match.

DTC converts the LPML modelling problem in a parametric design problem, where a LPML model is mapped into a design model that is formalized as a 8-tuple [20]

$$< P, Vr, A, C, R, DS, Pr, Cf >$$

where ***P*** is a set of unassigned parameters, which possible values ranges are represented by ***Vr***; ***A*** is the assignment set: a set of tuples $\{(p_i, v_{ij})\}$ that represent the values associated with each of the parameters; sets ***C***$(=\{c_1 \ldots c_N\})$ and ***R***$(=\{r_1 \ldots r_N\})$ represent the constraints and requirements that formalize the admissibility and suitability of a design; ***DS*** fixes the structure of the design to be configured; ***Pr*** describes the preferences and ***Cf*** the global cost function.

The mapping between a LPML model and a parametric design problem model is as follows. Parameter set, *P* is mapped into the set of unbound activities, whose values range, *Vr*, is either the range of available services or process templates. *P* may also include the I/O parameters of LPML activities, and the *Vr* are possible data connectors between I/O

parameters of activities connected by data flow.

The assignment set A is ultimately the set of concrete admissible and suitable services bound to a particular activity (including suitable operations). Alternately, unbound activities can be resolved by replacing them with matching process fragments. Those process fragments, in turn, may contain further unbound activities that require being resolved. Assignment set A may also include concrete data mapping connectors between I/O parameters of activities connected by data flow.

Constraints, *C*, and requirements, *R*, can be expressed either at LPML model or at a concrete activity level, in form of semantic annotations, in order to express limitations and desires on the requested functionality. Design Structure is given by the LPML workflow, but this structure is flexible and may change during the parametric design approach, in contrast to the common approach, due to the activity expansion using templates. Preferences *Pr* are expressed within LPML models as non-functional properties (NFP) while the Optimizer calculates the global cost function *Cf*. Neither preferences nor the global cost function is computed by DTC.

Parametric design procedure fulfils the assignment set *A*, assigning concrete values to the parameter set P, from those available out of the values range *Vr*, so that requirements are fulfilled but constraints are also met.

According to the nature of the found design model, parametric design procedure classifies solution design models those that are complete, admissible and suitable [20]. Furthermore, if the *Cf* is minimized, the solution is optimal.

DTC tries to find out solution design models for unassigned LPML models, while the Optimizer determines optimal solution design models. In default of found complete solutions, DTC returns found incomplete (suitable or not) admissible solutions. DTC could be invoked iteratively after analyzing and amending the returned LPML model with changes in the annotation sets of the activities included within the LPM model.

DTC implements the synthesis phase of parametric design procedure with a blackboard-based multi-agent system. Autonomous and specialized agents share a common backboard upon which they post new design models, which are modified versions of previously posted ones, after applying some specific knowledge.

Some agents, named as design modification agents (DMA), are specialized to introduce changes in the models, while other agents, named as design analysis agents (DAA) validate those changes. One BBCA coordinates the autonomous and independent agents.

At the beginning of the procedure, the blackboard is seeded with an initial LPML design model, whose assignment set is empty. At the end of the procedure, DTC returns none or several found solutions (completed assignment set). The procedure finalizes either when the requested number of solutions are found or when the DMAs can not post new design models.

DTC current prototype comprises four DMAs, which provides complementary approaches to resolve LPML activities using domain-specific and contextual knowledge:

- a Domain Specific Language (DSL) based DMA, which exploits DSL [21] descriptions of services and process templates according to concrete BPM requirements,

- a WSML DMA, which exploits domain specific WSMO or WSMO-lite annotated [22] descriptions of services and goals, according to concrete BPM requirements
- a Service Discovery (SD) based DMA, which binds activities to concrete services returned by a domain-independent Web-scale WSMO-Lite [8] discovery facility [10].
- a Semantic Link Design Operator (SLO) [13] based DMA, that establishes dataflow in the models.

DSL-based DMA exploits domain-specific knowledge concerning BP modelling building blocks: process fragments/templates and services described using adhoc domain-specific languages, easy to use by domain experts. WSML-based DMA and SD-based DMA are complementary approaches to exploit semantic description of services. WSML-based DMA consumes a relative small, targeted domain-specific knowledge, while SD-based DMA consumes the common domain-unspecific Web-scale SOA4ALL service registry.

Any DTC DMA works as a rule engine: for each LPML model activity, **A**, given the activity annotations or goal $\mathbf{A_G}$, DMA tries to apply a rule **R** when the rule condition $\mathbf{R_C}$ holds for $\mathbf{A_G}$:

$$R_C\ (\mathrm{A_G}) \rightarrow R\ (\mathrm{A})$$

Where the rule **R** could be as: i) add service **S** to the set of bound services, ii) replace activity **A** with template **T**, etc. Rules are stated explicitly (i.e. DSL-DMA) or implicitly (i.e WSML-DMA) in the DMA knowledge.

Any DTC DMA matches LPML activities (described by a goal **G**) with a candidate service, **S** or templates **T** when the semantic matching is exact [23] according to a specific domain ontology:

$$G \equiv S, G \equiv T$$

or plug-in (i.e. Subsumption relationships):

$$G \sqsubseteq S \vee G \sqsubseteq T$$

DTC DMA exploits only exact and plug-in matching between a LPML activity goal and a candidate service or template.

DTC incorporates other techniques to reduce the computational overload, such as :

- Detection of inadmissible design models D: $Admissible\ (D) \equiv \forall c_i \in C, |\neg c_i\ (D)$. DTC tags inadmissible design models avoiding other DMA may work upon them.
- Detection of identical posted design models D: $D_i = D_j \leftrightarrow \forall \left(p_i, v_i\right) \in A(D_i) \wedge \forall \left(p_j, v_j\right) \in A\left(D_j\right) \mid p_i = p_j \rightarrow v_i = v_j$. DTC ignores new identical design models posted into the blackboard.

2.3.3 Implementation architecture.

This section depicts the architecture and implementation details of the advanced DTC prototype. DTC is implemented as a SOA4ALL platform service as described in [2]. It is accessible by SOA4ALL studio and other SOA4ALL platform services through the DSB [3]. DTC architecture is shown in Figure 10. Detailed behavioral sequence diagrams can be found in [13].

The main DTC component, DTCImpl, is a service implementation of exposed IComposer interface. DTCImpl aggregates a set of specialized agents and one blackboard. Blackboard

is only accessible through the BlackboardControlAgent (BBCA). Other specialized agents are notified by BBCA every time a new design model, including the initial one, has been posted to the blackboard. Specialized DMAs are: Rule-based DMA, WSML-based DMA, SB-based DMA and SLO-based DMA. External platform services are invoked by specialized DMAs through service clients: ReasonerClient, SDClient, SLOClient and LPML API for storage access. Implementation details for each DMA are explained in next subsections.

The DTC core has been implemented using Spring framework IoC container [24], endowing DTC of beam configuration, management and observable pattern blackboard notifications for blackboard. All DTC agents, including the blackboard control agent have been implemented and configured as Spring beans.

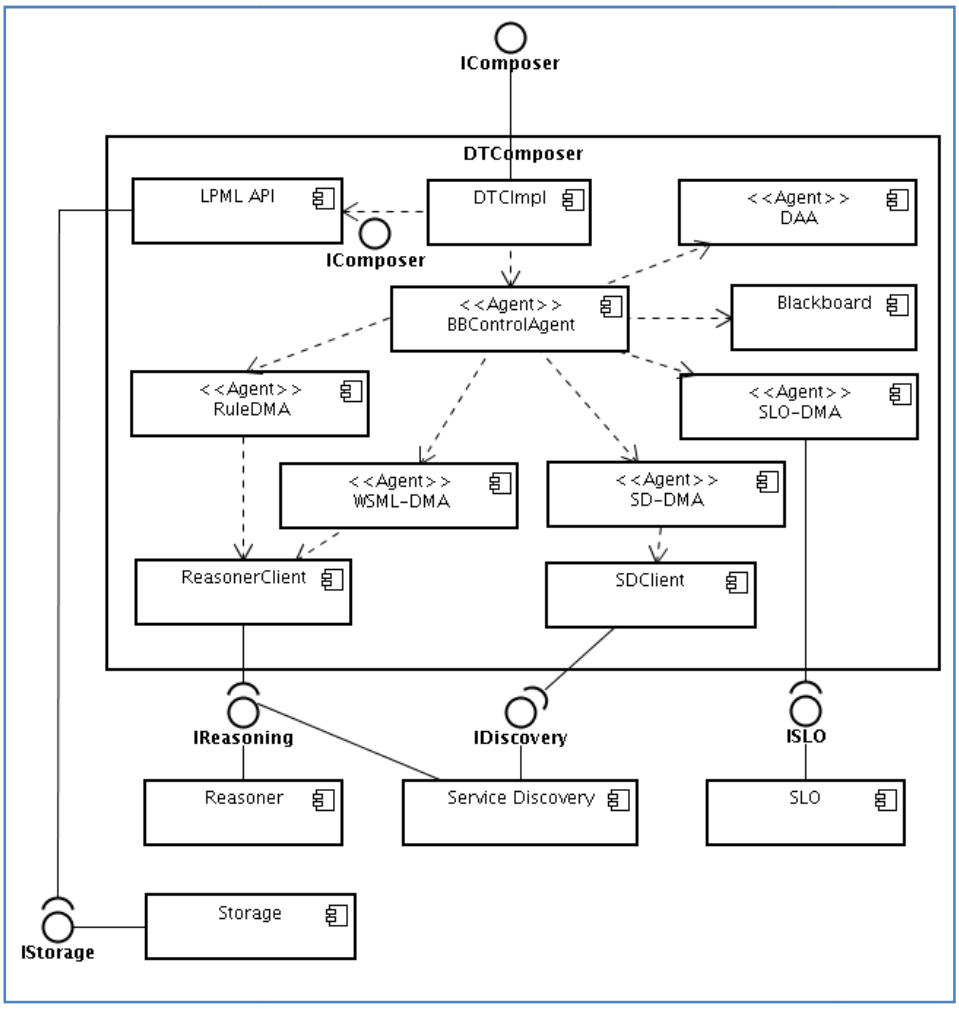

Figure 10 DTC Architecture

First DTC prototype implemented a preliminary version of the Rule DMA agent and the basic DTC architecture. Since then, the DTC prototype architecture has been re-defined and built. New agents have been introduced; SOA4ALL SD and Reasoner have been integrated.

Common DTC DAA has been implemented using JBoss Drools rule engine [25]. A specific Drools rule language (DRL) has been created for describing DAA rules, used to check the parametric design taxonomy of design models. The parametric design procedure taxonomy of design models have been developed using DAA DRL rules as follows:

```
rule "Analyze DM"
        when
                new design
        then
                check design is complete
                check design is admissible
                check design is suitable
                check design is io compatible
end
```

Next subsections provide details about the implementation of DMAs.

2.3.3.1 DSL-based DMA

DSL-based DMA are suitable to apply domain specific knowledge whose representation may vary from one domain to another. Within this DTC prototype, we have implemented an ad-hoc DSL used to describe available knowledge about known services and BP templates in a concrete domain, which is shown below:

```
template_def URL
        functionalClassification <URI> [1..*]
        input <URI> [1..*]
        output <URI> [1..*]
        global requirement <URI> [0..*]
        global constraint <URI>[0..*]
def_template
```

```
service_def URL
        functionalClassification <URI> [1..*]
        input <URI> [0..*]
        output <URI> [0..*]
        definition <URL>
        operation <name>
        global requirement <URI> [0..*]
        global constraint <URI>[0..*]
def_service
```

where:

- functional classifications (FC) are understood as described in the WSMO-Lite definition [8]. They are used, together with input/outputs, to match locally (at activity level) this template/service rule against the activity goal.
- I/O are model references of I/O parameters to ontology concepts, similar to the modelReference annotations in SAWSDL [26].
- global requirements and constraints (as defined by the parametric design procedure) are also model references to ontology concepts. They are globally matched against the global requirements and constraints defined at process level.
- definition points at the service definition, for instance, it contains a WSDL URL in

case of WSDL-based services.

- operation contains a valid operation name described within the service definition.

Activities within a LPML model are annotated with FC and I/O annotations, while requirements, R and constraints C are set at LPML model level. Then, a template **T** or a service candidate **S** matches the LPML activity goal $\mathbf{A_G}$ when:

$$Fc\ (A) \sqsubseteq Fc\ (S) \wedge\ I(S) \sqsubseteq\ I(A) \wedge O(S) \sqsupseteq O(A) \rightarrow S \in AS(A) \wedge R(S) \in R(D) \wedge C(S) \in C(D)$$

$$Fc\ (A) \sqsubseteq Fc\ (T) \wedge\ I(T) \sqsubseteq\ I(A) \wedge O(T) \sqsupseteq O(A) \rightarrow T \in AS(A) \wedge R(T) \in R(D) \wedge C(T) \in C(D)$$

$$Suitable\ (D)\ \equiv\ \forall\, r_i\ \in R \mid r_i(D)$$

$$Admissible\ (D) \equiv\ \forall\, c_i\ \in C \mid \neg c_i(D)$$

where activity A, service S or template T are defined by its functional classification Fc, input I and output O, AS(x) is the assignment set, R(x) is the requirements set and C(x) is the constraints set. According to above expressions, a service *S* is bound to a concrete activity *A* (or a template *T* replaces it) when: the *Fc* of *S* is subsumed by the *Fc* of *A*, input I of *A* subsumes the *I* of *S* and output *O* of *S* subsumes the *O* of *A*. Then, *S* is added to the list of services bound to the activity *A*, that is, the assignment set *AS*; the requirements and constraints attached to S are added to the set of requirements and constraints for the design model *D*, respectively. A similar approach is followed in case of template *T*. Afterward, the suitability and admissibility of the design model D is checked by a DAA.

DSL based DMA also uses JBoss Drools rule engine. DSL service/template definitions described above have been implemented using Drools DRL.

2.3.3.2 WSML-based DMA

WSML-based design modification agent is another class of agents that work on resolving business process described by goals into lightweight process composition that can be later executed. The idea of introducing such an agent is to enrich the composition phase by experimenting with injecting highly specialised knowledge-bases for particular domains. In such cases, there wouldn't be a need for querying general global repository with millions of services and applying filters for choosing most suitable ones, but rather a quicker and probably more precise answer would be served on the basis of highly relevant knowledge, engineered by some domain-specific expert. This is not by any means substitution for generic SOA4All Service Discovery that is handled by Discovery-Based DMA, but rather a way to enhance current approach with advantages that come from multi-agent problem solving methods [13], where agents specialised in some specific domain work together to provide a more accurate solution. Therefore, every agent has very narrow, but concrete specialisation. This approach can improve performance as only selected agents work on a process belonging to particular domain. In addition, the knowledge is distributed among many agents and flexibility and scalability can be easier to achieve. It is also easier for modeller to manipulate the knowledge, by just starting new agent with some domain specific KB, or withdrawing that knowledge by disabling particular agent.

WSML-based DMA works in a similar schema to every other DMA-class agents, by analysing goals in the LPML model and deciding if any specific knowledge can be applied. After some parts of the LPML process have been modified (knowledge was applied) a new

model is published, and other agents have possibility to further elaborate it.

Currently WSML-based DMA is working according to:

- WSMO-lite annotations of LPML activities, that describe: a) Functional Classification, b) preconditions, c) effects
- Process-level annotations: a)contextual information, b)requirements, c)constraints
- Reference to the WSMO Goal[5].

Goal resolving scenarios:

WSMO Goal matching

When certain LPML activity contains goal expressed using goalReference field of Goal class, we assume, that this activity was annotated with some predefined goal of given location (URI). In this scenario, we assume the WSMO Goal and matchmaking is performed against available services. Example of such goal is shown below (some information was removed for brevity):

```
goal WinterCatalogue1

capability WinterCatalogueCapability
        sharedVariables ?x
        postcondition WinterCatalogueCapabilityPostcondition
              definedBy
                     ?x memberOf ApparelSellingService and
                     ?x memberOf CatalogueUpdate and
                     ?x[season hasValue ?season] memberOf SeasonalCatalogue
                     and
                     ?season[month hasValue ?month] memberOf Season
                     and
                     ?month memberOf January.
```

The result is a list of WSMO services that provide Winter Clothes Catalogue.

WSMO-lite annotations matching

This matching scheme is preformed when activity's goal is described with WSMO-lite annotations (specifically: LPML SemanticAnnotations attached to certain LPML Activity). The knowledgebase used comprises WSMO-lite service descriptions and domain specific ontologies that provide extra information for better service selection.

The WSML-based agent can match goal with services based on WSMO-lite's FuntionalClassifications, Preconditions, Effects (and therefore it would behave in a similar manner to Discovery-based DMA), but can also act according to process' specific annotations (taking under consideration global Requirements and Constraints). To

[5] This functionality was introduced only temporarily due to lack of WSMO-lite support at the early stage of development and now is deprecated

accomplish that, agent is performing knowledge-intensive search, based on extra information provided by knowledge modeller in a form of WSML ontologies. Sample query to agent's knowledge-base is shown below:

```
extendedServiceMatch(<FCs,Requirements,Constaints,Preconditions,Effects>)
                                        ⇩
?x memberOf wsl#Service
and
?y[
      wsl#hasFunctionalClassification hasValue Payment,
      wsl#hasPrecondition hasValue ValidUserCredentials
      hasRequirement hasValue EveryCreditCard,
      hasConstraint hasValue Transactional,
      hasCorrespondingService hasValue ?x
] memberOf Agent012KB
```

The result is a list of WSMO-lite services that realise transactional payment with every credit card.

WSML-based DMA is implemented using wsmo-api, wsmo4j, wsml2reasoner [27]. For legacy WSMO support (WSMO Goals) it is also using WSMO-discovery module. The WSMO Lite annotation support is backed up by WSML ontologies, and WSMO Lite service description and employing IRIS as a reasoning facility.

2.3.3.3 Service Discovery based DMA

Service Discovery (SD) based DMA uses the SOA4ALL common Service Discovery platform service [10] in order to match LPML activity goals against global Web-scale third party services. SOA4ALL SD platform service accepts a subset of the WSMO Lite service model [8], that is, SD matches services according to the 5-tuple $< FC, I, O, P, E >$ where **FC**, **P**, **E** stands for WSMOLite functionalClassification, preconditions and effect respectively, and **I**, **O** describe concrete input/output parameters expressed within the P/E logical expressions.

However, this tuple describes the service as a whole, but not its concrete operations as needed by LPML modelling. Therefore, DTC uses the SOA4ALL SD platform service in order to seek for services only upon the functional classification based matching and then it filters the returned set of candidate services based on the matching of the I/O of their operations, exploiting sawsdl:modelReference annotations, according to the following algorithm:

input: a design model D

output: a modified D

begin

foreach $A_{Gi} \equiv G(A_i)$, A_i € D **do**

S=***sd***.*discover* (FC(A_{Gi}))

foreach S_j € S **do**

foreach o_k € O (S_j) **do**

if $I\ (Op_k) \sqsubseteq I\ (A_i) \wedge\ O\ (Op_k) \sqsupseteq O\ (A_i)$ **then**

S_j € S'

if ¬(|S'|=0) **then**

S(A) = S(A) U S'\S(A)

end

SD based DMA consumes a WSMOLite based SD service [10], exposed as a WSDL WS, using JAX-WS [28]. Additionally, SD based DMA consumes directly the same WSMOLite reasoning facility service [7] used by SD for additional querying.

2.3.3.4 Semantic Link Operator based DMA

The Semantic Link Operator (SLO) is a component that adds dataflow information to the LPML process models. The SLO is tightly integrated with the DTC and works at the last stage of creating complete process out of user-defined goals. Once the user goals are resolved by DTC and a proper solution has been found, the process is only missing data flow mappings in order to be complete. This is where the SLO starts its task. SLO takes the elaborated model and analyses it for semantic compatibility of inputs and outputs. If the model's I/Os are semantically correct, appropriate data mappings are created and solution is notified.

SLO works within DTC as a specialised DMA-class agent called SemanticLinkOperatorAgent. SLO Agent is working like all other registered agents by enhancing process design models in order to produce final solution. This agent also has its own knowledge base, the ontology of I/O compatibility that is used to reason about connections between services' inputs and outputs. Although SLO Agent is notified about new design models posted on the Blackboard, it acts only upon COMPLETE models. It means that SLO Agent is only analysing models, where all activities have attached services and therefore all I/Os are described with semantic annotations.

On the low level, current SLO Agent prototype uses an external SLO service that receives WSMO-lite annotations of inputs and outputs and calculates their semantic compatibility (satisfiability).[6]

[6] Next DTC prototypes will embed SLO service.

SLO Agent is providing appropriate inputs for SLO Service by analysing LPML model and its activities, according to its own algorithm presented below.

Each SLO Service query returns some information on how services' inputs can be satisfied by preceding outputs. After collecting all query answers regarding LPML model, SLO Agent decides whether this is enough to create I/O connectors [12] (precise mappings between input and output messages). If the whole model inputs and outputs are semantically compatible, a solution is posted to the blackboard.

The algorithm for computing semantic compatibility of LPML model is similar to the one proposed in [29]. LPML model is IO compatible when:

$$\bigcup_{a' \prec a} I(a) \subseteq^{SLO} O(a')$$

where $a' \prec a$ means that activity *a'* precede activity *a* in the control flow and *I(a)* is the set of input parameters of that activity, and *O(a)* is the set of outputs.

That literally means that the model is IO compatible, when every activity's input is satisfied by some output of previous activities in the control flow. Satisfiability is computed by calculating semantic matching between corresponding input and output, using SLO service.

When SLO Agent receives the LPML model, it works according to the following algorithm:

```
get list of activities A

foreach activity a in A:
    • get distinct list of activities A' preceding activity a: A'(a) = {a' : a' ≺ a}
    • let O(A') be set of all outputs that O(A') = {O(a') : a' ∈ A'}
    • let I(a) be a set of inputs of a
    • compute SLO(O(A'), I(a)) and get set of mappings M = {(i, o, s)} that describe semantic
      consistency s between input i and output o.

if every input I(A) is satisfied by some output then:
    • create data connectors
    • post solution.
```

$SLO(O(A'), I(a))$ describes invocation of core Semantic Link Design Operator service with 2 parameters: a set of outputs from preceding activities ($O(A')$), and a set of inputs of current activity ($I(a)$). All the inputs and outputs parameters are described using semantic annotations. SLO is also using I/O ontologies in order to reason about given parameters, and as a result, it returns pairs of inputs and outputs of semantically consistent parameters [29].

This information is used in order to update LPML model with I/O connectors between corresponding inputs and outputs. Based on semantic compatibility, an appropriate connector type is selected. Currently only *EXACT* and *PLUGIN* connector types are supported (EXACT, when input is exactly satisfied by corresponding output, PLUGIN – when output contains more information than needed by input). Connectors also contain information about syntactic and semantic mapping, that allow for lifting and lowering of message's parameters.

SLO DMA is implemented as a regular DMA-class agent, inside DTC. Semantic Link

Operator (SLO) Service that is used internally is invoked as an AXIS webservice through SOAP calls. Additionally SLO Service is using WSML for IO descriptions, and FaCT++ as a reasoning facility. Although upcoming months will be spent to fully implement support for WSMO-lite descriptions and use common SOA4All reasoner facilities.

SLO and Design Models Taxonomy

With the introduction of dataflow [12], a new dimension of Design Model taxonomy has been added. It is called *IO Compatibility* and has the corresponding tag that can have following values: IO_COMPATIBLE, IO_NOT_COMPATIBLE and IO_UNCHECKED_SOLUTION (for models that are not SLO checked). The new model taxonomy is illustrated on next figure.

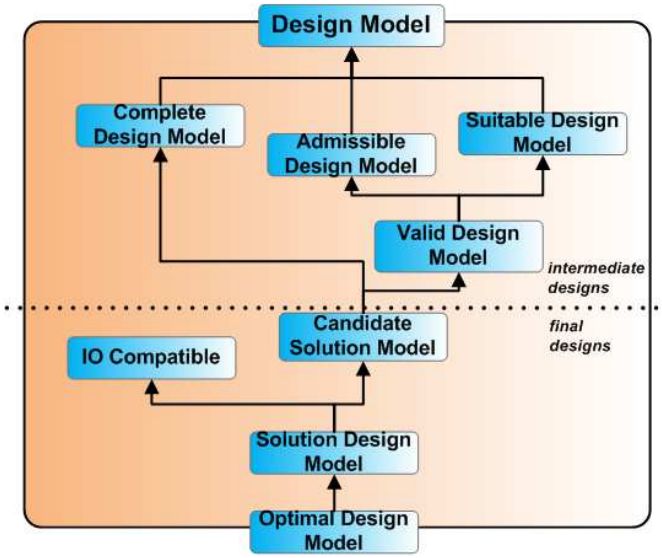

Figure 11 DTC design models taxonomy

Now the model D is a Solution Design Model if it is both Candidate Solution Model and is IO Compatible:

$$Solution(D) \equiv IOCompatible(D) \wedge CandidateSolution(D)$$

From technical point of view models, when models become solutions, when they are tagged by Design Analysis Agent with the following tag quadruple: (SUITABLE, ADMISSIBLE, COMPLETE, IO_COMPATIBLE).

2.3.3.5 Experiments and Testing.

DTC has been tested in the modelling of WP7 and WP9 LPML scenarios. Those experiments are detailed in Annex B.

2.4 Optimizer

In this section, we describe the advanced prototype of Optimizer platform service component, which theoretical grounding was described in [13].

2.4.1 Requirements and Functional Specifications.

This subsection describes new requirements requested for advanced Optimizer prototype.

Unlike the first version of the optimizer component (see [13]) was a standalone component, the new version is more integrated in the SOA4All platform. The performance of both version are the same, mainly the integration has been motivated by this new version, which were far from easy to achieve. For instance, the first version did not interact with the WP5 Service discovery component whereas the new version does. In addition, the optimizer is now totally LPML compliant whereas the first version was not. We also interface the optimizer as a web service and provide different client to ensure the SOA4All studio can interact with it. Finally, we upgraded our prototype by interfacing the optimizer to any Constraint Satisfaction Problem (CSP), so any CSP solver can be used to optimize the composition process.

Based on these new requirements and others described in [13], the functional description of Optimizer prototype is explained in detail. Furthermore, the public Optimizer interface exposed to be consumed by the SOA4ALL Studio is described.

2.4.1.1 Motivations and Requirements

Ranking and optimization of web service compositions represent challenging areas of research with significant implications for the realization of the “Web of Services” vision. Mostly due to performance optimization, context adaptation or specific user preferences and constraints, it is necessary to optimize the completed compositions. “Semantic web services” use formal semantic descriptions of web service functionality and interface to enable automated reasoning over web service compositions. To judge the quality of the overall composition, for example, we can start by calculating the semantic similarities between outputs and inputs of connected constituent services, and aggregate these values into a measure of semantic quality for the composition. First of all, the composition optimizer takes a specific interest modeling the way to compute “Quality of Compositions”. To this end, we suggested to combine semantic and non functional based criteria such as quality of service (QoS) for quality evaluation in web services composition. Therefore we proposes a novel and extensible model balancing the new dimension of semantic quality (as a functional quality metric) with a QoS metric, and using them together as ranking and optimization criteria. The semantic perspective comprises a set of metrics related to how well the functionalities of the constituent services fit together. The semantic quality is such a core metric, measuring the degree of semantic similarity between the outputs produced by constituent services and the inputs required by their peers (the SLO is used to this goal). Such a quality is one of the measures of the overall functional quality for the composition, indicating the “goodness of fit” between the functionalities of the constituent services. Web service compositions could thus be optimized and ranked using not only non-functional parameters such as the well-known Quality of Service, but also using semantic quality as a core indicator of functional quality. Our approach also demonstrates the utility of Genetic Algorithms to allow optimization within the context of a large number of services foreseen by the “Web of Services” vision.

Review of existing approaches to optimize web service compositions reveals that no approach has specifically addressed optimization of service composition using both QoS and semantic similarities dimensions in a context of significant scale.

Indeed main approaches focus on either non functional criteria such as QoS or on functional criteria such as semantic similarities between output and input parameters of web services for optimizing web service composition. In addition most of the proposals address composition optimization center on stochastic approaches [33], Constraint Programming [36] and Integer linear Programming (IP) [42], [39], with the latter considered showing most

promise. However, IP approaches have been shown to have poor scalability in terms of time taken to compute optimal compositions when the number of available services grows. The optimization problem can be also modeled as a knapsack problem [41], wherein [30] performed dynamic programming to solve it. Unfortunately the previous QoS-aware service composition approaches consider only links valued by Exact matching types, hence no semantic quality of compositions.

In contrast, we present an innovative model that addresses both types of quality criteria as a trade-off between data flow and non functional quality for optimizing web service composition.

Regarding this issue, we follow [32][31] suggest the use of GAs (Genetic Algorithms) to achieve scalable optimization in web service composition, yet we also extend their model by using semantic links to consider data flow in composition; considering not only QoS but also semantic quality (and constraints) of composition; revisiting the fitness function in order to avoid local optimal solution (i.e. compositions disobeying constraints are considered).

The context of significant and large scale in SOA4All is very important, so considered by using and simulating Gas to optimize web service compositions.

2.4.1.2 Functional Specifications

Due to previous motivations, the Composition Optimizer aims at optimize any compositions of services, whether described in a syntactic and/or semantic way.

The composition optimizer consumes the following resources as input parameters:

1. A composition (or process model) described in LPML (i.e., language defined in Task T6.3). The optimizer receives a complete process model and tries to replace current bound services with other web services which make better global cost function (i.e., in term of semantic and non functional quality).
2. A repository of web services with their syntactic (WSDL-based) and semantic (WSMO-Lite) descriptions attached;
3. A domain ontology and the ontologies attached to services;
4. A preference and constraint file in order to parameterize the optimization process. Such information is related to the criteria the end-user want to optimize first in case of multiple composition with same overall qualities.

The composition optimizer provides the following resources as output parameters:

1. A LPML description of the optimal composition. The optimizer transparently transforms compositions into their optimal versions by replacing service bindings and modifying the dataflow but without changing the workflow. Indeed the optimal composition is computed by assigning optimal services binding and optimal semantic connections assignments as well.

As described in this Section, the composition optimizer aims at being fully integrated with the other main SOA4All components.

2.4.1.3 Expected Benefits in Use Cases

As shown, the composition optimizer can be used to optimize any composition described by means of the LPML language.

Therefore the following situations will occur in the SOA4All use-cases:

- **WP7 (End-User Integrated Enterprise Service Delivery Platform):** Since all sorts of economic services and includes consulting, construction, maintenance, advertising, tourism, etc. are expected in this scenario, many services achieving the same overall functionality could be selected. The composition optimizer aims at supporting different users with different roles and skills (from the process analyst to non IT persons) in optimizing their compositions (once the latter is designed by means of the process editor).
- **WP8 (W21C BT Infrastructure):** In such a scenario, the optimization will act mainly at semantic level to ensure seamless composition of services. Indeed, given the high heterogeneity of services description in the scenario, the optimization process will act on this dimension, rather than on the non-functional criteria.
- **WP9 (C2C Service eCommerce):** through the use of *the Composition Optimizer*, C2C Service eCommerce use case will be entirely focused on providing an easy way for end users to use third party services offered through the framework and also optimize their compositions. This will enable them to build optimal eCommerce applications (in term of non-functional qualities such as prices, response time) in order to market and sell their own products. Contrary to the WP8 scenario the optimization will act mainly at non-functional level not only to ensure maximization of availability and reliability of composition but also to ensure the minimization of overall price and response time.

2.4.2 Theoretical grounding

This section summarizes new theoretical grounding implemented by the advanced Optimizer prototype, but not included in [13].

In the deliverable [13] a GAs based approach has been introduced to optimize the composition of semantic web services. However, we did not define the optimization in formal way. In this section, we formally define the optimization problem in order to easily adapt any other optimization approaches e.g., stochastic approaches, Constraint Programming and Integer linear Programming. In addition we justify the choice of Gas. Finally, we describe how the Gas have been parameterize to compute the optimal compositions.

The computation of the optimal composition is valued among a set of potential solutions. In the following, we formalize the problem as a Constraint Satisfaction Optimisation Problem (CSOP). Then the Genetic Algorithms, presented in deliverable [13], is used compute an optimal solution that meets constraints on i) the quality of their services and ii) the quality of their semantic links. To this end the quality model (equation (6) in [13]) is used to model local constraints on both semantic links (see its second component) and services (see its first and third components) whereas equation (7) in [13] is considered to model global constraints.

Example 14. (Compositions and Constraints)

Suppose a composition (Figure 12) with eight tasks $T_{i,1\leq i\leq 8}$ *(*here, the concept Task T_i refers to the concept of goal G_i in [13] without any difference*) to be bound to services, and eight semantic links* $sl_{i,j}$ *to be instantiated by concrete semantic links. Its process model consists of sequences, OR-, AND-Branching. In addition, we assume that such a composition of tasks is achieving a specific goal. The end-user is requested to provide some constraints on*

the composition she expects. For example, the end-user may have a limited budget and thus the execution price (see the definitions in the Quality Model Section 3.2.2 of [13]*) is constrained, or she cannot accept a matching quality below a given limit. We can also imagine local constraints on specific tasks and semantic links. From these constraints, the end-user could expect the optimal one regarding its quality.*

2.4.2.1 Constraint Satisfaction oriented Optimisation

Here we formalize web service composition as a Constraint Satisfaction Optimisation Problems (CSOP). We use the term CSOP to refer to the standard Constraint Satisfaction Problem (CSP) as defined in [40], plus the requirement of finding optimal solutions.

CSOP is a key formalism for many optimisation driven combinatorial problems such as ours. The success of this paradigm is due to its simplicity, its natural expressiveness of several real-world applications and especially the efficiency of existing underlying solvers. In addition, this formalism allows a generic representation of any optimisation-based web service composition problem with local and global constraints (Definition 1).

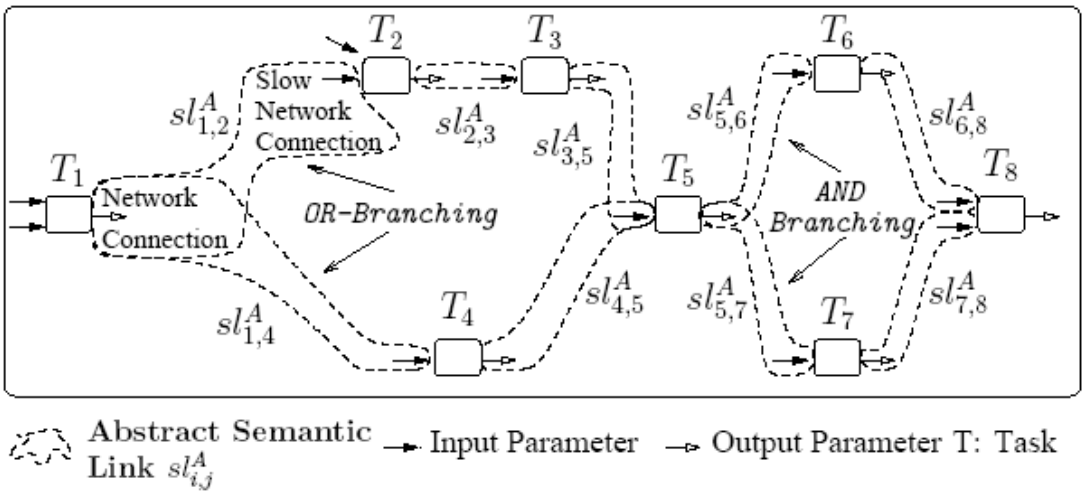

Figure 12 A Composition with no Service Binding[7]

Definition 1 (Composition Driven CSOP)

A Composition Driven CSOP is defined as a 4-tuple (T, D, C, f) where:

- T is the set of tasks (variables) $\{T_1, T_2, ..., T_n\}$ defined in the composition;
- D is the set of domains $\{D_1, D_2, ..., D_n\}$, each D_i representing a set of possible concrete services that fulfil the task T_i;
- C is the set of constraints i.e., local C_L and global C_G.

[7] See [13] for further information regarding the definition of semantic links $sl_{i,j}$.

- f is an evaluation function that maps every solution tuple $s \in S$ of the CSP (T, D, C) to a numerical value. Given a solution tuple s, $f(s)$ is called the f-value of s.

The local and global constraints are related to users constraints regarding both the semantic and non-functional quality of composition, services and their semantic links. Unlike constraints C_L, which need to be satisfied for any given assignment (i.e., services and semantic links) to specific tasks T, constraints C_G need to be met by the overall concrete composition.

Solving a composition driven CSOP consists in finding the solution tuple (i.e., an assignment of services $s_{i,1\leq i\leq n} \in D_1 \times D_2 \times ... \times D_n$ to tasks $T_{i,1\leq i\leq n}$ that satisfy all the constraints C) with the optimal (here maximal) f-value with regard to the application-dependent optimisation function f.

The optimisation problem i.e., determining the best set of services of a composition with respect to some quality constraints, is NP-hard. In case the number of tasks and their number of candidate services are respectively n and m, the naive approach (i.e., consisting in finding all the solutions first, and then compare their f-values) considers an exhaustive search of the optimal composition among all the m^n concrete compositions (at least conceptually speaking).

Since such an approach is impractical for large-scale composition, we address this issue by presenting a GA-based approach [35] which i) supports constraints not only on QoS but also on quality of semantic links and ii) requires the set of selected services as a solution to maximize a given objective f.

One of the optimization approaches, i.e., the GA-based approach is described in details in [13]. In the following section, we describe how the GAs have been parameterized.

2.4.2.2 Parameterization of the GAs process to optimize compositions

The optimal compositions are computed using an elitist GA where the best two compositions are kept alive across generations. A crossover probability of 0:7, a mutation probability of 0:1 and a population of 200 compositions have been considered. The roulette wheel selection has been adopted as selection mechanism. Finally we consider a simple stopping criterion i.e., up to 400 generations. Such values for parameters have been selected mainly because there is a little deterioration in term of performance beyond them.

2.4.3 Implementation architecture.

This section depicts the implementation details, the external interactions and API of the advanced Optimizer prototype.

2.4.3.1 Implementation Details

The internal architecture of the composition optimizer mainly consists in an adaptation of a GA algorithm (as described in [13]). The main issues were regarding the external integration (see Section 2.4.3.2). Of course, different steps of serializations are required. In our approach, we have dealt with WSML for services, WSMO for ontologies and LPML for composite services.

From a more technical point of view, the common description rate described in equation (3) of [13] is calculated by computing the equation (2) of Extra Description in [13], the least common subsumer [34] and the size ([38] p.17) of DL-based concepts. These DL inferences as well as the matching types have been achieved by means of a DL reasoning process (actually an adaptation of Fact++ [8] [37] for considering DL difference). The aggregation rules introduced in [13] are then used for computing each quality dimension of any composition. Finally, the combination of QoS with semantic calculation is computed by means of equation (9) in [13], thus obtaining the final quality score for the composition. The GA process is implemented in Java, extending the GPL library JGAP (http://jgap.sourceforge.net/).

2.4.3.2 Interactions and API

The composition optimizer has been exposed in SOA4All as an external web service, so it can be accessed using any JAX-WS framework. Optionally the DTC can be deployed as WS locally (see Annex A related to Installation and Configuration). The optimizer source code includes a software client API, OptimizerClient, to access this service from any client, for instance, from the Process Editor.

The composition optimizer interacts with the following SOA4All Tasks (and underline components):

- Task T1.4 DSB Deployment and Architecture to be integrated as a complete component of the SOA4All architecture;
- Task T2.3 monitoring system that provides execution logs as QoS data, required to optimize on this level.
- Task T2.6 Process Editor in order to give the end-user the possibility to optimize the composition or keep as it is. Such an integration is important to ensure the user has access to the optimizer. In addition, the task T2.6 will display the differences between an optimal and a non-optimal process, mainly at service binding level. Currently the optimizer is not integrated with the Process Editor. Both input models and the optimizer result models are visualized using an ad-hoc LPML visualizer (i.e., the same used by DTComposer).
- Task T3.2 WSMO-Lite Reasoner engine in order to valuate semantic connections between services in a composition by using subsumption relationships and partial ordering on them. So far the integration is not complete.
- Task T5.3/T5.4 Goal-driven Service Discovery that is required by the optimizer in order to discover services that could be replaced at design time, depending on semantic and non functionality of the overall composition;
- Task T6.4 Design Time Composer, which provides the composition to be optimized. In other words the output of the Design Time Composer will serve as input of the composition optimizer;
- Task T6.4 Template Generator, which also provides a composition as input parameter to be optimized in case the Design Time composer fails to retrieve the latter.

[8] http://owl.man.ac.uk/factplusplus/

- Task T6.5 Execution Engine, which could take the result of the composition optimizer as an input parameter. First of all, the composition provided by the optimizer is ready to be deployed (i.e., transformed into an executable format) and then executed by the Execution engine.

The main interface of the optimizer service is located: *SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/java/org/SOA4All/core/ICoreOptimizer4Serv.java*

The latter interface consists of one main method:

- The method to optimize the composition with available knowledge about semantic web services descriptions, a domain ontology, and some GA parameters (initially with default values). This method receive one input LPML process model serialized as XML and return only one output processed LPML process model, serialized as XML in the same XML format. This method can operated upon the whole process (optimize).

Its implementation is located: *SOA4All-service-construction-optimizer/src/main/java/org/SOA4All/core/CoreOptimizer4Serv.java*

The API exposed is:

```
@WebMethod
public  @WebResult(name = "String", targetNamespace = "http://core.SOA4All.org/") String
                optimize(
                        @WebParam (name="processURL", targetNamespace = "http://core.SOA4All.org/") String processURL)
                        throws Exception;
Returns the URL of the optimal composition (in LPML)
@param processURL    URL of the non optimal composition to be optimized
```

The endpoint of this web service is located on an one of the partner servicer (ATOS) : http://nexof-ra.atosorigin.es:8080/axis2/services/Optimizer?wsdl

In order to ease the access to the optimizer service, we have provided a client in the following location: *https://svn.sti2.at/SOA4All/trunk/SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/java/org/SOA4All/client/*

Some tests are provided in *https://svn.sti2.at/SOA4All/trunk/SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/java/org/SOA4All/core/Core.java*

2.4.3.3 Experiments and Testing.

Optimizer has been tested in the modelling of WP9 scenario. These experiments are detailed in Annex B .

2.5 DTCE prototypes implementation roadmap.

This section summarises the implementation and integration roadmap for DTCE prototypes. All DTCE tools, TG, DTC and Optimizer are integrated each other through the usage of the common LPML API. Integration of DTCE tools and EE (T6.5) will be performed by M30 through the T2.6 PE.

2.5.1 Template Generator Implementation roadmap

The following table shows the implementation roadmap for the Template Generator.

Functionality:	First Prototype (M12)	Advanced Prototype (M24)	Final Prototype (M30)
SOA4All logs input	NO (ad-hoc log)	NO (ad-hoc log)	YES
Context-driven logs filtering	NO	YES (simple)[9]	YES (full)
Schemas generation at different completeness level	YES	YES	YES
Schemas generation at different abstraction level	NO	YES	YES
Input parameters wizard	NO	YES	YES
GUI integrated in the Process Editor	YES	YES (improved)	YES
Export to LPML	NO	YES	YES
Store schemas to SOA4All infrastructure	NO	NO	YES

Table 1Template Generator implementation roadmap

[9] A simple implementation is planned as proof of concept: based on future availability of real SOA4All logs and on implementation of the context framework, feedback will be gathered and a final solution will be provided for the “Final Prototype” at month 30.

2.5.2 Design Time Composer Implementation roadmap

The following table shows the implementation roadmap for the DTC.

Functionality:	First Prototype (M12)	Advanced Prototype (M24)	Final Prototype (M30)
DTC Architecture	YES (partial)	YES	YES
Rule DSL DMA	YES (partial)	YES	YES
WSML DMA	NO	YES (partial)	YES
SLO DMA	NO	YES (partial)	YES
SD DMA	NO	YES (partial)	YES
Context	NO	YES (partial)	YES
Template Storage	NO	NO	YES
LPML	YES	YES	YES
Performance improvement	NO	YES (partial)	YES
Platform service	NO	YES	YES
Integration with T2.6 Process Editor	NO	NO[10]	YES
Integration with T1.4 DSB Deployment and Architecture	NO	NO	YES
Integration with T2.3 Reasoner	NO	YES	YES
Integration with T5.3/T5.4 Discovery	NO	YES	YES

[10] Integration initiated since DTC was exposed as platform service (M22)

Table 2 DTC implementation roadmap

2.5.3 Optimizer Implementation roadmap

The following table shows the implementation roadmap for the Optimizer.

Functionality:	First Prototype (M12)	Advanced Prototype (M24)	Final Prototype (M30)
GAs based Resolution	YES	YES	YES
LPML Support	YES	YES	YES
CSP based Resolution	NO	YES	YES
QoS and Semantic quality model	NO	YES	YES
WSDL, WSMO, BPEL Support	YES	YES	YES
WSMO-Lite Support	NO	NO	YES
Integration with T1.4 DSB Deployment and Architecture	NO	NO	YES
Integration with T2.3 Monitoring System	NO	NO	YES
Integration with T2.6 Process Editor	NO	YES	YES
Integration with T3.2 reasoning engine	NO	NO	YES
Integration with T5.3/T5.4 Discovery Engine	NO	YES	YES

Table 3 Optimizer implementation roadmap

3. Conclusions

This document has provided a comprehensive, integrated and detailed functional, technical and implementation description of the current prototype for the Design Time Composition Environment (DTCE). This view has been framed within the overall SOA4ALL architecture, aiming at depicting how the DTCE tools work in a collaborative way with SOA4ALL Studio Process Editor, covering most of the BPM lifecycle tasks, guided by the LPML methodology.

This document provides also detailed functional, technical, implementation, installation and configuration information about every DTCE tool prototype: Template Generator, Design Time Composer and Optimizer.

The DTCE tools are seamlessly integrated through a common LPML language and methodology, and through a common access point, the Studio PE. The DTCE tools simplifies the BP engineering life cycle at design time. TG lets modellers start from a BP template, but not from scratch, reusing implicit or hidden knowledge of past BP executions. DTC completes BP models with further details and helps in solving complex modelling tasks. Optimizer optimizes the BP model in terms of metrics such as the NFP (e.g. QoS) and the semantic quality.

The DTCE tools have been tested in the context of WP7 and WP9 scenarios. Results of those experiments are available in Annex B.

Current DTCE tools can be considered matured regarding the level of fulfilment of the envisioned features and the requested requirements (see [13]). However, they are still somehow a bundle of individual components that require further integration between them, the Studio PE, other SOA4ALL platform services (Service Discovery, Ranking and Reasoning) and the SOA4ALL infrastructure, in order fully to exploit the SOA4ALL benefits for user-friendly modelling of BP. In that sense, next development period (M30) will be mainly applied to complete that integration, so end-users may perceive SOA4ALL service construction tools (PE, DTCE and EE) as a single entity accessible from the PE.

Furthermore, DTCE features will be intensive tested by the real case studies scenarios, exploiting the available domain specific knowledge (SWS/Templates, domain models, context, etc) provided by the case studies developers. As a result of this evaluation, DTCE tools will be improved during this next period.

4. References

[1] SOA4ALL D1.3.2B. Distributed Semantic Spaces: A First Implementation, 2009.

[2] SOA4ALL D1.4.1A. SOA4ALL Reference Architecture Specification, 2009.

[3] SOA4ALL D1.4.1B. SOA4ALL Runtime V1, 2009.

[4] SOA4ALL D2.6.2 SOA4ALL Process Editor. First Prototype. 2009.

[5] SOA4ALL D2.2.2 Service Consumption Platform First Prototype. 2009.

[6] SOA4ALL D2.3.2 Service Monitoring and Management Tool Suite First Prototype. 2009.

[7] SOA4ALL D3.2.2 First Prototype Reasoner for WSML Core v2.0. 2009.

[8] SOA4ALL D3.4.2 Defining WSMO Lite as an extension of SAWSDL. 2009

[9] SOA4ALL D3.4.7 Defining extensions to WSMO for capturing contextual information. 2009.

[10] SOA4ALL D5.3.1 First Service Discovery Prototype. 2009.

[11] SOA4ALL D6.3.1. Specification of Lightweight Context-aware Process Modelling Languare, 2008.

[12] SOA4ALL D6.3.2. Advanced Specification Of Lightweight, Contextaware Process Modelling Language, 2009.

[13] SOA4ALL D6.4.1. Specification and First Prototype Of Service Composition and Adaptation Environment, 2009.

[14] SOA4ALL D6.5.2. Advanced Prototype For Adaptive Service Composition Execution, 2010.

[15] G. Greco, A. Guzzo, and L. Pontieri. Mining hierarchies of models: From abstract views to concrete specifications. In Proc. 3rd Intl. Conf. on Business Process Management (Bprocess mining'05), pages 32--47, 2005

[16] J. A. Hartigan and M. A. Wong. A K-Means Clustering Algorithm. Applied Statistics, 28(1): 100–108, 1979.

[17] Wielinga B. J., Akkermans J. M., Schreiber A. Th., A Formal Analysis of Parametric Design Problem Solving, In Proceedings of the 9th Banff Knowledge Acquisition Workshop (KAW-95)

[18] Pedrinacci C. (2005) Knowledge-Based Reasoning Over The Web, PhD. Dissertation, Universidad País Vasco San Sebastián, Noviembre 2005

[19] Erman L. D., Hayes-Roth F., Lesser V. R., and Reddy D. R.. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. Computing Surveys, 12(2):213–253, June 1980

[20] Motta E., (1999) Reusable Components For Knowledge Modelling Case Studies In Parametric Design Problem Solving,IOS Press (Netherlands)

[21] M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” ACM Computing Survies, vol. 37, no. 4, pp. 316–344, 2005

[22] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. Applied Ontology , 1(1):77–106, 2005

[23] Uwe Keller, Ruben Lara, Holger Lausen, Dieter Fensel: Semantic Web Service Discovery in the WSMO Framework. In Semantic Web Services: Theory, Tools and Applications, 2006

[24] R. Johnson, J. Hoeller, A. Arendsen, T. Risberg, C. Sampaleanu. Professional Java Development with the Spring Framework. Wrox. 2007

[25] M. Bali. Drools JBoss Rules 5.0 Developer's Guide. Packt Publishing 2009.

[26] Farrell J., Lausen H. Semantic Annotations for WSDL and XML Schema. W3C TR (http://www.w3.org/TR/sawsdl/). 2007

[27] S. Grimm, U. Keller, H. Lausen, and G. Nagypal. A Reasoning Framework for Rule-Based WSML. In Proceedings of 4th European Semantic Web Conference (ESWC), Innsbruck, Austria, June 3 - 7 2007

[28] R. Chinnici, M. Hadley, R. Mordani. Java API for XML-Based Web Services 2.0. http://jcp.org/aboutJava/communityprocess/final/jsr224/index.html. 2006.

[29] Freddy Lécué, Alexandre Delteil, Alain Léger: Extending Web Service Composition Languages with Semantic Data Flow. ICSC 2009: 174-183

[30] Ismailcem Budak Arpinar, Ruoyan Zhang, Boanerges Aleman-Meza, and Angela Maduko. Ontology-driven web services composition platform. Inf. Syst. E-Business Management, 3(2):175–199, 2005.

[31] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In GECCO, pages 1069–1075, 2005.

[32] Gerardo Canfora and Massimiliano Di Penta. A lightweight approach for QoS-aware service composition. In Proc. 2nd International Conference on Service Oriented Computing, 2004.

[33] Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. J. Web Sem., 1(3):281–308, 2004.

[34] William W. Cohen, Alexander Borgida, Haym Hirsh Computing Least Common Subsumers in Description Logics In AAAI, pages 754–760, 1992.

[35] David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[36] Ahlem Ben Hassine, Shigeo Matsubara, and Toru Ishida. A constraint-based approach to web service composition. In ISWC, pages 130–143, 2006.

[37] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In KR, pages 636–649, 1998.

[38] Ralf Kusters. Non-Standard Inferences in Description Logics, volume 2100 of Lecture Notes in Computer Science. Springer, 2001.

[39] Freddy Lecue, Alexandre Delteil, and Alain Leger. Optimizing causal link based web service composition. In ECAI, pages 45–49, 2008.

[40] E. Tsang. Foundations of Constraint Satisfaction. 1993.

[41] Tao Yu and Kwei-Jay Lin. Service selection algorithms for composing complex services with multiple qos constraints. In ICSOC, pages 130–143, 2005.

[42] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. IEEE Trans. Software Eng., 30(5):311–327, 2004.

Annex A. Installation and Configuration

This section describes the installation and configuration procedure for each DTCE platform service component. This procedure depends on how the platform service component is deployed and exposed to potential consumers.

Template Generator

Software: https://svn.sti2.at:/SOA4All/trunk/SOA4All-service-construction/SOA4All-service-construction-module-templategenerator

Dependencies:

- JDK 1.6.x
- ProM plug-in libraries
- ProM models libraries
- MXML libraries
- Cern Colt 1.0 library
- XML Pull parser 1.0 library
- LPSolve library

Installation:

The Template Generator code is part of the Process Editor, thus it is automatically installed with it. The TG has no particular configuration to be set.

Design Time Composer

Software: https://svn.sti2.at:/SOA4All/trunk/SOA4All-service-construction/SOA4All-service-construction-dtcomposer

Dependencies:

- JDK 1.6.X
- Tomcat 6.0.20
- Axis2 1.5.1 war file
- FaCT++ 1.2.2 (windows precompiled 32bit version at: http://factplusplus.googlecode.com/files/FaCTpp-win-v1.2.2.zip or sources http://factplusplus.googlecode.com/files/FaCTpp-src-v1.2.2.tgz)
- SOA4ALL Reasoner platform service

- SOA4ALL Service Discovery platform service

Installation and configuration:

- Install dependencies:

  Install JDK 1.6.X

  Unzip Tomcat distribution. Directory created referred as $TOMCAT_HOME

  Copy axis2.war into $TOMCAT_HOME/webapps

  Start Tomcat: $TOMCAT_HOME/bin/startup.sh

- Checkout project:

  SOA4ALL SVN Repository: svn co https://svn.sti2.at:/SOA4All/trunk/SOA4All-service-construction/SOA4All-service-construction-dtcomposer

  We refer to this checkout directory as $DTC_HOME

- Configure DTC installation dir:

  Edit $DTC_HOME/DesignTimeComposer_v2/src/main/resources/dtcomposer-service-configuration.xml

  Set <property name="INSTALLATION_DIR" value="$DTC_HOME" />

- Configure DTC platform service dedendencies for SD-DMA.

  Edit $DTC_HOME/DesignTimeComposer_v2/src/main/resources/dtcomposer-service-configuration.xml. Set Semantic Discovery and Reasoner endpoints.

```
<bean name="designModificationSDAgent"
        class="eu.soa4all.wp6.composer.agents.DesignModificationSDAgent">
        <constructor-arg
        value="http://localhost:8080/axis2/services/SemanticDiscovery?wsdl" />
        <!-- Service Discovery Endpoint -->
        <constructor-arg value="http://localhost:8765/soa4all/reasoner?wsdl" />
        <!-- Reasoner Endpoint -->
                …
</bean>
```

- Create DTComposer service aar file:

  cd DTC_HOME

  mvn -Dmaven.test.skip="true" clean install

- Deploy DTComposer service aar file :

  Copy

  $DTC_HOME/DesignTimeComposer_v2/target/DesignTimeComposer-1.0-SNAPSHOT.aar into $TOMCAT_HOME/webapps/axis2/WEB-INF/services

- Test deployment :

  Access http://localhosts:8080/axis2/services/listServices to check it is listed in Axis2 services.

- Test DTComposer service:

  Use tests under:

  $DTC_HOME/DTComposerClient/src/test/java/eu.SOA4All.wp6.composer.client.test.* packages

- Configure DTComposer service logs in Tomcat/Axis2:

  Edit $TOMCAT_HOME/webapps/axis2/WEB-INF/classes/log4j.properties

  Comment Axis2 log4j configuration and add DTComposer log4j configuration.

  Activate CONSOLE and/or LOGFILE appenders

  Set log level: Default ERROR for dependencies and DEBUG for DTC. See listing example below.

Semantic Link Operator service (used by DTC)

- Checkout project:

  SLO project is located at: https://svn.sti2.at/SOA4All/trunk/SOA4All-service-construction/SOA4All-service-construction-semantic-link)

- Checkout to your preferred location ($SLO_HOME)

- Prepare FaCT++ reasoner:

  Download appropriate FaCT++ version unpack it to your preferred location (referred to as $FACT_HOME)

- Create aar service package:

  cd $SLO_HOME

  mvn clean install

- Deploy SLO service aar file:

  Copy $SLO_HOME/target/SOA4All-service-construction-semantic-link-0.0.1-SNAPSHOT.aar

  into $TOMCAT_HOME/webapps/axis2/WEB-INF/services

- Start external reasoner:

  Run FaCT++ reasoner $FACT_HOME/32bit/FaCT++.Server.exe (in case of 32bit windows machine) on the same machine that DTC is executed on (localhost)

```
# AXIS2 LOG4J section ***********************************************
# Set root category priority to INFO and its only appender to CONSOLE.
#log4j.rootCategory=INFO, CONSOLE
#log4j.rootCategory=INFO, CONSOLE, LOGFILE
# Set the enterprise logger priority to FATAL
#log4j.logger.org.apache.axis2.enterprise=FATAL
#log4j.logger.de.hunsicker.jalopy.io=FATAL
#log4j.logger.httpclient.wire.header=FATAL
#log4j.logger.org.apache.commons.httpclient=FATAL
# CONSOLE is set to be a ConsoleAppender using a PatternLayout.
#log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
#log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
#log4j.appender.CONSOLE.layout.ConversionPattern=[%p] %m%n
# LOGFILE is set to be a File appender using a PatternLayout.
#log4j.appender.LOGFILE=org.apache.log4j.FileAppender
#log4j.appender.LOGFILE.File=axis2.log
#log4j.appender.LOGFILE.Append=true
#log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
#log4j.appender.LOGFILE.layout.ConversionPattern=%d [%t] %-5p %c %x - %m%n
# END AXIS2 LOG4J section *************************************************
# DTCOMPOSER LOG4J section ************************************************
log4j.rootCategory=ERROR, CONSOLE
#log4j.rootCategory=ERROR, CONSOLE, LOGFILE
log4j.logger.eu.soa4all.wp6.composer = DEBUG
# A1 ConsoleAppender.
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=%d [%t] %-5p %c:%L - %m%n
# A2 FileAppender.
log4j.appender.LOGFILE=org.apache.log4j.RollingFileAppender
log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGFILE.layout.ConversionPattern=%d [%t] %-5p %c:%L - %m%n
log4j.appender.LOGFILE.File=dtcomposer.log
# DTCOMPOSER LOG4J section ************************************************
```

Optimizer

Software: https://svn.sti2.at:/SOA4All/trunk/SOA4All-service-construction/SOA4All-service-construction-optimizer

Dependencies:

- JDK 1.6.X
- Tomcat 6.0.20
- Axis2 1.5.1 war file

Installation and configuration:

- Install dependencies:

  Install JDK 1.6.X

  Unzip Tomcat distribution. Directory created referred as $TOMCAT_HOME

  Copy axis2.war into $TOMCAT_HOME/webapps

  Start Tomcat: $TOMCAT_HOME/bin/startup.sh

- Checkout project:

SOA4ALL SVN Repository: https://svn.sti2.at/SOA4All/trunk/SOA4All-service-construction/SOA4All-service-construction-optimizer/

We refer to this checkout directory as $OPTIMIZER_HOME

- Configure installation dir:

  Edit $OPTIMIZER_HOME/src/main/resources/configuration.props and modify values of AVAILABILITY_OBJECTIVE, PRICE_OBJECTIVE, RESPONSETIME_OBJECTIVE, MATCHING_QUALITY_OBJECTIVE, ROBUSTNES_OBJECTIVES, GA_GENRATION, GA_POPULATION, depending on your preferences. By default, these values are initialised.

- Create Optimizer service aar file:

  cd OPTIMIZER_HOME

  mvn -Dmaven.test.skip="true" clean install

- Deploy Optimizer service aar file :

  Copy

  *$OPTIMIZER_HOME/target/*SOA4All-service-construction-optimizer-0.0.1-SNAPSHOT.aar *into $TOMCAT_HOME/webapps/axis2/WEB-INF/services*

- Test deployment :

  Access http://localhosts:8080/axis2/services/listServices to check it is listed in Axis2 services.

- *Run the Semantic reasoner:*

cd $OPTIMIZER_HOME/src/main/resources/Tool/FaCT++-win-v1.2.2/32bit/
FaCT++.Server.exe

- Test Optimizer service:
  Use tests under:
  $DTC_HOME/DTComposerClient/src/test/java/org.SOA4All.client.test. packages*
- Configure the Optimizer service logs in Tomcat/Axis2.

Annex B. DTCE Experiments

This annex collects a set of examples of usage of DTCE platform service components taken from WP7 and WP9 case studies.

Template Generator

Currently the Template Generator is integrated in the Process Editor and can be accessed from the top right corner of the user interface, as shown in the following picture:

Figure 13 - Template Generator GUI invocation

Creation of the templates hierarchy

Through the TG interface, the user is able to select which kind of algorithm to use, set its main parameters and pick the kind of logs he is interested in for the analysis.

Once the user has set all these preliminary parameters, the Template Generator widget retrieves the requested logs and passes them to the server side service in the SOA4All format.

The server side service converts all the entries received into MXML format and parses them to proceed to the analysis. The input will appear as represented in the following picture:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<WorkflowLog>
  <Source program="Fake log generator" />
  <Process id="TemplateA" description="1st provider template">
    <ProcessInstance id="011">
      <AuditTrailEntry><WorkflowModelElement>getProductList</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getProduct</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>AggregateCatalogProduct</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getProduct</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>AggregateCatalogProduct</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getProduct</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>AggregateCatalogProduct</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>aggregateData</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
    </ProcessInstance>
    <ProcessInstance id="013">
      <AuditTrailEntry><WorkflowModelElement>getProductList</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getProduct</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>UpdateCatalog</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getProduct</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>UpdateCatalog</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>aggregateData</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
    </ProcessInstance>
    <ProcessInstance id="201">
      <AuditTrailEntry><WorkflowModelElement>getProductDescriptionList</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>mergeItems</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>aggregateData</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
    </ProcessInstance>
    <ProcessInstance id="201">
      <AuditTrailEntry><WorkflowModelElement>getProductPricesList</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>mergeItems</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>aggregateData</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
    </ProcessInstance>
    <ProcessInstance id="301">
      <AuditTrailEntry><WorkflowModelElement>getAvailableCatalogs</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getProductFromCatalog0</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getItemAvailability</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getItemPrice</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>addItemToList</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>aggregateData</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
    </ProcessInstance>
    <ProcessInstance id="302">
      <AuditTrailEntry><WorkflowModelElement>getAvailableCatalogs</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getProductFromCatalog1</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getItemAvailability</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>getItemPrice</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>addItemToList</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
      <AuditTrailEntry><WorkflowModelElement>aggregateData</WorkflowModelElement><EventType>start</EventType></AuditTrailEntry>
    </ProcessInstance>
```


Figure 14 - MXML input ready to be analysed

The algorithm selected by the user takes all the MXML inputs, infers on the parsed data and generates a hierarchy of potentially interesting schema templates, proposing them to the user under a tree structure.

When the user selects one of the tree items drawn on the left output panel, the corresponding schema is loaded and presented adjacent to it.

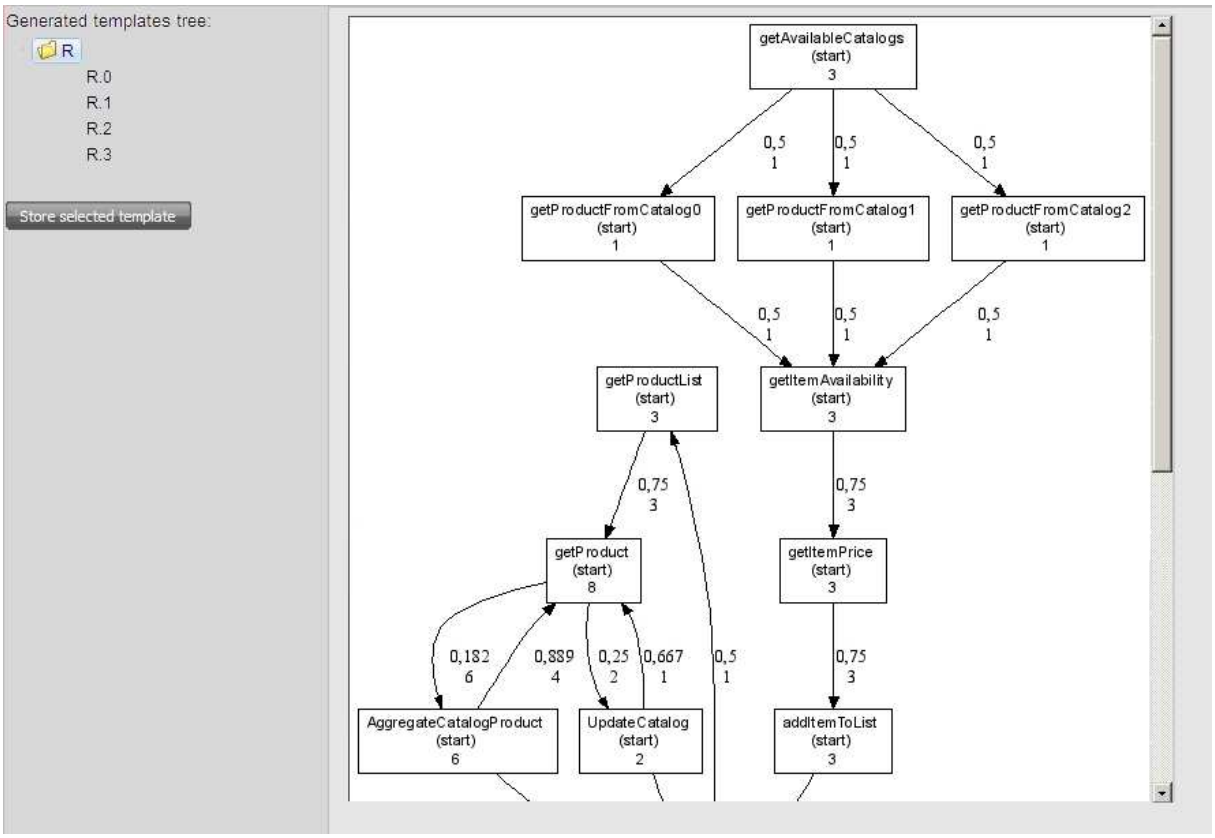

Figure 15 - Generated template schema tree and selected schema representation

Storing the wished schema

The user is now able to select the template that looks more coherent to his needs and can press the button under the tree to store the schema in LPML format.

The client module invokes the storage service to save the LPML conversion of the template the user has selected.

Now this template is available and ready to be edited by the Process Editor.

Design Time Composer

Currently DTC is not yet fully integrated with Process Editor. Therefore, these tests send manually created input models to the DTC service. Both input models and DTC result models are visualized by the test using an ad-hoc LPML visualizer (shipped within the DTC code).

Experiments using DTC in WP7 eGoverment scenario.

UnrestrictedBusinessRegistrationTest.

This test models the WP7 V1 business registration process supporting any payment method.

Input process model is a draft model including one general (at process level) annotation requiring any payment method: http://www.SOA4All.eu/eGoverment/ontology# UnrestrictedPayment.

All process model annotations (used al process level or activity) reference concepts from a common WSMO Lite ontology located under:

service-construction-dtcomposer/DesignTimeComposer_v2/src/main/resources/ontologies/WP7/wp7-ontology.wsml

This draft input model consists of a very simple business registration workflow, a sequence flow of annotated activities (we use annotations to describe each activity goal). A modeller using the Process Editor should create this model. In current test, we have created it programmatically using the LPML API (see D6.3.2). Next figure shows the input model as shown by the LPML viewer.

Figure 16 – WP7 input registration process for unrestricted payment.

This test invokes the method ***resolveProcessWithTemplate*** of DTC, requesting the replacement of the model activities of the input process with matching process templates. Similar results would be obtained invoking the method ***resolveProcess*** that tries to apply all knowledge registered within the DTC, since we have not included matching SWS for the activities within the initial process model in the current WP7 knowledge. However, DTC also support SWS matching (See WP9 example). The invocation of DTComposer service is like this:

String outputModelXml = composer.resolveProcessWithTemplate(inputModelXml);

Composer is the Axis2 stub. This method accepts a LPML process model serialized as XML and returns a processed model, serialized in the same format. Next figure shows the output model as shown by the LPML viewer.

Figure 17 – WP7 output registration process for unrestricted payment.

This test and the next one use only DSL Rule (Drools) based knowledge (describing templates) as described previously and WSML descriptions of SWS (WSMO Lite) discoverable through the SD-DMA. WSMO Lite (WSML) SWS descriptions and queries are used also in WP9 test. DSL Rules for WP7 test are located in directory:

service-construction-dtcomposer/DesignTimeComposer_v2/src/main/resources/rules/wp7

WSMO Lite SWS descriptions are located under

service-construction-dtcomposer/DesignTimeComposer_v2/src/main/resources/ontologies/wp7/sws-wp7.wsml

CreditCardBusinessRegistrationTest.

This test models the WP7 V2 business registration process supporting only a credit-card payment method.

Input process model is a draft model including one general (at process level) annotation requiring any payment method: http://www.SOA4All.eu/eGoverment/ontology CreditCardPayment. Apart from this different annotation, input model is the same than in the previous test. Next figure shows the input model as shown by the LPML viewer.

Figure 18 – WP7 input registration process for credit card payment.

This test behaves exactly as previous ones, and only differs on the input process model and therefore in the model solution returned. Next figure shows the output model as shown by the LPML viewer.

Figure 19 – WP7 output registration process for credit card payment.

Experiments using DTC in WP9 eCommerce scenario.

DTComposerWP9ScenarioDemo.

This test models a variant of the WP9 eCommerce Webshop Catalogue update process. The modelling process is a combination of manual modelling using the Process Editor (which is simulated in this test programmatically, that is, the manual models are created programmatically) and assisted modelling using the DTC service. This test explains a possible jointly process modelling phase in which the modeller (end-user) through the Process Editor is accessing the DTC service.

This test uses two sources of SWS knowledge:

- DSL rules (DROOLS) describing available domain specific templates and SWS, located under:

  service-construction-dtcomposer/DesignTimeComposer_v2/src/main/resources/rules/wp9

- WSML Lite SWS descriptions and queries, located under:

  service-construction-dtcomposer/DesignTimeComposer_v2/src/main/resources/ontologies/wp9

All process model annotations (used al process level or activity) reference concepts from a common WSMO Lite ontology located under:

service-construction-

dtcomposer/DesignTimeComposer_v2/src/main/resources/ontologies/WP9, subdirectories context, FC, domain-specific-ontologies

This test works as follows:

- An initial draft model created by the modeler is loaded, which is shown below. This model contains only one activity described by annotations of type FC. Besides, the whole models is annotated with one requirement and one constraint, describing properties to be satisfied at process level.

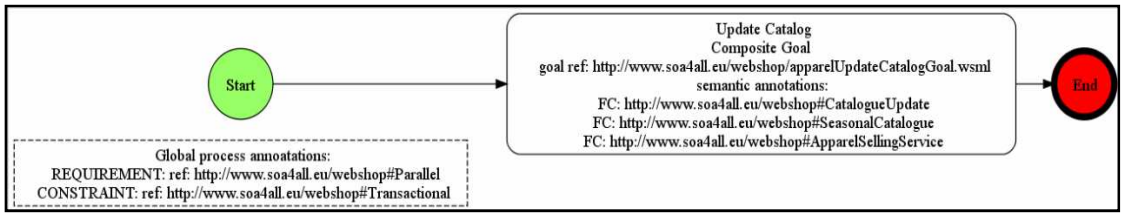

Figure 20 – WP9 input update catalogue process.

- Test invokes method ***resolveGoalWithTemplate*** of DTComposer service to resolve the unique draft input model activity with one template. This is a way to support domain specific service discovery within business process modeling, invoking DTC service.

String outputModelXml = composer.resolveGoalWithTemplate(inputModelXml, goalTarget);

DTC returns one found process model (the best ranked by DTC), which is shown below

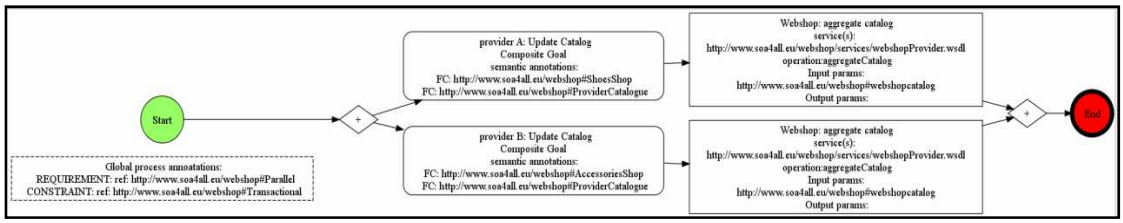

Figure 21 – WP9 output update catalogue process.

- Modeler introduces manual changes in the returned model, adding a new update catalog flow (for another provider), between the exclusive gateways (fork and merge). Modeler adds new global and activity annotations.

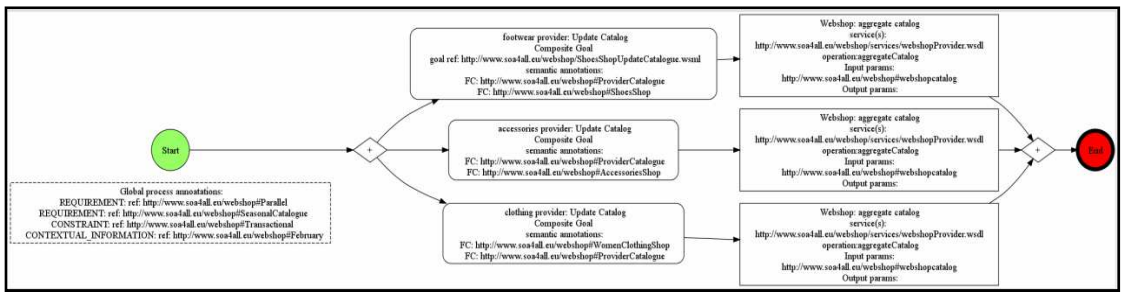

Figure 22 – WP9 manual amended update catalogue process.

- Test invokes method ***resolveProcessWithTemplate*** of DTC service to expand all process model activities with available template.

  outputModelXml = composer.resolveProcessWithTemplate(inputModelXml);

  DTC tries to apply registered knowledge, being able to resolve some unbound activities (not assigned to concrete SWS, but described by activity annotations). Some activities are resolved and others are left unresolved. Best ranked process model is returned.

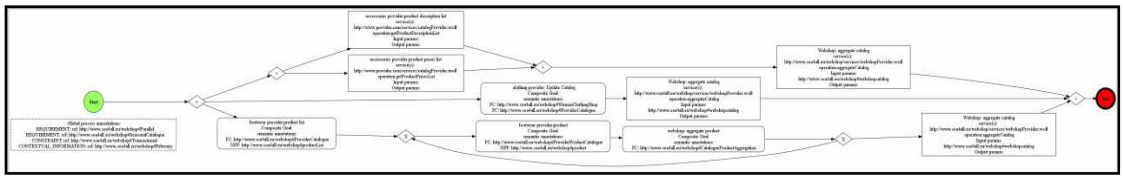

Figure 23 – WP9 output update catalogue process.

- Test invokes method ***resolveProcessWithWS*** of DTC service to resolved unbound process activities, binding them to concrete SWS.

  outputModelXml = composer.resolveProcessWithWS(inputModelXml);

  Some activities are bound and others are let unbound, depending on matching knowledge. Best ranked process model is returned.

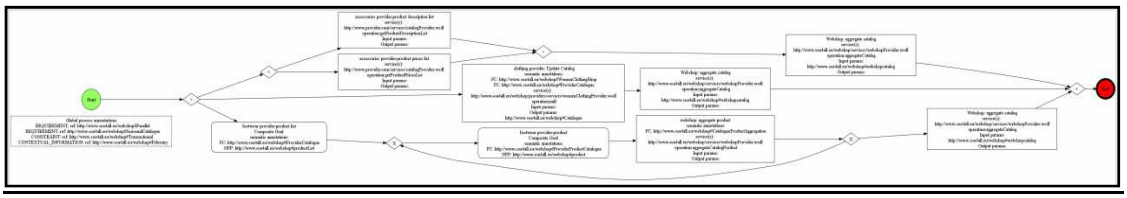

Figure 24 – WP9 output update catalogue process.

- Finally, the modeler may introduce additional manual changes in the model, in order to resolved still unbound activities (since the DTC fails to bind them using the available knowledge).

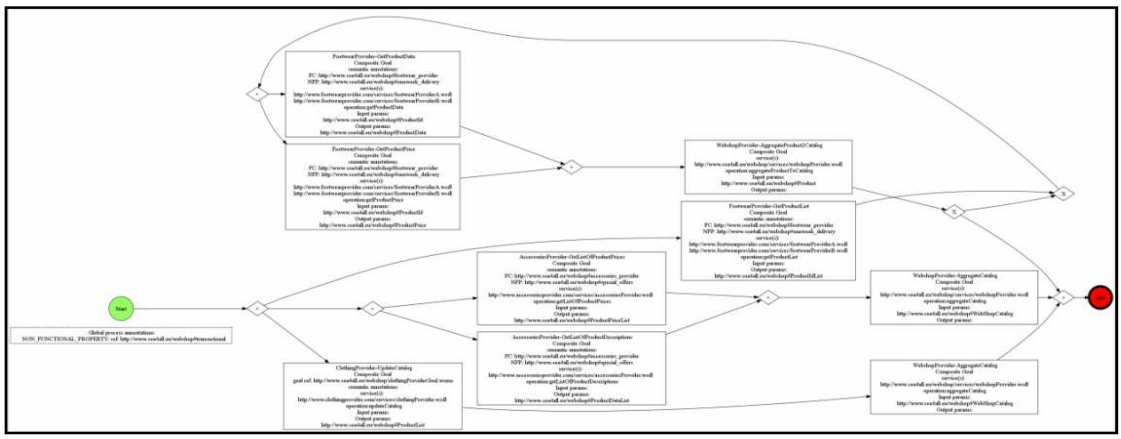

Figure 25 – WP9 manual amended update catalogue process.

- Then, Semantic Link Operator works on the complete model (that is, all activities resolved to services), using as input model the last one completed manually by the modeler. After applying SLO, the outputs and corresponding compatible inputs are connected, adding to the model necessary connector objects. Connectors contain information how to transform resulting output of service invocation into proper input for next service. On the picture below, connectors are show as a green circles, and arrows represent dataflow.

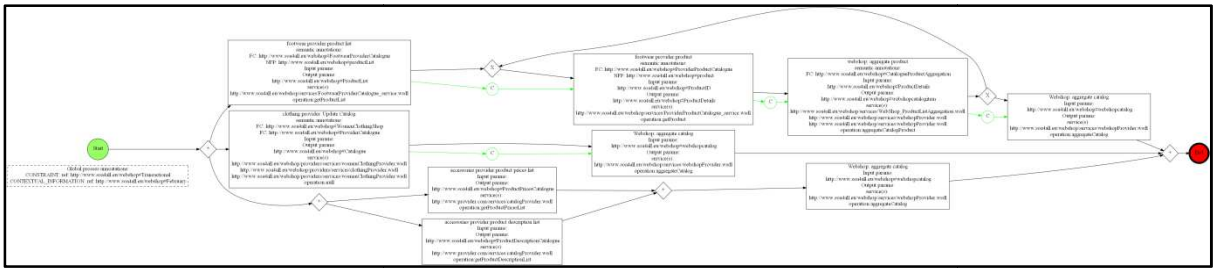

Figure 26 – WP9 update catalogue process with some data flow connectors.

After this step, the process is ready for further optional optimization (see next paragraphs) or deployment into the Execution Environment using the Deployer (D6.5.2).

Optimizer

The Composition Optimizer in Use with WP9 Scenario.

This test models a variant of the WP9 E-Commerce Webshop Catalogue update process. The modelling process is a combination of manual modelling (using the Process Editor, simulated in this test programmatically, that is, models created manually are in the test created programmatically) and assisted modelling using the Optimizer service. This test explains a possible jointly process modelling phase in which modeller (user) using the Process Editor, the result of the DTComposer service and the optimizer service are mutually collaborating.

The composition optimizer requires different sources of information along this scenario:

- The input composition model computed by the DTComposer
  - *Its LPML representation* is also stored locally for test purpose in the following location: *SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/resources/input/NonOptimalProcess.xml*
  - Its graphical representation is stored in the following location: *SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/resources/input/ NonOptimalProcess.svg*
  - Its UML representation is stored in the following location: *SOA4All-service-construction/SOA4All-service-construction-*

optimizer/src/main/resources/input/ WP9\%Update\%Catalog\%Final\%Process\%with\%Connector.pdf

- A configuration file stored in the following location: *SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/resources/configuration.props covering:*
- GAs parameters to optimize the composition such as parameters of its cost function (i.e., AVAILABILITY_OBJECTIVE, PRICE_OBJECTIVE, RESPONSETIME_OBJECTIVE, MATCHING_QUALITY_OBJECTIVE, ROBUSTNES_OBJECTIVES), the number of generation and the population considered by the GA process, the reasoner URL considered to compute semantic similarities between services, the URL of the Services location.
- *A local repository of WSMO semantic web services descriptions located under: SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/resources/data/ISWC2009/. In this repository ten candidate services are available to replace any service of the composition returned by the DTComposer (*Figure 28 *illustrate two examples of WSML services)*

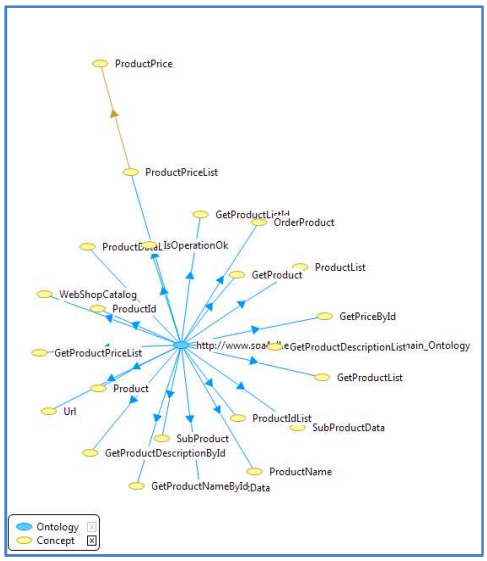

Figure 27 Illustration of a WP9 Domain Ontology

- *A* WSMO domain ontology (all references to semantic annotations refer to concepts in this ontology – see a brief overview in Figure 27) located under: *SOA4All-service-construction-optimizer/src/main/resources/data/ISWC2009/DomainOntology.wsml*

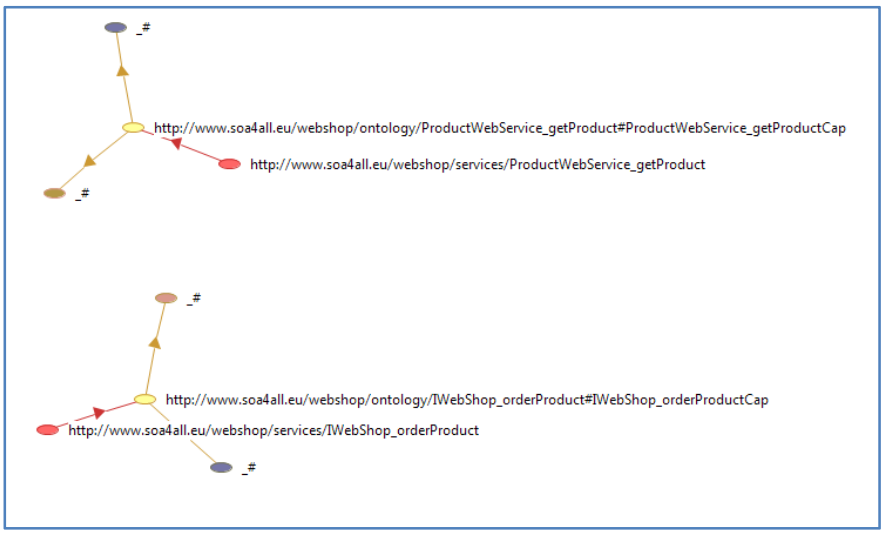

Figure 28 Illustration of two WP9 WSML services

The composition optimizer provides different results along this scenario:

- The output composition model computed by the Optimizer
  - *Its LPML representation* is also stored locally for test purpose in the following location: *SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/resources/output/OptimalProcess.xml*
  - Its graphical representation is stored in the following location: *SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/resources/output/ OptimalProcess.svg*

This scenario works as following:

- The input composition model provided by the DTComposer is loaded. The graphical representation is shown in Figure 29.

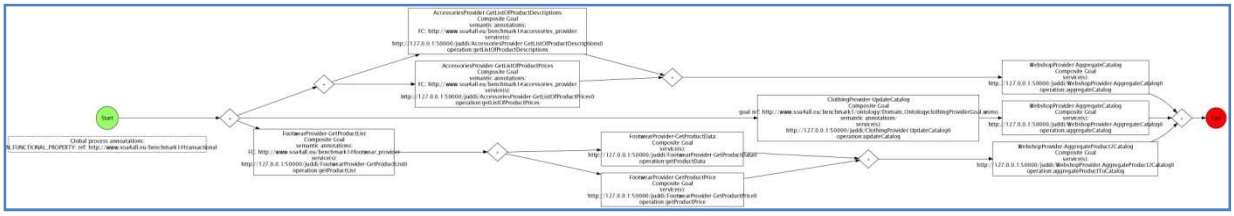

Figure 29 Non-Optimal Composition

- The method: */SOA4All-service-construction/SOA4All-service-construction-optimizer/src/main/java/org/SOA4All/core/Core.java* invokes the method *Main* of the Optimizer service to compute the optimal composition
  - First of all a table (Figure 30) with initial services and semantic links binding is presenting to the end user.

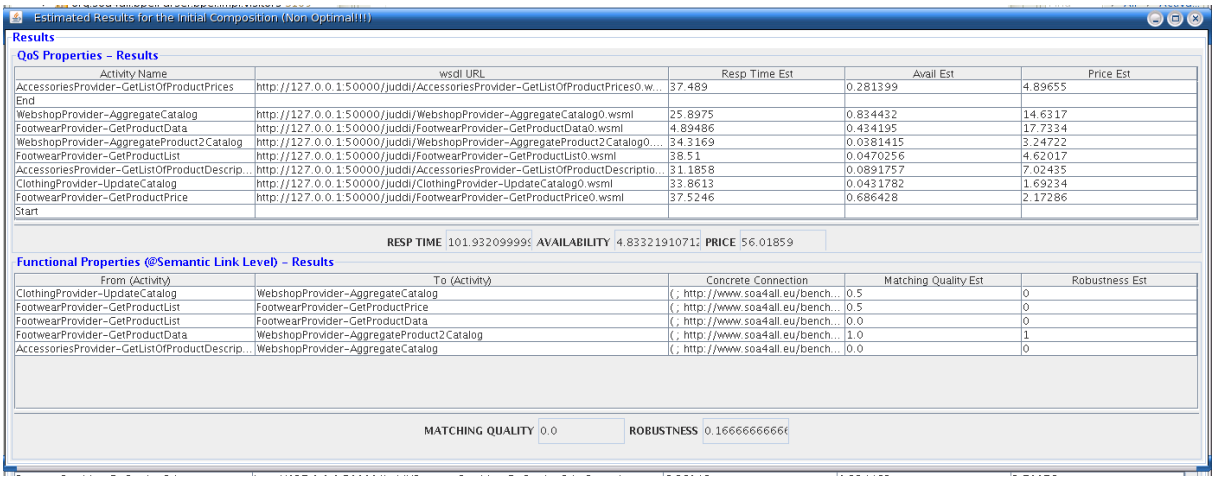

Figure 30 Initials Services and Semantic Links Binding

- Then the optimal composition is computed. Then, a table (see Figure 31) with final and optimal services and semantic links binding is presenting to the end user.

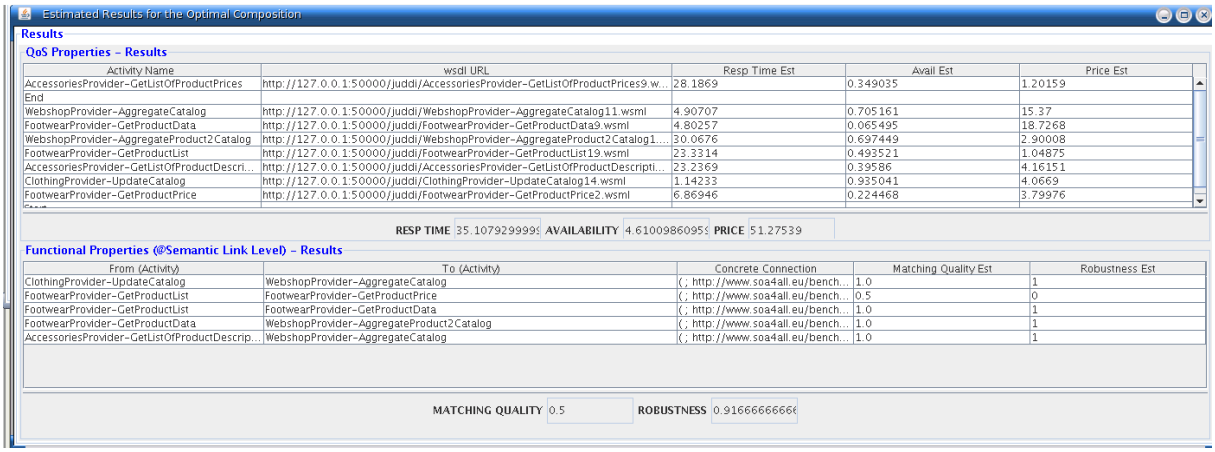

Figure 31 Final and Optimal Services and Semantic Links Binding

- The output composition model computed by the composition optimizer is returned to the end-user. The graphical representation is shown in Figure 32.

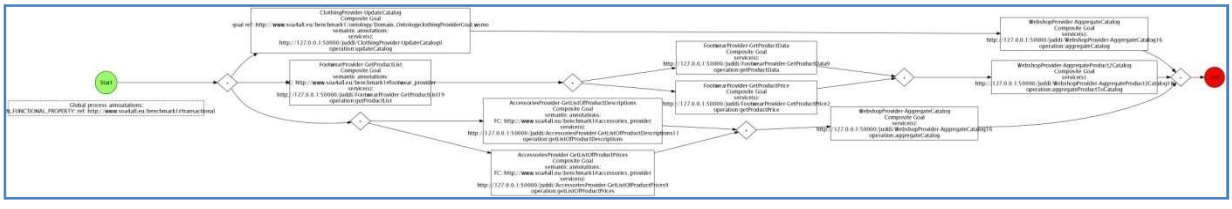

Figure 32 Optimal Composition