

Project Number: **215219**
 Project Acronym: **SOA4ALL**
 Project Title: **Service Oriented Architectures for All**
 Instrument: **Integrated Project**
 Thematic Priority: **Information and Communication Technologies**

D6.3.2. Advanced Specification Of Lightweight, Context-aware Process Modelling Language

Activity:	Activity 2 - Core R&D Activities	
Work Package:	WP 6 - Service Construction	
Due Date:	M18	
Submission Date:	14/09/2009	
Start Date of Project:	01/03/2008	
Duration of Project:	36 Months	
Organisation Responsible of Deliverable:	SAP	
Revision:	25	
Author(s):	Florian Schnabel SAP Lai Xu SAP Yosu Gorrongoitia ATOS Mateusz Radzimski ATOS Freddy Lecue INRIA Gianluca Ripa CEFRIEL Sven Abels TIE Sean Blood TIE Martin Junghans UKARL Adrian Mos INRIA Nikolay Mehandjiev UNIMAN	
Reviwers:	Barry Norton UIBK Matthias Born SAP	

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
01	12/06/09	ToC defined according to the Wiki, rough descriptions provided to each section according to the Wiki	Florian Schnabel (SAP)
02	15/06/09	Further work on section descriptions	Florian Schnabel (SAP)
03	15/06/09	Refinement of data flow section	Lai Xu (SAP)
04	15/06/09	Refinement of ToC	Florian Schnabel (SAP)
05	18/06/09	Comments and updates by UNIMAN	Freddy Lecue (INRIA)
06	19/06/09	Update of partner contributions	Florian Schnabel (SAP)
07	19/06/09	Update of contents based on Freddy's comments	Florian Schnabel (SAP), Freddy Lecue
08	29/06/09	Final allocation of partner contribution, update deliverable based on ATOS' comments.	
09	30/06/09	Draft of element specification, extension of data flow section	Florian Schnabel, Lai Xu (SAP)
10	30/06/09	Integration of section about context-awareness	Matthias Born (SAP)
11	30/06/09	Integration of Yosu's comments	Florian Schnabel (SAP)
12	02/07/09	Refinement of element descriptions	Florian Schnabel (SAP)
13	05/07/09	Refinement of element descriptions, ATOS input for goals and implementation, TIE input for section 5	Florian Schnabel (SAP), Yosu Gorronogoitia(ATOS), Sven Abels (TIE)
14	15/07/09	Integration of semantic annotation section written by ATOS	Yosu Gorronogoitia (ATOS)
15	15/07/09	Comments on section 5	Sven Abels (TIE)
16	20/07/09	Update mediation, integrate data flow contribution of UNIMAN, moved data mediation in 3.3 to 4.3	Florian Schnabel, Lai Xu (SAP), Freddy Lecue (INRIA)
17	04/08/09	Update section 4.1	Florian Schnabel (SAP)

18	07/08/09	Update APIs in section 5.1	Sean Blood (SAP)
19	11/08/09	Update element description, section 5.1, goal section	Martin Junghans (UKARL), Sean Blood, Florian Schnabel (SAP)
20	12/08/09	Finalize section 3	Florian Schnabel (SAP)
21	12/08/09	Update section 4, 5, 6	Florian Schnabel (SAP), Martin Junghans (UKARL)
22	18/08/09	Update of section 5.2	Sven Abels (TIE)
23	27/08/09	Update of section 4	Florian Schnabel (SAP)
24	01/09/09	Update conclusion	Florian Schnabel (SAP)
25	14/09/09	Final editing	Malena Donato (ATOS)

Table of Contents

EXECUTIVE SUMMARY	7
1. INTRODUCTION	8
1.1 PURPOSE AND SCOPE OF THE DELIVERABLE	8
1.2 STRUCTURE OF THE DOCUMENT	8
2. SUMMARY OF THE RECENT WORK IN T6.3	9
2.1 SUMMARY OF D6.3.1	9
2.2 SUMMARY OF RELATED WORK	9
3. LIGHTWEIGHT, CONTEXT-AWARE PROCESS MODELLING	11
3.1 ABSTRACTION LAYERS FOR THE LPML	11
3.2 SEMANTIC ANNOTATIONS FOR PROCESSES AND ITS ELEMENTS	12
3.3 CONTEXT-AWARE PROCESS MODELLING PRINCIPLES	14
3.4 SUMMARY AND REMARKS	16
4. LIGHTWEIGHT, CONTEXT-AWARE PROCESS MODELLING LANGUAGE	18
4.1 METAMODEL OF LIGHTWEIGHT PROCESS MODELING LANGUAGE	18
4.1.1 <i>Design Process for Lightweight Process Modelling and Execution</i>	19
4.1.2 <i>Modeller's view of the LPML Metamodel</i>	20
4.1.3 <i>Complete LPML Metamodel</i>	21
4.1.4 <i>LPML Metamodel Elements</i>	22
4.2 PATTERNS AND TEMPLATES	27
4.3 GOALS	27
4.4 DATA FLOW PERSPECTIVE	32
4.4.1 <i>Data Manipulation and Operators</i>	32
4.4.2 <i>External View of Data Flow Manipulation</i>	33
4.5 SUMMARY AND REMARKS	34
5. LPML API: REQUIREMENTS, DESIGN AND IMPLEMENTATION	35
5.1 LMPL API REQUIREMENTS	35
5.2 LPML API SPECIFICATION	36
5.2.1 <i>Using EMF as a Basis for the LPML API</i>	41
5.2.2 <i>Serialization in RDF/S</i>	42
5.2.3 <i>Transformation into BPEL</i>	43
5.3 SUMMARY AND REMARKS	49
6. LANGUAGE EVALUATION	50
6.1 USER CATEGORIZATION	50
6.2 COMPLETENESS AND EXPRESSIVENESS OF THE LPML	50
6.3 PATTERN-BASED ANALYSIS OF THE LPML	52
6.4 RECOMMENDATIONS	54
7. CONCLUSIONS	55
8. REFERENCES	56
ANNEX A. SAMPLE FILE	58
ANNEX B. TERMINOLOGY	67
ANNEX C. LPML AND SEMANTIC ANNOTATION LANGUAGES	69
ANNEX D. EVALUATION INFORMATION	70

Glossary of Acronyms

Acronym	Definition
B2B	Business-to-Business
BPA	Business Process Analysis
BPEL	Business Process Executable Language
BPM	Business Process Management
BPML	Business Process Modelling Language
BT	British Telecom
CCTS	Core Components Technical Specification
D	Deliverable
ebXML	Electronic Business using eXtensible Markup Language
EPC	Event-driven Process Chains
IDE	Integrated Development Environment
IT	Information Technology
PHP	PHP Hypertext Pre-processor
QoS	Quality of Service
SAP	Systeme Anwendungen und Produkte
SDK	Software Development Kit
SOA	Service-Oriented Architecture
T	Task
UML	Unified Modelling Language
WP	Work Package
WSMO	Web Service Modelling Ontology
YAWL	Yet Another Workflow Language

List of Figures

Figure 1: Lightweight process modelling language stack	11
Figure 2: Semantic annotations for LPML elements.....	13
Figure 3: Abstraction of a Context-Driven Process Flow Scenario	15
Figure 4: Combined Abstraction of a Context-Driven Process Flow Scenario	15
Figure 5: Design Process for Lightweight Process Modelling.....	19
Figure 6: Modeller's view of the LPML metamodel	20
Figure 7: Semantic annotations for the LPML	21
Figure 8: Complete LPML metamodel	22
Figure 9: Reference to WSMO-Lite ontology	30
Figure 10 LPML Goal concept.....	31
Figure 11: Example for LPML usage in the SOA4All composer (Task 2.6)	37
Figure 12: Additional helper classes for the LPML API	41
Figure 13: LPML to BPEL transformation chain	47
Figure 14: Simplified representation of the STP-IM Metamodel	48
Figure 15: Relation of LPML to semantic annotation languages	69

List of Tables

Table 1: Graphical symbols for the Lightweight Process Modelling Language	9
Table 2 : Description of Process	23
Table 3: Description of ProcessElement and its children	23
Table 4: Service and goal description	24
Table 5: Elements for semantic annotation.....	25
Table 6: Elements for the data flow handling	26

Executive Summary

Existing process modelling languages and especially executable process modelling languages are not designed for non-experienced users. In SOA4All we have therefore introduced Lightweight Process Modelling seeking to lower the entry barrier for process modelling. Non-experienced users get advanced guidance during the modelling activities.

This deliverable will provide an advanced specification of the lightweight process modelling methodology and Lightweight Process Modelling Language (LPML) as described in D6.3.1. We will describe in more detail the identified three design principles of lightweight modelling. These design principles comprise abstraction of process models, the use of semantic annotations, and context-awareness. In order to realize these design principles we have created new elements for the LPML. Selected concepts of existing process modelling languages like BPMN and BPEL complement the LPML. We will present a coherent metamodel of the elements, properties, and relationships.

On the programmatic perspective, the LPML requires to be managed by an API that abstracts and hides the complexities of the LPML elements and their concrete serialization formats to the programmer. This deliverable describes as well the LPML API, which provides programmatic process modeling and serialization support, either for storage as RDF and transformation into SOA4ALL extended BPEL 2.0¹ or other executable languages.

The LPML is the SOA4All language for process modelling used in the entire project. The graphical, abstract process models are created by the end-user using the process editor of T2.6. These abstract process models are enhanced by the composer and the optimizer of T6.4. Finally the process models represented in the LPML are executed by the execution engine developed in T6.5.

¹ We refer to the extended BPEL language used by D6.5.1 Execution Environment

1. Introduction

1.1 Purpose and Scope of the Deliverable

Existing process modelling languages and especially executable process modelling languages are not designed for non-experienced users. In SOA4All we have therefore introduced Lightweight Process Modelling seeking to lower the entry barrier for process modelling. Non-experienced users will be able to abstract from composition details and get advanced guidance during the modelling activities.

This deliverable will provide an advanced specification of the lightweight process modelling methodology and Lightweight Process Modelling Language (LPML) as described in D6.3.1. We will describe in more detail the identified three design principles of lightweight modelling. These design principles comprise abstraction of process models, the use of semantic annotations, and context-awareness. The LPML is combination of a selection of appropriate, existing process modelling concepts, elements, and artifacts and new modelling concepts. The reused concepts are mainly defined in BPMN and BPEL. We based our selection on literature analysis (Koehler and Vanhatalo 2007; zur Muehlen and Recker 2008) and the requirements of the use cases. Furthermore a coherent metamodel of the LPML and its elements, properties, and relationships will be presented. This metamodel will provide specific elements in order to implement the mentioned design principles.

On the programmatic perspective, the LPML requires to be managed by an API that abstracts and hides the complexities of the LPML elements and their concrete serialization formats to the programmer. This deliverable describes as well the LPML API, which provides programmatic process modeling and serialization support, either for storage as RDF and transformation into SOA4ALL extended BPEL 2.0² or other executable languages.

The LPML is the SOA4All language for process modelling used in the entire project. In order to enhance the abstract, graphical models by execution details, a specific design process will be set up. The abstract, graphical process models are created by the end-user using the process editor of T2.6. These abstract process models are enhanced by the composer and the optimizer of T6.4. Finally the process models represented in the LPML are executed by the execution engine developed in T6.5. The detailed proceeding of each task is covered by this deliverable.

The contents of the proposed process composition language has to be evaluated in order to prove the applicability in practice. We will follow an evaluation approach by appealing to the concept of ontological completeness and coverage. Evaluating a language would be strongly dependent on both the target users and usage of the language. The target usage of the language is covered by SOA4All use cases.

1.2 Structure of the Document

This document is organised as follows. Section 2 covers a summary of the recent work done in T6.3 as well as related work. The design principles of the lightweight process modelling are addressed by Section 3. Afterwards, in Section 4, the LPML is described in more detail. The requirements, design, and implementation of the LPML API is presented in Section 5. Section 6 gives an insight into the envisaged evaluation of the LPML. Finally, Section 7 concludes the document.

² We refer to the extended BPEL language used by D6.5.1 Execution Environment









2. Summary of the Recent Work in T6.3

2.1 Summary of D6.3.1

Table 1 recalls the graphical elements for the LPML. We adopted seven notations from BPMN and added two goal related notations. We suppose that these graphical symbols are only understandable by advanced modellers. The process editor developed by T2.6 uses even more abstracted symbols. In this deliverable D6.3.2 we will provide detailed element descriptions. However, the graphical symbols are only one option to visualize the element descriptions of Section 4.1.

In order to better abstract the language and make it more lightweight we will dismiss the graphical representation of composite activity goals as described in D6.3.1.

Table 1: Graphical symbols for the Lightweight Process Modelling Language

	Starts a process flow		Ends a process flow
	An activity is a unit of work, the job to be performed		Sequence Flow defines the execution order of activities
	Exclusive Gateway When splitting, it routes the sequence flow to exactly one of the outgoing branch on conditions. When merging, it awaits one incoming branch to complete before triggering the outgoing flow		Parallel Gateway When used to split the sequence flow, all outgoing branches are activated simultaneously. When merging parallel branches it waits for all incoming branches to complete before triggering the outgoing flow
	Inclusive Gateway When splitting, one or more braches are activated based on branching conditions. When merging, it awaits all active incoming branches to complete.		
	Atomic Activity Goal Atomic goals are those that are associated with a single concrete activity, involving just one step of computation.		

The main design principles have already be introduced in D6.3.1 including the LPML stack, context-awareness, the use of patterns, templates, and goals. We will refer to D6.3.1 for a more detailed description. However, the main aspects are repeated in the appropriate sections of this deliverable D6.3.2.

2.2 Summary of Related Work

The related work of our LPML is covered by D6.1.1 and D6.3.1. In the following, we will only present the work that has not been described yet by these two deliverables. We will introduce

two BPEL extension mechanisms that influenced the design of the LPML.

BPEL-Light (Nitzsche, van Lessen et al. 2007b): The purpose of BPEL-Light is to make BPEL independent of the Web Service Stack. Therefore, new elements are defined replacing those elements that reference to WSDL-based elements or elements referencing WSDL interfaces. Being independent of WSDL means that process tasks or activities can reference WSMO goals, WSMO-Lite services or other service interfaces.

BPEL4SWS (Nitzsche, van Lessen et al. 2007a): This BPEL dialect defines references to WSMO goals and to interfaces described in OWL/S. BPEL4SWS therefore extends BPEL similar to BPEL-Light by elements that replace all those elements depending on WSDL. BPEL4SWS is a kind of instantiation of BPEL-Light. The semantic web services will be grounded to WSDL.

3. Lightweight, Context-aware Process Modelling

As described in D6.3.1 lightweight process modelling is a combination of techniques that seek to lower the entry barrier for process modelling. In the following, we will present the identified aspects that are the different abstraction layers of the lightweight process modelling stack, semantic annotations, and context awareness.

3.1 Abstraction Layers for the LPML

As depicted in Figure 1 our modelling stack comprises three layers in general. The top layer forms a graphical representation layer that has been designed for non-modelling experts. It hides as much information as possible from the user by relying on a set of simple and easy-to-understand symbols. This new layer has been defined in SOA4All and is described in more detail in Section 4.1.2. Our framework also allows modelling experts to model processes in a more detailed graphical representation. This is addressed by the middle layer in Figure 1. However, this layer is not part of the research work of SOA4All since there are already a couple of languages existing, like BPMN or YAWL. The bottom layer provides the textual representations of processes. It forms the canonical format of our process model and is described in the SOA4All LPML. The LPML metamodel is based on the Eclipse Modelling Framework (EMF) (Foundation 2009) metamodel and can have various representations, e.g. in UML, Java or XML. It is then easy to transform the LPML models into XPD L or WS-BPEL models or to serialize the models in RDF-S in order to be stored in a semantic space. The LPML models are based on the information given by the user. The models will be then enhanced in several steps in order to contain enough information for execution. Besides the provision of a basis for executable models, the textual representation will facilitate exchange and reuse of process models independently of the graphical representation.

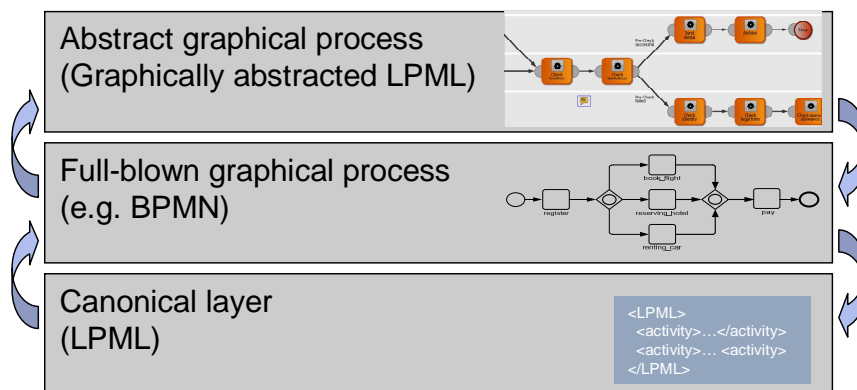


Figure 1: Lightweight process modelling language stack

Model Transformation

Starting point for process modelling in SOA4All is the process editor as part of the SOA4All Studio. The process editor is used to create the abstract graphical processes as depicted in the upper layer of Figure 1. In the backend the process editor directly creates an LPML process model represented in Java. Hence, a simple model transformation from the graphical abstraction into an LPML representation is performed within the process editor.

As described above the LPML can have various representations. It will be easy to transform an XML representation of the LPML model into XPD L and hence symbolize it graphically in BPMN. However, this model transformation from LPML into XPD L is not addressed by

SOA4All.

3.2 Semantic Annotations for Processes and its Elements

In this Section, we will describe the semantic annotations for activities and the process as a whole. Semantic annotations provide additional, machine-readable information that refers to ontology concepts. Key for SOA4All is the lightweightsness of these annotations. The annotations will be integrated into the LPML as WSMO-Lite annotations and are referenced by WSDLs through SAWSDL annotations. An overview of the relation of the LPML to existing semantic annotation languages can be found in Annex C.

The semantic annotations for processes and its elements mainly depend on the adoption of knowledge representation models, such as ontologies. The benefit will be to partially automate the modelling of processes using some domain and context specific knowledge. The project SUPER defined five ontologies in order to combine semantics and process modelling³. Contrary to SUPER, we will define the semantic extensions in a more lightweight way (similar to the SAWSDL approach for WSDL). We will define semantic annotations for process elements such as activities, goals, etc. and for the process as a whole. The semantic annotations will then reference to concepts or instances of existing domain specific ontologies for SWS like WSMO, WSMO-Lite, or Micro-WSMO or they will include logical expressions or metadata.

Hereafter, we will explain how semantic annotations can be used for the lightweight modelling methodology and how they are included in the LPML metamodel. Annotations are used to complement the syntactic description of processes and main modelling elements by providing a semantic meaning. The purpose of such annotations is to support some sort of automation during both modelling and execution of processes. This support will be offered by the SOA4ALL modelling tools: Process Editor, DT Composer, Optimizer and Runtime Executor. Annotations can also be used to check the fulfilment of models with the requirements and constraints. In addition, human modellers can also use annotations to share a better understanding of process models and their modelling elements, patterns and templates.

The semantic annotation approach in LPML is depicted in Figure 2. The SemanticAnnotation class contains a URI reference to an ontology concept or instance. The annotation is ontology-agnostic, that is, it can reference concepts described by different ontologies like WSMO, WSMO-Lite, or Micro-WSMO. The modelling tools will properly interpret those annotations. SemanticAnnotation also includes an optional expression property that can be used either to contain a logic expression or a literal (supporting the expression of metadata: keyword=value). That could be useful to describe authoring process information using, for instance, the Dublin Core ontology. Optionally modelling tools can use AnnotationType enumeration to categorize the taxonomy of supported annotation types.

³ See <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-251/paper13.pdf>

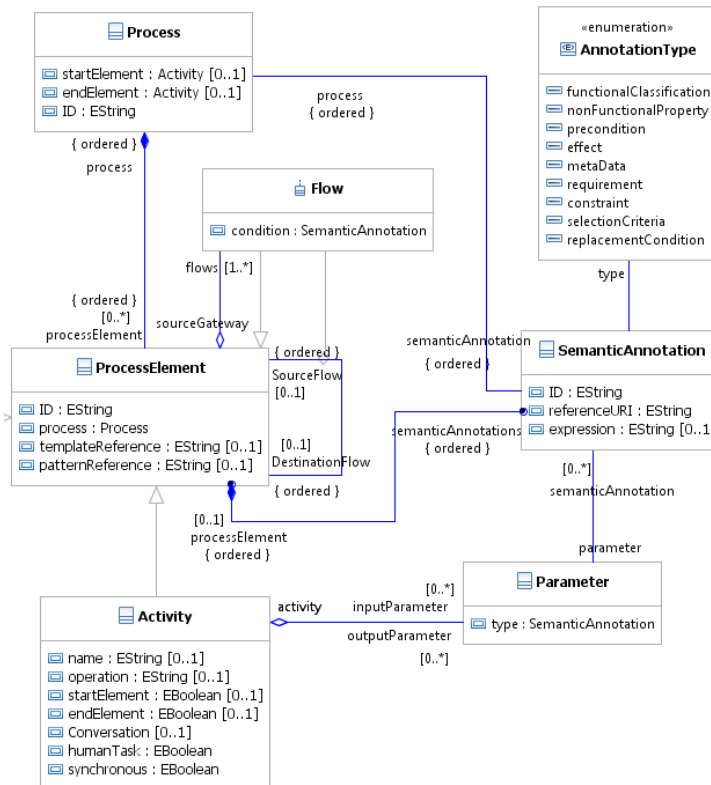


Figure 2: Semantic annotations for LPML elements

Modellers can use annotations to provide semantic meanings to process models as a whole or to some modelling elements. The annotations have to follow a certain type defined for the LPML. We can use these annotations to describe:

- **Requirements:** A requirement references an ontology concept. The requirement range is therefore the ontology concept range. Accepted requirement values are instances of that concept. Requirements are used to describe processes and its elements (mostly activities/goals). At process level, requirements can be used to specify the domain specific scope for the process and other global requirements, for instance: bureaucratic procedure, recorded procedure, acknowledge procedure, etc. For instance in WP7 scenarios, a requirement can specify an eGovernment registration domain (comprising any formal application/procedure registration). The expected checking level could be determined from the domain specific ontology used to reference annotations or could be explicitly stated by the human modeller using another requirement annotation. When applied to activity goals, annotations described the functional classification of desired SWS, as it is explained in Section 4.1. Examples of requirements for activities are, for instance: document and recording management, application/procedure checking, failure management, archiving, payment (electronic, tax office), electronic record management, physical document management, etc.
- **Constraints:** we model constraints using annotations as requirements are. Nevertheless, constraints and requirements are conceptually dissimilar since requirements have positive meaning while constraints have restrictive or negative implications, limiting the scope or acceptable functionality. As requirements, they can be applied either to processes themselves or to their modelling elements: activities/goals. Examples of constraints are free/non-free procedure, checked procedure, legal checking, reject unchecked or failed applications, credit/debit card payment, etc.

- Non-functional properties: they can be applied either to processes themselves or to their modelling elements. Examples of restrictions are: the geographical location of external partner services (i.e. payment services), response times, cost ranges, number of concurrent invocations supported, secure transactions, acknowledge transactions, etc. At process level, non-functional properties may impose global non-functional restrictions upon external partners participating in all the activities of the process. Non-functional properties could also be used to determine possible template expansions, for instance, selecting from available payment templates, to set human tasks or to set the role in order to validate or execute the process.
- Metadata (parameter=value) supporting, for instance authoring information such as author, creation date, versions, revisions, etc.
- Logical expressions, expressed in any logical language supported by the modelling tools, which can be used as conditions of Exclusive Gateway flows or for Goal preconditions/effects.
- Other annotations specifically related to Activities/Goals, such as preconditions and effects that are described in Section 4.3.

LPML semantic annotations can be in principle optionally attached to any process element including the process itself. Whether annotations are optional or mandatory for a certain process element will be described in Section 4.1.4

- Process: annotations are used to describe global requirements, constraints, NF properties and metadata. These annotations have global scope and precedence over annotations of the process modelling elements.
- Activity: annotations are mainly used to describe NF properties of attached SWS, supporting optimization and self* features during runtime.
- Goal: annotations are used to describe preconditions, effects, functional classification and NF properties of wanted SWS.
- Gateway/Flow: annotations are used to describe flow conditions in Exclusive Gateways. If the condition is true-evaluated that flow is followed, otherwise discarded.
- Parameter: Activities have placeholders for input variable (message in) and output variable (message out). Both can be annotated with semantic concepts that describe their types. In case of abstract activities (Goals), input and output annotations are used to complement the goal description. Alternative, in case of concrete activities, input/output annotations could be used to match the types that appear within the SAWSDL description of the bound SWS.
- Connector: annotations can be used to describe some activity connector properties, such as truncating elements (when passing from parameters of one activity to another) and the connection type. They are established by the SOA4ALL modelling tools.

3.3 Context-aware Process Modelling Principles

This section will describe the context-driver principle. In SOA4All, we will mainly address three aspects, the dynamic appearance of structures, the dynamic linguistic representation of components, and the component instantiation.

The basic idea of context awareness is not new. In fact, the concept context has been researched for many years within related disciplines (Akman and Surav 1996). In the domain of artificial intelligence, context is usually defined as the generalization of a collection of assumptions (McCarthy 1993) for both knowledge management and communication

management (Brezillon and Abu-Hakima 1995). In computer science, one of the most commonly cited definition was given by Dey (Dey 2000) who defines context as "any information that can be used to characterize the situation of an entity" and categorises it into four dimensions of location, identity, time and activity. Dey (Dey 2000) also states that context should be modelled and formalized.

Within the last years, there have been initial approaches that consider context awareness within business process models (Rosemann and Recker 2006; Rosemann, Recker et al. 2006; Saidani and Nurcan 2007). Rosemann et al. (Rosemann, Recker et al. 2008) contemplate on the situations which affect the flow of business process models. In their work, they do not focus on structural differences and how context methods can actually change the flow of these models. Their focus is set on the formalization of these situations in the form of process contexts. The context awareness aspects which are developed within our work consider the context-driven differences in the structural appearance of business process models and focus on various levels, context-driven terminologies for business artefacts, and context-driven instantiation of abstract activities. Furthermore, our proposal does not focus on any specific application of contexts and neither on any specific context categorisation. It rather contemplates context-driven differences in general on an abstract level. We have already described the context-driver in D6.3.1. In the following, we want to summarize briefly the general idea.

One aspect of the lightweight process modelling environment is to allow for the definition of contextualized processes by supporting the specification of context information sources and their role they play in the process. Hence, our framework integrates the principles of context awareness aspects in the appearance of business process models and their terminology. Context awareness plays an essential role for the dynamic appearance of structures, the dynamic linguistic representation of components, and the component instantiation. The context-driver principle allows to identify, store, and represent a business process artefact only once while specifying the differences depending on specific context categories (e.g., business process, industry, country, etc). Context awareness can be applied on each layer of our process modelling language stack. However, it is of great importance in the canonical process representation. Based on that we can define various ways in symbolizing the context in the graphical process models. Figure 3 depicts a scenario with two abstract business process models in two different business contexts. The activities of these models are marked with capital letters.

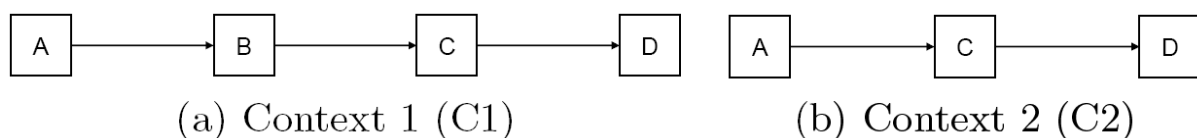


Figure 3: Abstraction of a Context-Driven Process Flow Scenario

We can recognise that both business process models are similar. In fact, they are the same, apart from the aspect that activity B is omitted in business context C2. Without context awareness we would have kept two separate business process model representations. However, with the usage of context awareness we are able to have one business process model representation which is able to consider differences regarding its appearance in a specific business context, which is depicted in Figure 4.

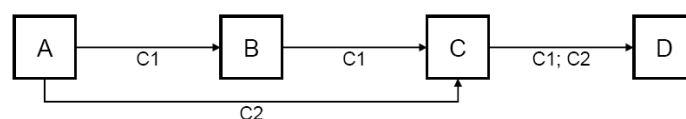


Figure 4: Combined Abstraction of a Context-Driven Process Flow Scenario

Process structures embrace context awareness aspects of all levels. They define the context-driven flow of business processes and contain activities. In addition, relations between activities and other business artefacts are context dependent as well.

Furthermore, every process artefact has a unique semantic representation, and the context in which it is used assigned. In a process model this representation constitutes labelling and naming. A problem why business process models are sometimes not easy to understand is that business artefacts can be called differently depending on the business situation or the business sector. For example, a financial department uses the term 'Invoice', whereas a sales department uses the term 'Bill' for the same entity. Thus, the term actually depends on the business context. Although these different business terms refer to the same entity, their business context is very important for two reasons. The first reason is the selection of the correct business term for the creation of labels. The second reason is the selection of business artefacts based on their labels, which might result in ambiguous situations without business contexts. The general idea is to create a standardised, consistent, and understandable description of every process artefact using ontologies and to link each process artefact to a specific business context. The primary purpose is to achieve consistency in the naming and to facilitate the understandability of the business process models depending on a business context. For this purpose, the conceptual framework specifies a common process knowledge base using ontologies to which these models can refer. This knowledge base builds the basis for the semantic meaning of business artefacts and business process models and it therefore provides a common understanding regardless of the used business process modelling notation, syntax, or terminology. The content in this process knowledge base is supposed to be maintained on a collaborative basis without much manual user-interaction. The common understandability supports business collaboration within and across companies. We want to give a brief example. Let us assume a modeller creates an activity 'create invoice', which is already available in the knowledge base. Now, the idea is to avoid creating a second instance of this activity, but rather to adjust the valid business context of the activity 'send offer'. Logically, a modeller would first need to define, in which business context he is modelling. As a result, the usage of insufficient business artefacts and terminology can be avoided.

The third aspect of the context driver principle in SOA4All is to provide a frame for potential instantiations of abstract activities. The user will model process activities as a set of requirements and constraints. In order to translate these roughly described activities into goals and services the context information can be helpful in order to preselect or propose potential instantiations.

As one of the SOA4All extensions to existing approaches we will introduce an attribute `contextReference` into the LPML elements containing a reference to a context file. This context file will be read by the process editor, the components provided by T6.4, and the execution engine of T6.5. The `contextReference` can as well be implemented as semantic annotation. However, a coherent concept for context-awareness in SOA4All is not yet provided. We envisage providing that concept by month 30.

3.4 Summary and Remarks

In this Section, we have described the basic principles for lightweight process modelling. We have covered semantic annotations that support automatic discovery, instantiation, composition, and execution of processes and process elements. The annotations contain information such as requirements, constraints, or metadata and are generated by the user or by context information. Besides the semantic annotation and context-awareness of process elements, we have introduced the lightweight process modelling language stack. This

modelling stack provides a graphical representation layer of the process models that is easily understandable by non-expert users. The graphical representation of the LPML models is created by the process editor and addressed in T2.6. The textual representation of the LPML serving as canonical format is as well addressed by SOA4All. We will now go on with the detailed description of the LPML elements.

4. Lightweight, Context-aware Process Modelling Language

In this Section, we will give an insight into the LPML. We will therefore introduce the design process within SOA4All for process modelling and execution. This process describes the necessary steps in order to support the user in creating an executable process model out of the abstract graphical model. Afterwards we will describe the LPML metamodel and its elements in more detail. In order to make the metamodel easily understandable we will provide several views on the metamodel. In addition, we will cover special aspects of the LPML, such as patterns and templates, goals, and the data flow. The patterns, templates, and goals will support the user in modelling the control-flow of its processes. The data flow aspect highlights the process model from another perspective than the control flow. As patterns, templates, and goals do for the control flow we will introduce new means supporting the user in modelling the data flow.

4.1 Metamodel of Lightweight Process Modeling Language

The aim of the LPML is to simplify the work of a process designer hiding technical aspects, performing automatic optimizations and allowing late binding to concrete services and service substitution at runtime. Thus, the metamodel is devised taking into account both the usability of the tool that will be provided to the user and the underlying design process.

In fact, the LPML is devised as visual notation with specific constraints to be used by a process modeller expert in order to create an executable process. The LPML metamodel describes the elements, their properties, the relationships between each element and the constraints applicable in their usage. Some elements of the LPML are not provided directly by the process designer but can be derived automatically by the tools exploiting predefined semantic descriptions and ontologies of services and goals.

In this Section, we will provide first a brief description of the design process that allows to go from a conceptual description of a process (provided by the user) to its execution. Then the metamodel for our LPML is presented in more detail.

Basically, we will define one holistic metamodel comprising two views, the modeller's view on the abstract LPML layer and the backend view of the complete LPML metamodel. As mentioned in Section 3.1, the LPML metamodel is based on the EMF metamodel [50] and is visualized here in UML class diagram notation.

4.1.1 Design Process for Lightweight Process Modelling and Execution

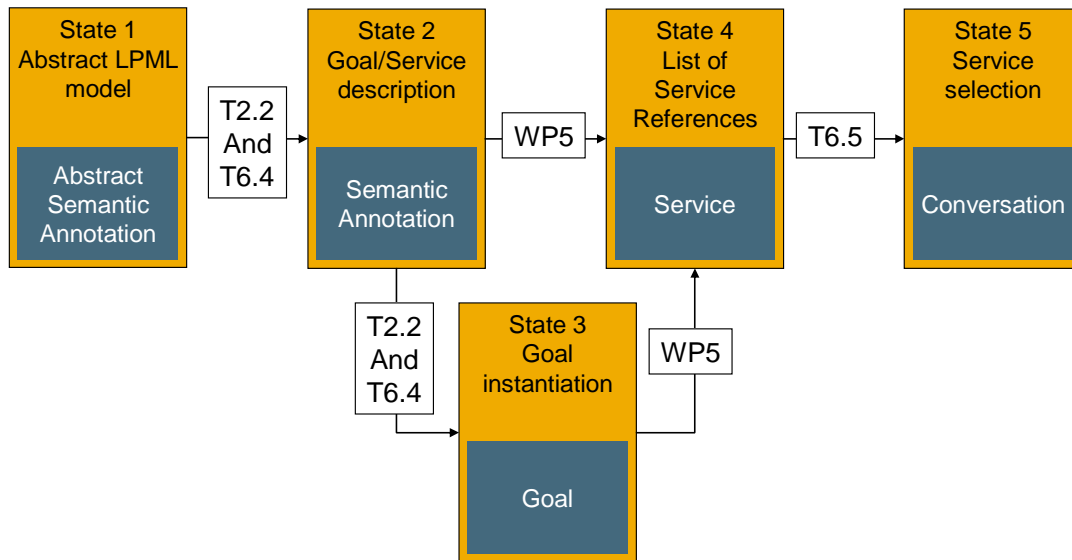


Figure 5: Design Process for Lightweight Process Modelling

Figure 5 depicts the procedure of how to define an executable process starting from an abstract semantic description (expressed in a visual notation) defined by the user, with the tools that automatically perform some of these steps. The procedure is performed in five steps:

1. The first step for the user is to specify, by dragging and dropping graphical elements in the Process Editor, an abstract process model. The modeller's metamodel view (State1 in Figure 5) covers the information that is provided by the user's process model in the process editor. The activities contain abstract semantic descriptions, e.g. in natural language. These semantic descriptions comprise information like requirements, constraints, non-functional properties, or metadata.
2. The second step is to create semantic service and goal annotations out of these rough descriptions. T2.2 and T6.4 will enhance the process model by annotations for the functional classification, non-functional properties, preconditions, and effects. While a goal annotation formulates a request to the characteristics, the service annotation describes a concrete characteristic.
3. Now the T6.4 components can map the semantic annotations to an existing goal or services publicly described in a repository. In case of mapping the annotations to existing goal descriptions, we have to figure out the goal that fits best. The goal instantiation is referred to as state 3 in Figure 5.
4. Either the selected goal out of state 3 or the semantic service annotation is given to the discovery engine of WP5 in order to find a set of appropriate services. The service set is ordered in a list. For each service in the list the ServiceGrounding is instantiated that contains the reference of the service URI. The ServiceGrounding element acts as the GoalConversation and provides the binding to an existing service. Furthermore this view addresses the service replacement at runtime. This is represented by state 4 in Figure 1.
5. The final step is now performed by the execution engine developed in T6.5. This execution engine at runtime selects the best-fitting service out of the list and executes it. Before execution the Execution Engine is able to exploit semantic annotation of

candidate service for automatically generate mapping script necessary for adapting the execution to the actual service interfaces.

Steps 1, 2, 3 and 4 can iterate several times in a cycling design process. The user can start the modelling from each of the states 1, 2, 3, or 4.

The presented design process for the model enhancement is performed in several steps by multiple tasks. In order to make this enhancement procedure better understandable we will provide in the next section several metamodel views dedicated to the model states before and after each step.

4.1.2 Modeller's view of the LPML Metamodel

The LPML metamodel view of the modeller serves as communication means for the abstract graphical layer. The user will only see a subset of elements that are essential for the graphical process representation. Figure 6 provides an UML representation of the modeller's view of the LPML metamodel.

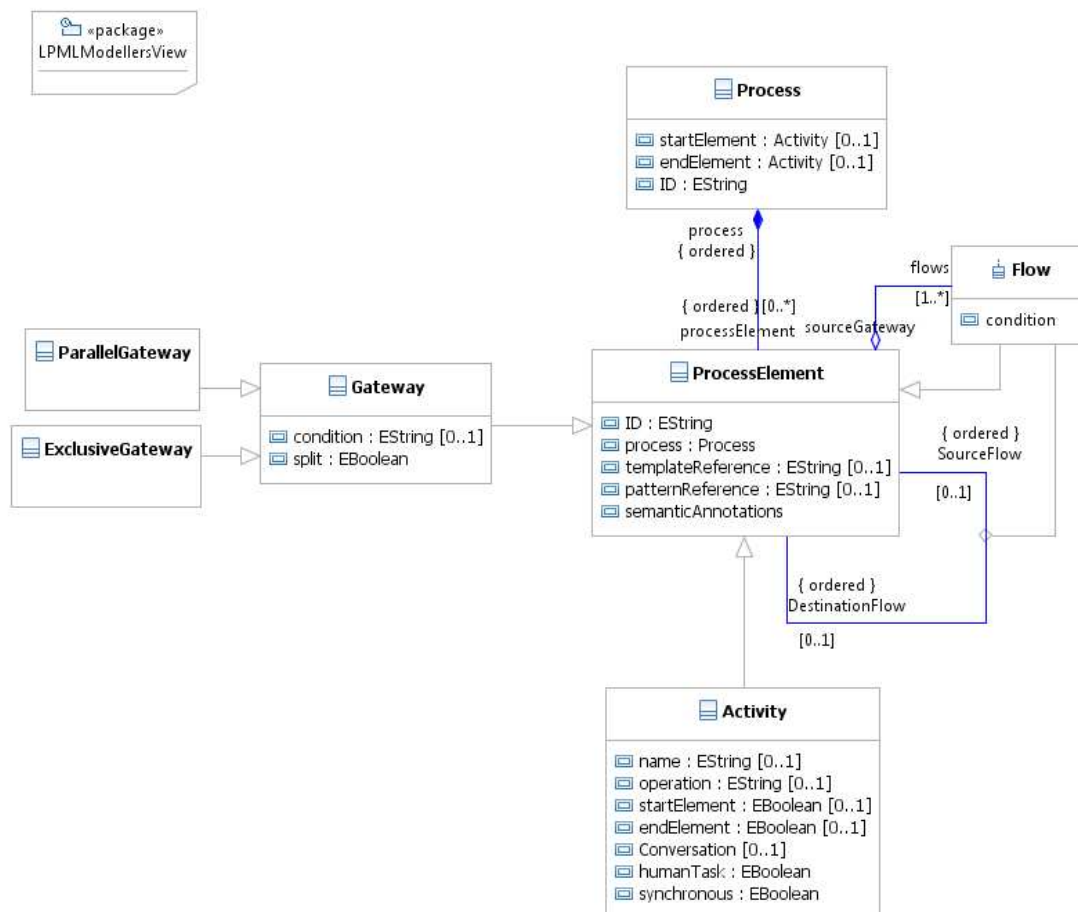


Figure 6: Modeller's view of the LPML metamodel

All the LMPL elements of this view and their characteristics are described in the following.

- **Process** represents the container of all other process elements. It is characterized at least by a start element and an end element and has an unambiguous ID.

- **ProcessElement** is a general construct for referring every element in the process model. Each ProcessElement is connected to another through a precedence association characterized by a Flow.
- **Flow** is an association class related to the association of two process elements. It can represent both control and data flow.
- **Gateway** is a ProcessElement that represents a process split or merge according to a specific condition. It can be a ParallelGateway or an ExclusiveGateway.
- **Activity** is a ProcessElement that specifies the execution of some unit of work.

4.1.3 Complete LPML Metamodel

As described by Section 4.1.1, we have to create the semantic service and goal annotations out of rough element descriptions. In the following, we will describe the metamodel elements for the semantic annotations. Figure 7 visualizes the metamodel view covering the semantic annotations.

- **SemanticAnnotation** contains the reference to the annotation file in case of an existing ontological annotation, in case the annotation is newly created it is represented by the attribute *expression*. Any annotation is of a certain type *AnnotationType*.
- **AnnotationType** enumerates the potential annotation types. It is limited to annotations for functional classification, non-functional properties, preconditions, effects, metadata, requirements, constraints, selection criteria, and replacement conditions.

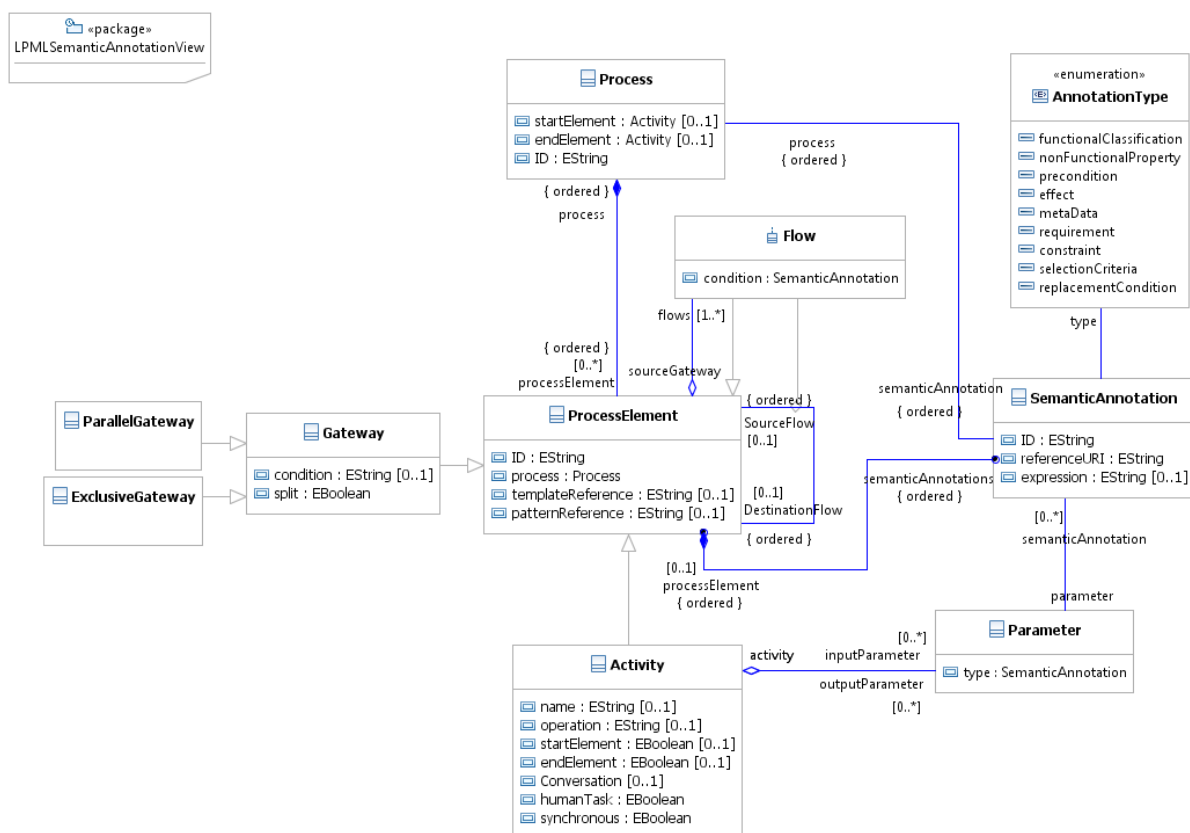


Figure 7: Semantic annotations for the LPML

Now we are going to map the semantic annotations to existing goal and service descriptions. In case of mapping the annotations to the goal descriptions we have to figure out the goal that fits best. The goal instantiation is referred to as state 3 in Figure 5. In the complete LPML metamodel (see Figure 8) the element Goal will be instantiated and reference the goal. The selection of the appropriate goal is done by T2.2 and T6.4 based on the SelectionCriteria.

Either the selected goal out of state 3 or the semantic annotation data – both including the parameter for input and output variables - is given to the discovery engine of WP5 in order to find a set of appropriate services (class Service in Figure 8). The service set is ordered in a list according to SelectionCriteria (see Figure 8). For each service in the list that is described through SAWSDL annotations, the Service class contains the reference of the service URI.

Furthermore, the LPML metamodel addresses the service replacement at runtime. We therefore have added the replacementCondition class specifying criteria for a potential service replacement as depicted as state 4 in Figure 5.

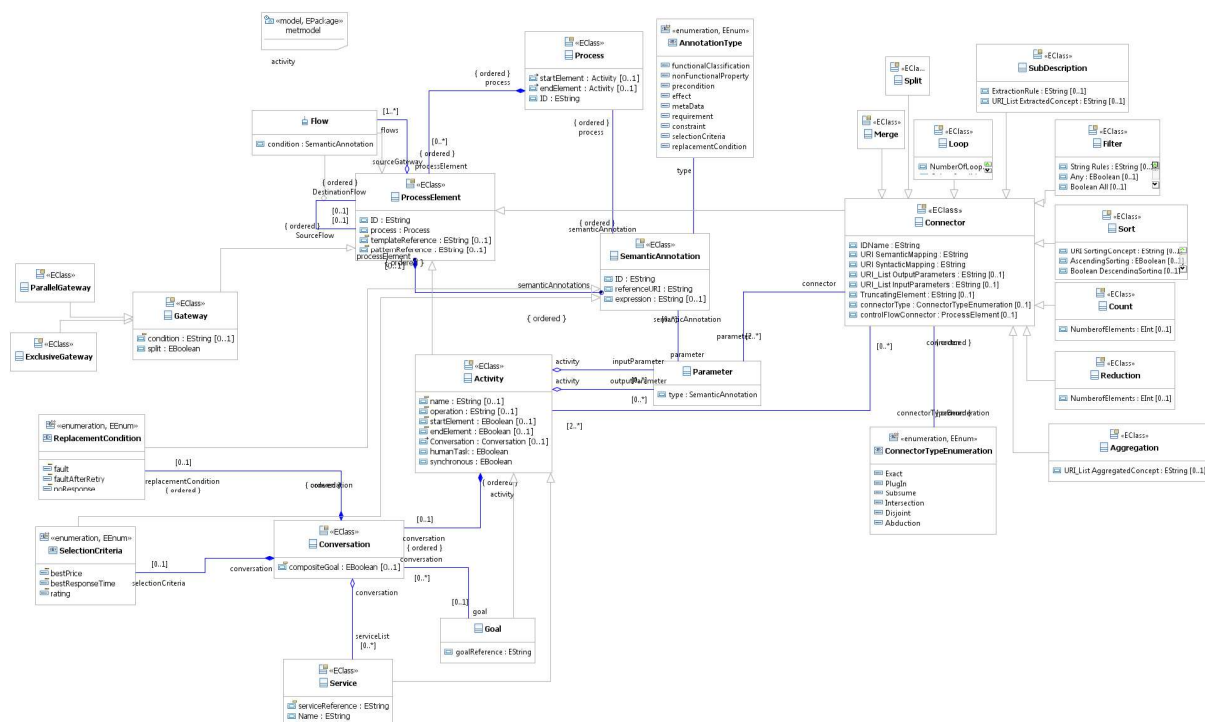


Figure 8: Complete LPML metamodel

The LPML metamodel is independent of any existing process modelling language. Thus we keep flexibility and allow for transforming the LPML into any existing language. In a later step in SOA4All we will provide a concept for the mapping and transformation of the LPML into an extended version of BPEL.

In the Annex A you can find process models in LPML representation out of the use cases.

4.1.4 LPML Metamodel Elements

We will now provide a detailed description of the elements of the LPML metamodel. The elements are grouped in several tables comprising information about the elements included, the attributes or literals, a potential reference to a context file, semantic annotations, and the rational of the inclusion. In order to keep the LPML really light we won't support event

handling as it is in other process modelling languages. We will subsume events to activities.

The LPML metamodel includes the process element serving as modelling container. It defines the process structure. Processes have a special association to exactly one start element that represents the entry point into the process and one end element representing the entity that performs the callback in case the process has terminated. The start element is thereby invoked by external callers and triggers the whole process. The process element can be encapsulated and published as a service and is described in detail in Table 2.

In addition we included the attributes `isPattern` and `isTemplate`. These two attributes define whether a process is a pattern or a template. In these two cases the process does not necessarily contain a start and an end element. We will describe the support for patterns and templates in detail in section 4.2.

Table 2 : Description of Process

Process	
Elements included	All elements can be included. A process necessarily contains one start and one end activity.
Attributes	ID isPattern isTemplate
Reference to context file	Yes
Semantic Annotation	functionalClassification precondition effect metaData

Table 3 covers the description of *ProcessElement* and most of its children. The children described here are the basic process structuring elements for the control flow. The process element is an abstraction of potential modelling elements. It contains common attributes and is part of the process. An *activity* represents a working step within a process.

A gateway splits or merges the control flow. We explicitly won't support an inclusive gateway element. This element can be replaced by a combination of an exclusive and a parallel gateway.

The *Flow* element is an association class attached to the relation of two process elements. This association describes the source and destination element. In addition it is a child of *ProcessElement*. The aggregation between *Flow* and *ProcessElement* describes the amount of incoming or outgoing flows a process element has.

Table 3: Description of ProcessElement and its children

Element	Related Elements	Attributes	Reference to context file	Semantic annotations
Process Element	Part of: Process Children: Activity Flow Gateway	ID templateReference patternReference	Yes	Not applicable

	Connector			
	Association: SemanticAnnotation			
Activity	Parent: ProcessElement Children: Goal Service Association: Conversation Connector parameter	Name Operation startElement endElement humanTask synchronous	Yes	FunctionalClassification nonFunctionalProperty precondition effect metadata requirement constraint
Flow	Parent: ProcessElement	Condition	Yes	Not applicable
Gateway	Parent: Process Element Children: ExclusiveGateway ParallelGateway	Condition split	Yes	Not applicable
Exclusive Gateway	Parent: Gateway		Yes	Not applicable
Parallel Gateway	Parent: Gateway		Yes	Not applicable

In order to separate the activity description and its instantiation we created the Conversation element that associates goals or services to an activity. This approach is similar to the separation of activities (invoke, receive etc.) and their partner link. While the Goal element references existing goals, the service class provides a list of services. The Conversation can have attached a goal and a list of potential services. As described in the design process for lightweight process modelling (see section 4.1.1) the conversation references a concrete service that is selected by analysing the semantic annotations, the referenced goal, and the SelectionCriteria. The SelectionCriteria class is of type enumeration and defines the ranking of services in the service list. In case a selected service is not available, the ReplacementCondition of type enumeration defines when to replace that service. Table 4 gives an overview of the Conversation element and attached services and goals.

Table 4: Service and goal description

Element	Related elements	Attributes/Literals	Reference to context file	Semantic annotations
Conversation	Association: Activity Goal Service ReplacementCondition SelectionCriteria	compositeGoal	Yes	Not applicable
Service	Association: Conversation Parent:	serviceReference		Not applicable

	Activity			
Goal	Association: Conversation Parent: Activity	goalReference		FunctionalClassification nonFunctionalProperty precondition effect
Selection Criteria	Parent: SemanticAnnotation Association: Conversation	bestPrice bestResponseTime rating	Yes	Not applicable
Replace- ment Condition	Parent: SemanticAnnotation Association: Conversation	Fault faultAfterRetry noResponse	Yes	Not applicable

Table 5 provides a detailed description of the elements needed in order to attach semantic annotations to the process elements. The SemanticAnnotation element is the class for all types of annotations for process elements and contains besides the attributes ID and referenceURI the attribute expression. Expression has a value in case a semantic annotation is created from scratch. In case a semantic annotation already exists the referenceURI references this annotation. The class AnnotationType enumerates the potential annotation types. The relation of inputParameter and outputParameter of an activity is specified by the Parameter element.

Table 5: Elements for semantic annotation

Element	Related elements	Attributes/Literals
Semantic Annotation	Association: Process ProcessElement AnnotationType Parameter Children: ReplacementCondition SelectionCriteria	ID referenceURI expression
Annotation Type	Association: SemanticAnnotation	functionalClassification nonFunctionalProperty precondition effect metaData requirement constraint selectionCriteria replacementCondition
Parameter	Association: SemanticAnnotation Connector Activity	Type

Table 6 addresses the data flow handling. The Connector element is responsible for the data mapping between services. Further, various specifications of the connector element are provided. All elements for the data flow handling refer to a context file.

Table 6: Elements for the data flow handling

Element	Related elements	Attributes
Connector	Parent: ProcessElement Association: Activity Parameter ConnectorTypeEnumeration Children: Merge Split Loop Filter SubDescription Sort Count Reduction Aggregation	IDName URISemanticMapping URISyntacticMapping URIListInputParameters URIListOutputParameters TruncatingElement connectorType controlFlowConnector
ConnectorTypeEnumeration	Association: Connector	Exact PlugIn Subsume Intersection Disjoint Abduction
Merge	Parent: Connector	
Split	Parent: Connector	
Loop	Parent: Connector	NumberOfLoop
Sub Description	Parent: Connector	ExtractionRule URIListExtractedConcept
Filter	Parent: Connector	Rules Any All
Sort	Parent: Connector	URISortingConcept AscendingSorting DescendingSorting
Count	Parent: Connector	NumberOfElements
Reduction	Parent: Connector	NumberOfElements
Aggregation	Parent: Connector	URIListAggregatedConcept

A sample file of an existing WP9 process can be found in Annex A.

4.2 Patterns and templates

There is wide agreement that patterns can accelerate designing process models and reduce modelling time. Patterns enable participants of a community to communicate more effectively, with greater conciseness and less ambiguity (Medicke and McDavid 2004; Buschmann, Henney et al. 2007; Tran, Coulette et al. 2007). According to D6.3.1, we thus choose to use a part of well-known workflow patterns from (van der Aalst, ter Hofstede et al. 2003a) as our *process pattern* in order to support modelling the control flow perspective. The patterns range from very simple to very complex and cover the behaviour that can be captured within most business process models.

The workflow patterns from (van der Aalst, ter Hofstede et al. 2003a) are however too fine-grained and not sufficiently enriched with information on the context and consequences to represent a reusable solution. As described in D6.3.1, therefore, we introduce *workflow templates* – here as well for the control flow perspective – that are different combinations of process patterns. The processes represented by a workflow template are sound. Certain workflow templates can be enriched by specific information in order to be applicable to different business domains.

Workflow templates and process patterns are similar to the processes as their descriptions are stored in the semantic spaces and can be referenced by a URI to the respective description. As already stated, workflow templates are sound processes. Process patterns are not sound. In order to distinguish between process descriptions, workflow templates, and process patterns, two annotations are added to their descriptions. The flag `isTemplate` is set to `true`, if a workflow template is described and to `false` otherwise. The flag `isPattern` is set to `true`, if a process pattern is described and to `false` otherwise.

Once business processes are modeled within the LPML, workflow templates and process patterns might be incorporated. As workflow templates are sound processes, it is also possible to model novel processes by simply instantiating and possibly customizing the workflow template. In those cases, when templates and patterns are reused, it is useful to identify which activities and goals of the process are part of which template or pattern. This objective is accomplished by annotating each process activity, activity goal, or composite activity goal that is part of a workflow template or process pattern with a reference to the corresponding process pattern or workflow template. If activities or goals are not part of a template or pattern, then the annotation is not used for these elements.

On the level of the LPML information model, we therefore introduce an annotating notation that may be attached to the following process elements: activity, atomic activity goal, and composite activity goal. The LPML UML model also shows the attribute that represents such an annotation of activities. The visualization of these annotations is similar to comment annotations or OCL (Object Constraint Language) constraints on attributes within UML diagrams. The simplicity of this notation gives non-expert users a quick understanding of the matter.

The process pattern and workflow template annotations contain a reference to a pattern or template, respectively, available in the semantic spaces. Since workflow templates can be considered as sound processes, which description is stored in the semantic spaces, workflow templates are also identified by a unique URI. The same applies for referencing the process patterns. Consequently, URIs of template and pattern descriptions represent the references within the annotations of activities and goals.

4.3 Goals

Process activities are traditionally concrete and bound to services or other means of

implementation at design time. As introduced in D6.3.1 and D6.4.1 we will use activity goals as unbound activities that are bound to a particular service either at design time or at runtime by SOA4ALL WP6 composition services (Composer or Executor). Goals will support users in modelling the control flow perspective.

LPML introduces a Goal element to describe those unbound activities. LPML Goal is aligned to the SOA4ALL Goal specification, based on WSMO Lite, which is described in this section.

Syntax of Goals

Here we talk about the information model and its UML representation. Obviously, the syntax is independent from a later BPEL serialization.

Semantics of Goals

The following paragraphs show one approach to define a semantics of LPML goals. First, the description of Web services is outlined. The functionality-based discovery by WP5 provides a semantics of the functional classification of WSMO-Lite service descriptions. Thus, we will also briefly describe the classification and its relationship to the Web service descriptions. Derived from these requirements, the structure of a LPML goal is investigated. Afterwards, a semantics of such goals is introduced. LPML goals must at least contain or reference the information required for a discovery as developed in WP5. For more detailed information about the functionality-based discovery we refer to D5.3.2.

WSMO-Lite, as presented in (Vitvar, Kopecky et al. 2008), does not provide any formal semantics. It also does not provide an interpretation of the classification and the classification hierarchy. That is, what does it mean if a Web service is assigned to a class of the functional classification root, and what does the subclass relationship between classes of the functional classification mean. The meaning of the WSMO-Lite elements intended for the purpose of describing the functionality of Web services becomes even more unclear, when classification is considered along with pre-conditions and effects. A formal semantics of functionalities should have a clear understanding of a class, a sub class relationship, and the classification of services.

Functionality-based Discovery A functionality of a Web service is described by the tuple (I, O, Φ, Ψ) with I the set of input variable names, O the set of output variable names, Φ the pre-condition, and Ψ the effect. Inputs and outputs are sets of locally unique names of variables, i.e., strings. Preconditions and effects are logic expressions on an abstract level. From a more concrete perspective, WSMO axioms represent those logical expressions. These axioms may include the input and output variables. Input and output variable names are subsets of the variables within the axioms of preconditions and effects. By M18, a WP5 goal is composed out of the above mentioned attributes. The first ranking prototype will not include non-functional properties. After M18, WP5 goals will be extended to non-functional properties. As a consequence, the LPML has to be extended accordingly, once the ranking algorithm specifies how preferences over non-functional properties are defined.

Each Web service is described and can be discovered by the description of the functionality. The discovery engine matches functionality descriptions of the available Web services with the desired functionality of a search query. Anticipatory goals must be able to express the desired functionality of a Web service that is bound at runtime.

As WSMO-Lite is used to describe services, the functional classification that comes with WSMO-Lite can be incorporated to the functionality description. WP5 provides a semantics of the functional classification, which is a classification hierarchy of functionality descriptions. Note, that this is a hierarchy on functionality description and not on service descriptions. The construction of the hierarchy relies on the definition of 'sub class of'-relationship between functionality classes.

A class $S(c)$ is super class of another class c , if $I_{S(c)} \subseteq I_c \wedge O_{S(c)} \subseteq O_c \wedge \Phi_c \Rightarrow \Phi_{S(c)} \wedge \Psi_c \Rightarrow \Psi_{S(c)}$ holds. Consequently, by this sub class relationship guarantees that the functionality of sub classes extend the functionality of their super class. That is, a class c may provide more input or output variable names or further pre-conditions or effects than a class $S(c)$. Then c is a sub class of $S(c)$ in such hierarchy.

Given a hierarchy as defined in D5.3.2, WSMO-Lite Web service descriptions are assigned to the most specific functionality classes that match the functionality (I, O, Φ, Ψ) of the corresponding Web services. Since a goal is bound to a service at process execution time, the goal must provide the information that is necessary to create the functionality description of the desired Web service.

A LPML goal provides the following attributes: inputs, outputs, pre-conditions, effects, a functional classification, and with the development of the ranking algorithm a notion of preferences over non-functional properties. The semantics is based on the functionality rather than just types of input and output variables. Existing approaches like (Keller, Lara et al. 2004) that also take the functionality into account do not scale for a large number of Web services. Note, the WSMO-Lite description of Web services is related to the description of goals, since the Web service functionality descriptions are matched against the desired functionality description of goals. In contrast to WSMO-Lite description of Web services, LPML goal descriptions are enhanced by the input and output sets in order to allow a distinction between input/output variables and those bound variables occurring in pre-condition and effect axioms. The definition of a semantics of goals relies on the mathematical underpinning of the service functionality description such that goals become universally comprehensible.

With respect to LPML goals, the semantics of the functional classification hierarchy can be summarized as follows: classes of the classification hierarchy are viewed as goals and the 'sub class of'-relationship is regarded as a 'sub goal of'-relationship between goals. As mentioned, the classification of services assigns the service descriptions to a particular set of functionality classes, which correspond to the goals. Thereby, goals are also hierarchically structured by the same functional classification hierarchy. The classes of the classification hierarchy can be abstracted by names given to a goal. This means, goals define functionality classes and the structure of the functional classification used by the WSMO-Lite ontology implies a subclass relationship upon the goals of services. The overall effect of a service that is assigned to a set H of goals of the functional classification is the conjunction of the effects of the individual goals in H and the effect of the service itself. That is, similar to queries described in D5.3.2, goals incorporate class names in order to hide the complexity of a functionality description of a class. Furthermore, queries or goals using the functionality classification are much more efficiently to compute. On top of the classification hierarchy, there are goals that do not depend on other goals, i.e., classes that are not sub class of another class in the functional classification. The semantics of a WSMO-Lite service description introduced above is used to provide a semantics of LPML goals. As depicted in Figure 9 both WSDL and hRESTS services can refer to the WSMO-Lite ontology.

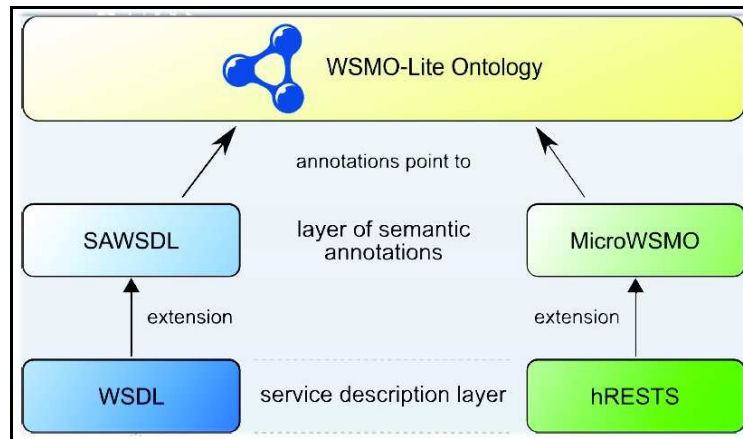


Figure 9: Reference to WSMO-Lite ontology

In the following we will present a short example defining preconditions and effects of a web service description:

- **Precondition of a Web service description:** BookOrder(o) and hasISBN(o, i) and ISBN(i) and hasCreditCardDetails(o, c) and CreditCardDetails(c) and (hasCardType(c, "VISA") or hasCardType(c, "Amex")) and hasExpiration(c, e) and AtLeast3MonthsInFuture(e)
- **Effect of a Web service description:** OrderConfirmation(oc) and hasDetails(i, d) and hasPrice(i, p) and hasOrderedGood(oc, d) and hasAmount(oc, p) and paymentBy(oc, c)

The precondition and effect description of the goal could be as follows:

- **Desired Pre-condition, i.e., a query or a goal:** JournalOrder(jo) and hasISBN(jo, i) and ISBN(i) and CreditCardPayment(jo, c) and hasCardType(c, "VISA")
- **Desired Effect, i.e., after a goal was bound to a service and the service was executed:** OrderConfirmation(oc) and hasDetails(i, d) and hasOrderedGood(oc, d) and paymentBy(oc, c)

This example shows how one possible class “BookSellingWebService” could look like. The class does not have to be equal to the actual service description. In the functional classification hierarchy of functionality classes, we could add a sub class to “BookSellingWebService” if we require more pre-conditions to hold. Let’s say, “ComicBookSellingWebService” is a sub class, if we add to the pre-condition one axiom that ensures that the ordered book is a comic book.

The desired pre-condition given in this example could be regarded as what a user might express to formulate a goal of a book selling web service. It would match the “BookSellingWebService”, but it is far from efficient. For the expression of goals, we utilize the functionality classes that already express the common functionality of book selling Web services. Thus, the user does not need to write large input/output sets, pre-condition/effect expressions. The user specifies the classes and also might provide further input/output/pre-condition/effect to refine the goal.

LPML Goal Metamodel

The LPML Goal concept is represented by the UML model in Figure 10.

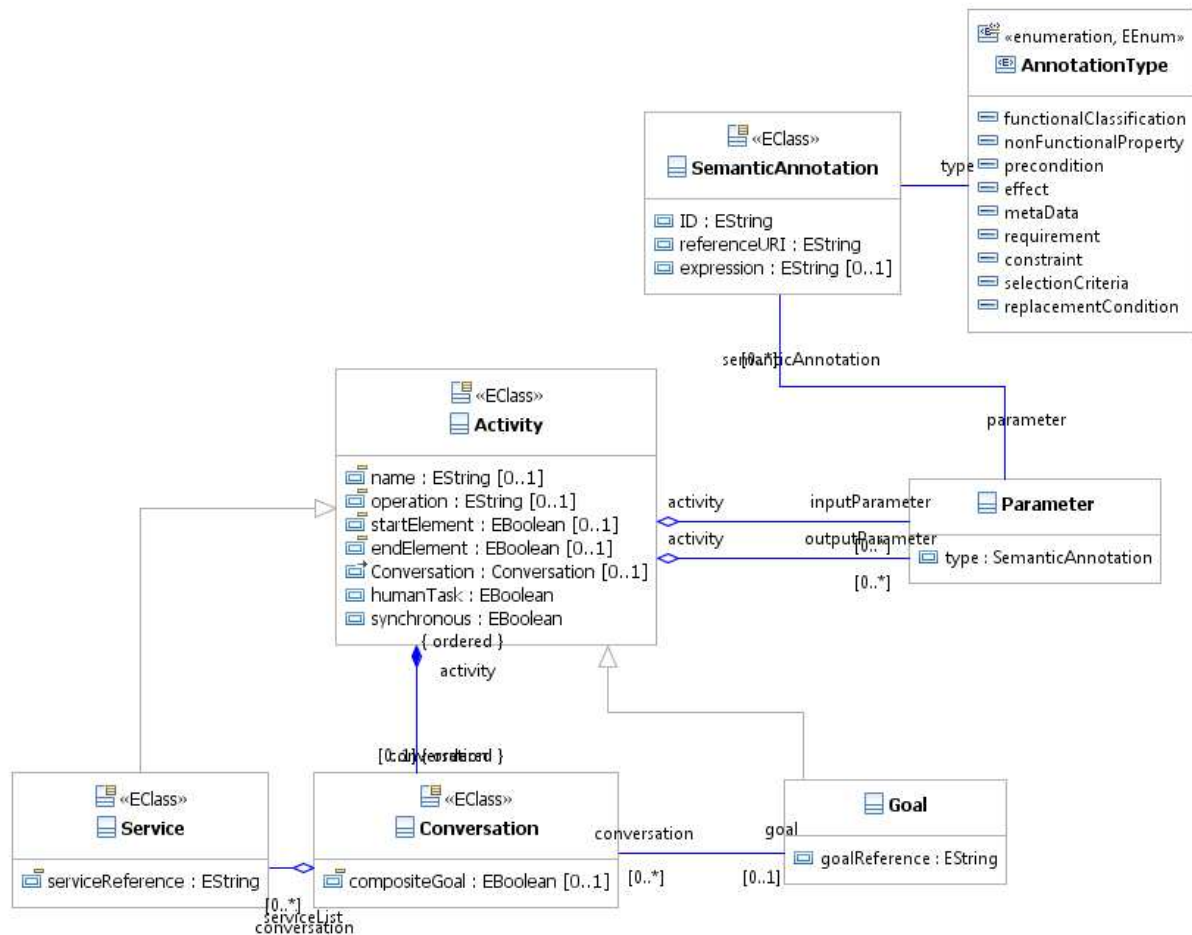


Figure 10 LPML Goal concept

A **LPML Goal** represents an unbound abstract Activity that has some optional properties of type: Functional classification, non-functional property, precondition and effect. A LPML Goal can be understood as an abstract classifier for fitting SWS. An instance of the goal class represents a concrete goal, which provides concrete instance references to some of its optional properties. They can be optionally provided by the modeller, but they could also be derived from domain specific contextual information and knowledge bases by WP6 tools, concretely: Composer and Optimizer at design time, and Executor at runtime. The goal approach is quite suitable for average users, since they can browse and inspect available goals stored within the SOA4ALL Goal registry, choose one of them and populate it with concrete values.

All LPML Goal elements are optional. However, at least one of them has to be specified. A LPML Goal describes optional requirements using a **Functional Classification** element, which points at a subclass of a WSMO-Lite FunctionalClassificationRoot. Goal preferences are described with the **Non-functional Property** element that points at a WSMOLite NonFunctionalParameter instance. Preconditions are described by the **Precondition** element, which points at a WSMO-Lite Condition instance. Postconditions are described by the Postcondition element, which points at a WSMOLite Effect instance.

Alternately, a LPML Goal can refer to a WSMO-Lite goal instance available within the SOA4ALL goal registry using the reference property. In this case, if other LPML Goal properties are specified, they complement/refine the referenced goal.

LPML Goal and Activity are annotated with optional semantic annotations. In this way we

allow a unbound number of annotations of different types (see AnnotationType enumeration in Figure 10). Therefore, any optional properties of goals, such as requirements, preferences, preconditions, effects, inputs and outputs are represented as optional annotations of any type defined by the AnnotationType: FunctionalClassification, NFProperty, Precondition, and Effect. Complementary, a Goal can reference to a Goal instance available somewhere. A goal matching mechanism is optionally used to set the requested discovery matching to the WP5 discovery service.

4.4 Data Flow Perspective

For process-aware applications various perspectives can be distinguished. While the control-flow perspective captures aspects related to control-flow dependencies between various tasks (e.g. parallelism, choice, synchronization, etc.) the data perspective deals with passing information, scoping of variables, etc. Same like defining patterns and templates for the control flow we can define patterns for the data flow. However, this is not main work of SOA4All. In the following we will present how the user is supported in modelling the data flow.

Besides purely control flow oriented constructs, the LPML aims at providing some data flow oriented constructs for supporting mashup-based service composition. We do not aim at replacing complex workflow languages though, but rather promote a data processing model. To this end, we present an (non exhaustive) list of operators enabled by the LPML. Such operators are required to model data manipulation through the LPML. By doing so the SOA4All approach differentiates from existing approaches that handle mediation by dynamically defining mediators. The latter approach would require deep knowledge in ontologies that the typical SOA4All users won't have. Afterwards we will give an insight into the SOA4All data handling from the perspective of each involved component.

In Annex B you will find the terminology of the concepts used in this section.

4.4.1 Data Manipulation and Operators

In the following we will consider only input and output based data manipulation. We will provide a mechanism to manipulate both data types and values. In addition we assume that services require and consume RDF graph respectively as input and output parameters. The current list of operators supported by the LPML for data manipulation is as following:

- **Merge (Union) Operator:** This operator takes an arbitrary number of RDF graphs as inputs, expressed in RDF/XML or N3 format, and produces an RDF graph that is composed of the merge of its inputs. The Split operator is the reverse of Merge, it splits an RDF graph into multiple identical RDF graphs.
- **Split Operator:** This operator receives an RDF graph and splits it into two identical output RDF graphs. This operator can be used when the end-user wants to perform different operations on data from the same RDF graph. The Merge operator is the reverse of Split, it merges multiple RDF graph into a single combined RDF graph.
- **Count Operator:** This operator counts the number of items in the input RDF graph, and outputs that number.
- **Filter Operator:** The Filter operator can be used to include or exclude items from a RDF graph. Therefore some rules can be created on top of the language to compare RDF graphs to values the end-user specifies. For example, you may create a rule that says "permit items where the 'item.Tag' is of 'ontology#concept' type. We could also model a rule that says "omit any items where the 'item.Tag.published' is before 'Date#date'".

- **Reduction Operator:** This operator returns a specified number of items from the top of the input RDF graph. This operator limits the number of items in the output RDF graph. We could also imagine selecting a random item from a RDF graph.
- **Sort Operator:** This operator (functional) sorts an input RDF graph by any item element, such as title or description. Items can be sorted in either ascending or descending order.
- **Loop Operator:** The Loop operator introduces the idea of sub-data processing. Any other operators could be inserted inside the Loop operator. An input RDF graph is provided to the Loop operator, the sub-data processing is ran once for each item in the input RDF graph.
- **Sub-Description Operator:** In case the data required by the end-user is deep in the description, this operator can be used to extract and select some sub-descriptions from the input. The sub-description operator is the reverse of Aggregation; it extracts description of RDF graph into a more general RDF graph.
- **Aggregation Operator:** In case specific description on data required by the end-user is coming from different services, this operator can be used to aggregate the different descriptions of input RDF graphs in one description as an output RDF graph. The Aggregation operator is the reverse of sub-description; it aggregates descriptions of RDF graphs into a more specific RDF graph.

Each connection (operator, service) has the following optional attribute regarding the data passing: Rounding-up, rounding-down, truncating (these options are shown and can be selected trough the process editor and then modelled by the LPML). This option is required in case the data provided and consumed are not of the same data type, so a process of rounding-up, rounding-down, truncating could be required in some cases. Once the composition modelling through the LPML, the truncating process of data is first achieved by T6.4 (at semantic level) and then by T6.5 (at syntactic level).

4.4.2 External View of Data Flow Manipulation

All described operators are considered as semantic and syntactic mapping elements in LPML. The data they manipulate are propagated through the end-user (process editor in T2.6), the composer (T6.4), the optimizer (T6.4), and the executor (T6.4). In the LPML approach data mediation is handled by these four components.

From an End-User Perspective

From an end-user perspective, the aforementioned list of operators will be available through a toolbox provided by T2.6. Besides simply drawing connections from outputs of services to inputs of other service, the end-user has the possibility to specify the “kind” of connections she would have between her services, actually like other Mashup editors (i.e. Yahoo! Pipes, Deri Pipes, SIMILE Banach), stating that the output will be send to this input. In that way the end-user could visualise, easily drag and drop the appropriate operators and then link them to the services she wants to appear in the final composition. All these connections are stored through the LPML description of the composition; they describe how services are connected.

Alternatively, in case the user requires more specific or advanced data manipulation within her process, T2.6 provides some (heavy-weight) functionality for dragging/dropping the appropriate external (built-in) service, achieving the latter required specific data manipulation.

From a Composer and Optimizer Perspective (T6.4)

From a T6.4 perspective, the data manipulation is ensured (only) at semantic level i.e., the composer (as a *back-end mechanism*) simply checks the semantic consistency of data connections (or semantic compatibility i.e., a mapping between those two data types is “semantically” possible) upon domain specific data type ontology. This is achieved by

exploiting some light semantic based reasoning through the semantic link operator of the composer and optimizer in T6.4 (through WP3 reasoning). Semantic mapping is easier for the user, but it leaves the syntactic mapping issue open (to be resolved by the execution engine). In case of consistency the information is sent out to the execution engine (T6.5) with possible pre-checking by the end-user.

Otherwise the connections are labelled to “semantic-*unsuitable*” and returned to the end-user through the process editor. In the latter case the (semantic) data manipulation is only *semi-automatically supported*.

The semantic compatibility is helpful for computing the syntactic mapping (achieved by the execution engine in T6.5). The support is done by means of some scores between data.

From the Execution Engine Perspective (T6.5)

Once the data manipulation is ensured at semantic level, the syntactic mapping takes place through T6.5 by means of some Assign/Copy elements + XPath/XQuery processes. This is achieved by passing the XML data of one service to the other service (more specially when there is heterogeneity of XML encoding). In other words the semantic mapping is propagated to the executor engine (as a *back-end mechanism*) which is in charge of finalising the syntactic mapping (automatically at transformation time during transformation from LPML to e.g., BPEL, or at runtime) and so creating the proper executable composition.

In case the syntactic mapping of T6.5 fails, the fault connections are labelled to “syntactic-*unsuitable*” and returned to the end-user through the process editor (based on a tool that allows user-based syntactic mapping). During execution the execution engine should then call the editor again in order to transform data.

In this case the (syntactic) data manipulation is only *semi-automatically supported*.

Key Summary

The T2.6 Process Editor relies on T6.4 (semantic mapping) and T6.5 (syntactic mapping) to do data mapping when possible, and when not, it should provide a tool for manual mapping, but relying on T6.5 for the syntactic serialization into BPEL.

4.5 Summary and Remarks

In this section we provided a deep insight into the LPML elements and the mechanism they support. These mechanisms implement the modelling principles defined in section 3. We first introduced the design process of how the abstract, graphical process models are systematically enhanced in order to be eventually executed. Afterwards we presented the metamodel from the modellers perspective as well as the complete LPML metamodel. We then described in more detail how the LPML implements the new design principles for process modelling, namely support for patterns and templates, for goals and for data flow. While patterns, templates and goals are mainly defined for supporting users in modelling the control flow, the data flow perspective highlights to facilitate the passing of data.

5. LPML API: Requirements, Design and Implementation

Previous sections describe the LPML metamodel, that is, the set of elements, properties, relationships, assumptions, constraints, etc. that constitute the LPML. However, on the programmatic perspective, LPML requires to be managed by an API that abstracts and hides the complexities of the LPML elements and their concrete serialization formats to the programmer. This section describes the LPML API, which provides programmatic process modeling and serialization support, either for storage as RDF and transformation into SOA4ALL extended BPEL 2.0⁴ or other executable languages. In the former case, a RDF schema for LPML and its mapping to the LPML elements is provided within the API. In the latter case, we describe the BPEL extensions and the mapping between LPML elements and SOA4ALL extended BPEL 2.0 elements.

While this section describes LPML API, its implementation will be released as a separate deliverable (as part of D6.5.2 due on M24)

5.1 LMPL API Requirements

SOA4ALL process models are described using the LPML metamodel. From a pure technical perspective, LPML models require to be programmatically managed and formally serialised in order to satisfy some main requirements:

- Support a programmatic creation, modification and usage of process models, accessing and inspecting processes and their elements within any SOA4ALL component, just like accessing any on-memory model of POJO/Java Beans objects, in a very similar way the XML DOM does with XML documents. LPML API should also include helper classes providing common features required to manage LPML models: accessing process model elements, adding new elements, replacing elements, etc.
- Support for storage and retrieval into/from the SOA4ALL template repository included in the main SOA4ALL data repositories: Semantic Space nodes distributed along the SOA4ALL infrastructure. This requirement allows further reuse of process models, fragments and templates, one of requirements for the Lightweight Process Modelling approach.
- Support for interoperability among T2.6 SOA4ALL Studio Process Editor (see T2.6) and WP6 components, and internally among the WP6 components: composer, optimizer, process template generator (T6.4) and executor (T6.5). Those components interchange mainly process models (in LPML format). Due to the WS RCP oriented integration approach encouraged by the SOA4ALL DSB, these SOA4ALL components have to exchange LPML based process model objects. Nonetheless, this is discouraged since LPML contains complex POJO beans, which would require building complex SOAP bean serializers and deserializers. Therefore is better SOA4ALL tools exchange RDF documents (serialized as XML documents) which are default supported by common SOAP engines.
- LPML is intended to describe process models, abstract process models (templates), and process fragments. LPML serialization should provide suitable features for inspecting process blocks or fragments, to retrieve process models as a whole or inquire concrete elements, leveraging on existing Web data representation technologies, such as XML/XSD or RDF/S, accompanied with powerful querying support, that is, XQuery/XPath, SPARQL respectively.

⁴ We refer to the extended BPEL language used by D6.5.1 Execution Environment

- Support for the validation (well-formedness) of imported (loaded) LPML process models.
- Support for LPML translation into BPM executable languages

WP6 components (i.e. Composer, Optimizer, Process Templates Generator, Executor) interact with each other during the Design Time and Deployment Time modelling phases, mostly exchanging LPML process models through the DSB. Although the RDF/S serialization of LPML process models is suitable for the needs of Composer, Optimizer and Process Templates Generator, it is not suitable for the Executor. The reason is that LPML, even if executable (since it contains all required information for the execution time), it is not supported by the Executor engine, since it uses an Apache ODE process execution engine, which only understands BPEL 2.0. Therefore, LPML API should include a LPML to BPEL translator (as part of D6.5.2 deliverable due on M24) which creates SOA4ALL extended BPEL 2.0 executable process models from LPML process models. LPML is not serialized directly into BPEL 2.0 since:

- LPML aims to be a lightweight version compare to most WS-based work-flow languages
- LPML is agnostic of the underlying WS-based work-flow technology
- LPML has not a one-to-one mapping with BPEL 2.0 (even with required SOA4ALL extensions)

but LPML provides a mapping into BPEL 2.0 (plus some required SOA4ALL extensions) and it could provide a similar mapping and serialization into other workflow languages (i.e. YAWL)

The SOA4ALL extensions to BPEL 2.0 and the LPML to BPEL translation techniques and mappings are described in next sub-sections.

5.2 LPML API Specification

In order to make the LPML available for the other work packages and tasks, a library is created by task 6.3 that will provide interfaces as described below.

This library will be provided as a Jar file to other parts of SOA4All. It will be used as a base whenever LPML code needs to be exchanged. The library supports two types of serialization as described above:

- Extended BPEL 2.0 which will be usable by the execution engine but may also be used by 3rd party tools as it is backwards compliant.
- RDF which is the base for storing LPML models in the semantic spaces.

As such, the library may also be used to convert between those two serializations by simply loading an LPML model and serializing it again.

The LPML library focuses on the model itself and its loading and saving to different formats. Other work packages will then use this library to do specific operations with it such as displaying it to the user (2.6) or optimizing the model (6.4) or executing it (6.5). However, they are not part of the LPML library itself but are rather using it as a base library for their tasks. The following picture shows the usage of the LPML API:

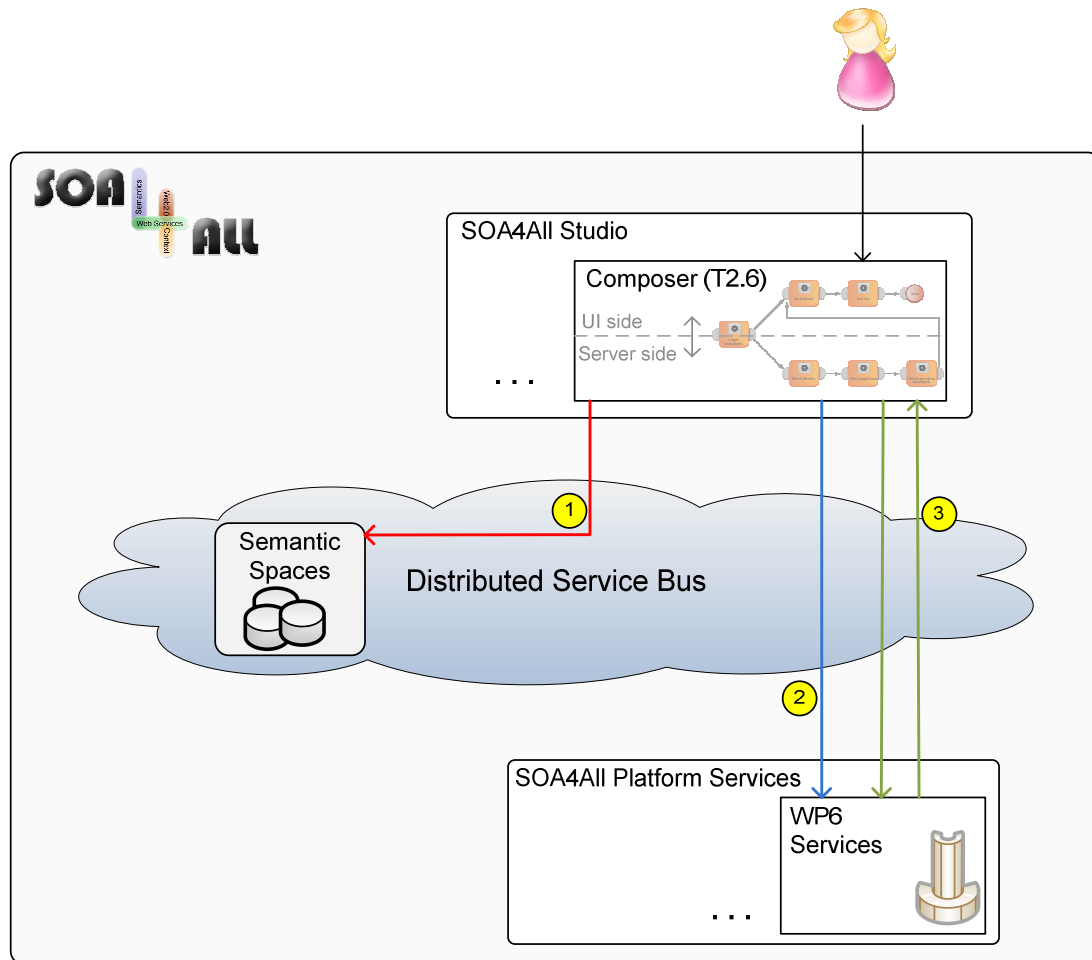


Figure 11: Example for LPML usage in the SOA4All composer (Task 2.6)

The figure shows three usages of the LPML for the communication and execution of the T2.6 composer which allows users to create processes graphically.

1. The first example shows the saving of a model in the Semantic Spaces. The 2.6 composer will use the save method of the library for performing this which will serialize the model data into RDF and send it to the Semantic Spaces using the Distributed Service Bus.
2. The second example shows the execution of a process. The 2.6 composer will use the LPML library to serialize all data into extended BPEL 2.0 or RDF and send it to the WP6 execution engine via the Distributed Service Bus. The WP6 execution engine will use the LPML library to deserialize the model back to a POJO representation.
3. The third example shows a two-way exchange between the 2.6 editor and the WP6 Template generator. The 2.6 composer will send a request to the 2.6 service via the Distributed Service Bus. The template generator will create a template and serialize it to RDF or extended BPEL 2.0 which will afterwards be sent back to the 2.6 composer. This will use the LPML library to convert the data into POJO instances and will then load it into the UI part of the 2.6 editor in order to display it to the user.

LPML API manages LPML models as POJO objects and provides support for a

programmatic inspection and access to LPML model objects. The API heavily relies on the LPML metamodel which has been described in earlier sections of this document. As such, the API uses the metamodel internally and allows users to create instances of the different parts. For example, users can create an Instance of the `Activity` class and add it to the metamodel in order to add an `Activity` to a `SequenceFlow` which is in itself included in a process.

In addition to this, the API implementation will contain two additional interfaces for any implementations that perform loading and saving activities. The API will contain two implementations out of the box for BPEL 2.0 and RDF:

```
public interface LPMLExporter
{
    public String export(Process process);
}

public interface LPMLImporter
{
    public Process import(String lpmldata);
}
```

Example for an implementation:

```
public class RDFHandler implements LPMLExporter, LPMLImporter
{
    @Override
    public Process import(String lpmldata)
    {
        . . .
    }

    @Override
    public String export(Process process)
    {
        . . .
    }
}
```

The LPML API allows users

- To **create** a new LPML model

By creating an instance of a new process without any data inside. It acts as a starting point for new models. The resulting POJO can be used to directly or indirectly add new elements to the process such as Flows, Conversations, Activities or Goals.

Example 1: Creating an empty process

```
Process p=new MetamodelFactoryImpl().createProcess();
```

Example 2: Creating a simple process:

```
MetamodelFactory factory = new MetamodelFactoryImpl();
```

```
//create a new process:
```

```
Process p=factory.createProcess();
```

```
//create services (could also be any other subclass of
ProcessElement):
```

```

Service s1=factory.createService();
s1.setServiceReference("http://tieglobal.com/SOA4All/product.wsdl");
s1.setName("Check Purchase Items");

Service s2=factory.createService();
s2.setServiceReference("http://tieglobal.com/SOA4All/purchase.wsdl");
s2.setName("Check Purchase Items");

Service s3=factory.createService();
s3.setServiceReference("http://tieglobal.com/SOA4All/check.wsdl");
s3.setName("Check Purchase Items");

//let's create a goal as well:
Goal g=factory.createGoal();
g.setGoalReference("http://tieglobal.com/SOA4All/productservice.wsmo");
g.setName("OrderProductGoal");

//now we have to add them all to the process.
p.getProcessElements().add(s1);
p.getProcessElements().add(s2);
p.getProcessElements().add(s3);
p.getProcessElements().add(g);

```

- To **load** an existing LPML model

This feature will allow users to import an existing model from a string. Formats that will be implemented in the project are extended BPEL 2.0 and RDF.

Example:

```

LPMLImporter importer=new RDFHandler();
Process process=importer.import (lpmldata);

```

- To **save** an LPML model

Similar to the import facility, this feature will allow users to export a model into either extended BPEL 2.0 or RDF.

Example:

```

LPMLExporter exporter=new RDFHandler();
String rdfdata=exporter.export (process);

```

- To **convert** an LPML model

As the LPML implementation supports different formats, this method can be used to convert between them. This will allow users to do a bidirectional conversion between extended BPEL 2.0 and LPML RDF.

Example:

```

LPMLImporter importer=new RDFHandler();
Process process= importer.import(lpmldata);

LPMLExporter exporter=new BPELHandler();
String bpeldata= exporter.export(process);

```


- To **validate** the syntax of an LPML model (RDF and extended BPEL 2.0)

This method may be used to validate a string, which contains either extended BPEL 2.0 information or RDF information. Please note that this validation will only check the syntax and not the semantic correctness of a model.

Example:

```
try
{
    LPMLImporter importer=new RDFHandler();
    Process process = importer.import(lpmldata);
    //file is valid
    //...
}
catch(ValidationException ve)
{
    //file is invalid
    //...
}
```

The library allows developers to modify, add and delete parts of the model by using the properties and methods of the API which are in sync with the LPMML metamodel definition.

The LPML API is designed to be widely extendible in terms of the serialization types. As such it makes use of interfaces which may be implemented by different serialization types. However, within the SOA4ALL project, only RDF and extended BPEL 2.0 will be implemented. Nevertheless, new serialization types may be added at a later stage when providing new versions of the LPML API. However, backwards compatibility will be provided in those cases.

As shown in the metamodel, the central point of an LPML instance is a `Process` object. A process object is interlinked with other elements directly or indirectly such as `Flows`, `Activities` and `Goals`.

Please note that the operations and properties that are provided by the LPML POJO classes are identical to the LPML metamodel and can therefore be seen in Figure 8.

Optionally, the LPML core API can be extended by other SOA4ALL components that provide additional helper classes specialized on concrete features. For instance, T6.4 components such as `Composer` and `Optimizer` will add additional interfaces as add-ons to the LPML core API. This is depicted in Figure 12.

T6.4 `Composer` request additional operations in the LPML API to support semi-automatic process composition and adaptation. These operations consume and produce some objects that not necessarily are part of LPML but are required by the semi-automatic composition parametric techniques. `Composer` uses ***replaceCompositeGoal*** to expand a process activity described as a composite goal with a matching process template. Similarly, ***replaceGoal*** replace an abstract activity described by a goal with a concrete activity bound to a SWS.***Map*** operations create dataflow connectors between parameters of two activities.

Remaining operations support semi-automated design-time parametric composition and adaptation. `Composer` requires parameterized process models where parameters refined the domain specific contextualized behaviour of the business process. Specialized parameters are requirements and constraints, as they were described in section 3.2. Requirements and constraints can be specified as assignment sets, that is, pairs (parameter, value) or as boolean evaluated logical expressions.

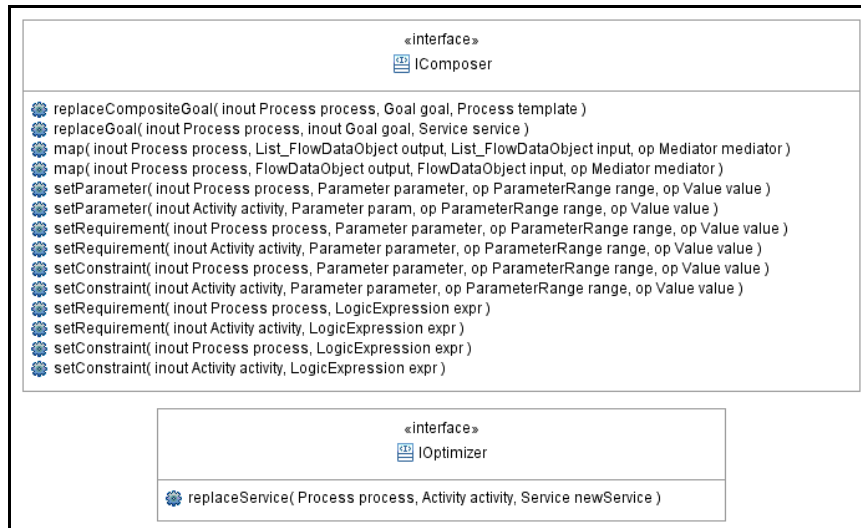


Figure 12: Additional helper classes for the LPML API

5.2.1 Using EMF as a Basis for the LPML API

We have decided to leverage the Eclipse Modeling Framework in designing and implementing the LPML, its API and the BPEL generation logic. This section gives the reasoning behind this decision by giving a very brief overview of EMF and presenting its advantages in the context of LPML.

EMF Overview:

“The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model.”⁵

EMF contains different parts out of which we use the main EMF core, which contains a metamodel (called Ecore) for describing models with an expressivity similar to UML class diagrams. The core part of EMF also provides code-generation capabilities that significantly facilitate the creation of APIs for models built with EMF and that supports round-trip engineering needed in model evolution. It ensures that the UML model of LPML is kept in sync with its Ecore counterpart, which in turn keeps the Java code for LPML elements updated. If any of these are changed during LPML evolution, the other are kept in sync.

In addition to Ecore and code generation support, EMF provides runtime support for model management including change notification (listeners can be notified when the model instance has been notified), persistence support (easy storing/loading of XML-based serialized models) and undo/redo.

All of the above-mentioned features are useful for the LPML API, as presented in the following paragraphs.

EMF Advantages to LPML

- API generation: The LPML has been designed in UML, which facilitates shared understanding and collaborative design of the language elements. Using EMF’s generative capabilities, the LPML Ecore representation was extracted out of UML and in addition, Java classes corresponding to each of the LPML elements were automatically created. This approach allows for future changes due to possible language evolution to be easily

⁵ Source: Eclipse Modeling Framework Project website: <http://www.eclipse.org/modeling/emf/>

incorporated in the code with minimal potential for errors. It also saved significant amounts of time by avoiding manual coding all the classes and their relationships in Java. The API simply needs to use these classes in order to operate with LPML elements in a natural way (e.g. using the EMF factory to generate instances of types Activity or Process).

- Reusability of SOA metamodel-based transformation approach: As detailed in section below, the EMF foundation for LPML enables its integration with the metamodel-based transformation approach for SOA that is provided by the STP-Intermediate Model project. We are extending the STP-IM to account for LPML as a source language for transformations, and for extended BPEL as a target language for transformations.

- Serialization: EMF provides a serialization mechanism that is very straightforward to use (it amounts to a couple of method calls for creating a file). It generates XML files that contain elements corresponding to the Ecore instance. In LPML's case we have elements such as <process> or <activity> that directly map to the language elements. Of course, in SOA4All our main target for serialization is RDF which is also going to be used for LPML serialization, but EMF serialization provides a simple alternative that might be offered as an export facility.

- Change notification: EMF offers a notification mechanism that can be used to signal model instance changes (e.g. a new Activity has been added in the editor). This could be leveraged by the Process Editor or future Process Editor plugins to add behaviour in a clean and extensible way (e.g. for an Activity of a certain type, open up a wizard to guide the user in specifying different parameters).

- Undo/Redo and Transactions: for Studio developers, EMF provides a mechanism for implementing consistent Undo/Redo operations that keep the model instance in a coherent state. It also provides means of executing concurrent requests while preserving model integrity, through its transactional capabilities (the default EMF-generated classes are not thread-safe). This could be useful in collaborative design of a process in the Process Editor for instance.

5.2.2 Serialization in RDF/S

This section describes the RDF schema used to serialize the LPML models into RDF and the mapping between the LPML information model and the RDF Schema when there is not a direct one-to-one mapping.

LPML models will be serialized as RDF, as requested by the storage requirement. Since LPML models are implemented programmatically as POJO objects we can use some of the available POJO serialization into RDF frameworks, such as JenaBean⁶.

Jenabean is a RDF/OWL persistence framework for Java Beans. Bindings between POJO objects and RDF schema are managed using the Java 5 annotation mechanism. That approach imposes minimal changes in the POJO model, which is quite convenient and compatible with the EMF infrastructure of the LPML API. Jenabean uses the Java Bean conventions to inspect POJO properties and derive their mapping into the RDF schema, based on the supplied POJO annotation. For instance, in the following Process class of LPML API, JenaBean annotations are shown in next code snippet:

```
@Namespace("http://eu.SOA4All.wp6.lightweighbpml#")
public class Process implements Cloneable {

    private Collection<FlowObject> elements;
    ...
}
```

⁶ <http://code.google.com/p/jenabean/>

```

    @RdfProperty("http://eu.SOA4All.wp6.lightweighbpml#hasElement")
    public Collection<FlowObject> getElements() {
        return elements;
    }

    ...
}

```

@Namespace("http://eu.SOA4All.wp6.lightweighbpml#") describes the base namespace of the RDF Schema for LPML.

@RdfProperty("http://eu.SOA4All.wp6.lightweighbpml#hasElement") describes the URL of the RDF property to get the process elements.

RDFS Classes are represented by JenaBean as @Namespace#<ClassName> and POJO getters by the value of the @RdfProperty annotation.

Using this minimal annotation set, JenaBean can create a RDF Schema, serialize and deserialize LPML POJOs into/from RDF sources, using JenaBean Bean2RDF and RDFSMapper objects. Code snippets of LPML API are shown below.

```

public void mapToRDFS(Process process, String filePath) {
    Model m = createModel();
    Bean2RDF writer = new Bean2RDF(m);
    writer.saveDeep(process);
    System.out.println(" - model saved");
    try {
        OutputStream fileOutputStream = new FileOutputStream(filePath);
        m.write(fileOutputStream, "RDF/XML-ABBREV");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public Process mapFromRDFS(String filePath, URI processURI) {
    OntModel m = ModelFactory.createOntologyModel();
    try {
        InputStream inputStream =
            RDFSMapper.class.getResourceAsStream(filePath);
        m.read(inputStream, "");
    } catch (Throwable e) {
        e.printStackTrace();
    }

    RDF2Bean reader = new RDF2Bean(m);
    Process process = reader.loadDeep(Process.class, processURI);
    return process;
}

```

5.2.3 Transformation into BPEL

This section describes the SOA4ALL Extended BPEL 2.0 language (that is the concrete SOA4ALL extensions) into which LPML process models are transformed for execution purposes. Firstly this section describes the BPEL 2.0 extension mechanism, then the SOA4ALL LPML extensions to BPEL 2.0, and finally the technical approach to the LPML to BPEL transformation.

For making the LPML models suitable for execution, we have considered XPD (Coalition 2005) or WSBPEL (Oasis 2006) as a language representation that is supported by process execution engines. We decided to use an extended version of WS-BPEL 2.0 for execution, the SOA4All Extended BPEL 2.0. This decision takes into account the SOA4All execution engine that will use that BPEL dialect.

5.2.3.1 Description of BPEL 2.0 extension mechanisms

First of all, BPEL supports extensibility by allowing namespace-qualified attributes to appear on any BPEL element and by allowing elements from other namespaces to appear within BPEL defined elements (see D6.5.1, sect. 5.3).

In addition, BPEL provides two explicit extension constructs: `<extensionAssignOperation>` and `<extensionActivity>`.

- The `<extensionAssignOperation>` construct can be used to extend the standard `<assign>` activity. This way it is possible to include extensible data manipulation operations defined as extension elements under namespaces different from the BPEL namespace. For further details, see D6.5.1, sect. 8.4.
- The `<extensionActivity>` construct can be used to include in a BPEL process definition new activities that are not defined by the specification. The contents of an `<extensionActivity>` element is a single element, qualified with a namespace different from BPEL namespace, that make available BPEL's `standard-attributes` and `standard-elements`. An `<extensionActivity>` may be a structured activity, i.e., it may contain other activities. For further details, see D6.5.1, sect. 10.9.

Extensions are allowed in BPEL constructs used in WSDL definitions.

Apache ODE (≥ 2.0) supports the extensibility mechanisms provided by BPEL 2.0. In particular, it provides a plug-in architecture that allows for registering third-party extensions (D6.5.1).

5.2.3.2 Description of LPML BPEL 2.0 extensions

In order to support the execution of processes expressed in LPML several steps are needed as described in section 4.1. The last step consists in translating the result of the process optimizer (T6.4) in an extended version of BPEL to be executed by the SOA4All Execution Engine (EE) developed in T6.5.

The EE, as described in T6.5, is composed of 2 main components, the Lightweight Process Executor and the Lightweight Process Deployer. The Process Deployer is in charge of transforming the output of the Composition Optimizer and some of the information coming from the process model developed using the Process Editor in a process ready to be invoked and executed, exposed as a service. This executable process is a process described in BPEL 2.0 language plus some extension made using the extension mechanisms provided by the BPEL 2.0 specification.

In order to allow for the dynamic replacement of services inside a process in reaction to contextual situations we added some extensions to BPEL 2.0. In this section we describe the extensions.

5.2.3.2.1 Namespace

The namespace used for SOA4All BPEL extensions is: <http://www.SOA4All.eu/serviceConstruction/LPML/executable>.

This namespace is referenced in the following of the document with the prefix **b4all**.

5.2.3.2.2 *Extension Activities*

In SOA4All we define 1 extension activity: **b4all:adaptiveInvoke**. A b4all:adaptiveInvoke activity is created for each activity in the process model that allows for service substitution. The b4all:adaptiveInvoke has the following attributes and elements:

- b4all:humanTask: this attribute is a boolean that, if set to true means that the task is to be executed by an human being through the Human Tasks Server. The default value is false;
- b4all:replacementCondition: this attribute is a literal that represents an element in a taxonomy that defines the set of pre-defined replacement conditions. The replacement conditions are the situations in which the engine will try to substitute a service with another one. If not specified the default replacementCondition is "fault" The replacement conditions allowed by the engine are:
 - o fault;
 - o faultAfterRetry;
 - o noResponse.
- b4all:selectionCriteria: this attribute is a literal that represents an element in a taxonomy that defines the set of pre-defined selection criteria. The selection criteria is the criteria applied by the engine for selecting a substitute service from a list of alternatives. The default criteria is "rating". The selection criteria literals allowed by the engine are:
 - o price;
 - o responseTime;
 - o rating
- b4all:alternativeServiceList: this element lists the services that can be used as alternatives in the service substitution. b4all:alternativeServiceList is a list of sub-elements: b4all:alternativeService;
- b4all:alternativeService: this element is an URI that points to a SAWSDL specification.

5.2.3.2.3 *Extension Assign Operation*

In SOA4All we define 1 extension assign operation: **b4all:connector**.

A b4all:connector element is created for each instance of the Connector class contained in the process model. The b4all:adaptiveInvoke has the following attributes and elements:

- b4all:semanticMapping: this element lists a set of annotations expressed as referenceURI and/or expressions.
- b4all:synctactingMapping: this element lists a set of annotations expressed as referenceURI and/or expressions.

5.2.3.2.4 *Extended BPEL Syntax*

Syntax for b4all:adaptiveInvoke

```

<bpel:extensionActivity>
  <b4all:adaptiveInvoke
    partnerLink="NCName"
    portType="QName"?
    operation="NCName"
    inputVariable="BPELVariableName"?
    outputVariable="BPELVariableName"?
    standard-attributes
  >
  standard-elements
  <replacementCondition value="NCName">
    <b4all:selectionCriteria>
  </replacementCondition>*
  <b4all:alternativeServiceList>?
    <b4all:alternativeService>
  </b4all:alternativeServiceList>

```

```

</bpel:adaptiveInvoke >

```

Syntax for alternativeService

```

<b4all:alternativeService
serviceDescription="anyURI"/>

```

Syntax for selectionCriteria

```

<b4all:selectionCriteria
  value="NCName"/>

```

Syntax for b4all:connector:

```

<bpel:extensionAssignOperation standard attributes>
  <b4all:connector>?
    <b4all:annotation type="QName" reference="anyURI" expression="NCName">
  </b4all:connector>
  standard elements
</bpel:extensionAssignOperation>

```

5.2.3.3 LPML to BPEL transformation

As previously mentioned, SOA4ALL processes defined in LPML will be exported in BPEL 2.0 in order to leverage support from existing process-engines.

To achieve this goal, we have chosen to use the Eclipse STP-IM (SOA Tools Platform Intermediate Model) project, which entail a transformation chain as depicted in Figure 13:

The following paragraphs will give a brief overview of each of the main elements of the

transformation chain: STP-IM, LPML2IM and IM2BPEL. Note that this section does not aim at fully describing the transformation process; rather it just presents the overall strategy for doing so. A complete description of this process is going to be provided in D6.5.2 at M24.

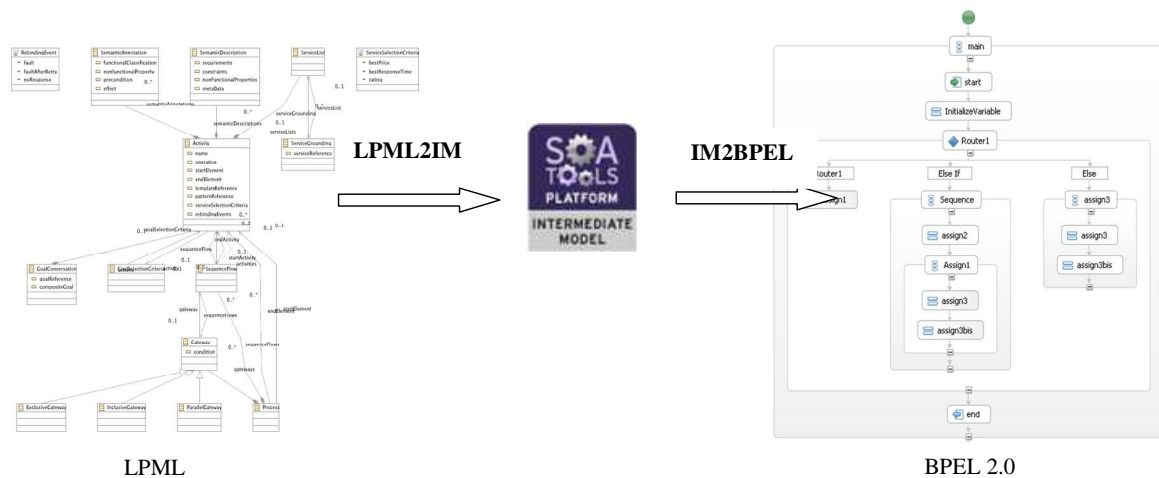


Figure 13: LPML to BPEL transformation chain

Eclipse STP-IM

INRIA is leading the Eclipse sub-project called STP-IM (SOA Tools Platform Project Intermediate-Model). This component is meant as a "bridge" between editors and its elements have the role of conceptual transport between different development spaces. It does not aim to offer a complete conceptual reasoning platform for SOA, its purpose is rather to capture as much common SOA design information from different perspectives as possible. We are not describe the entire STP-IM metamodel in detail in this section (for more details, see http://wiki.eclipse.org/STP_Intermediate_Metamodel). However it is important to mention that STP-IM addresses the architecture-oriented standards in SOA such as SCA and JBI as well as the workflow and process definition standards. This latter part is directly relevant in the context of the LPML transformation work as it captures important information about processes.

Figure 14 provides a partial and simplified representation of the STP-IM metamodel showing some of the main elements.

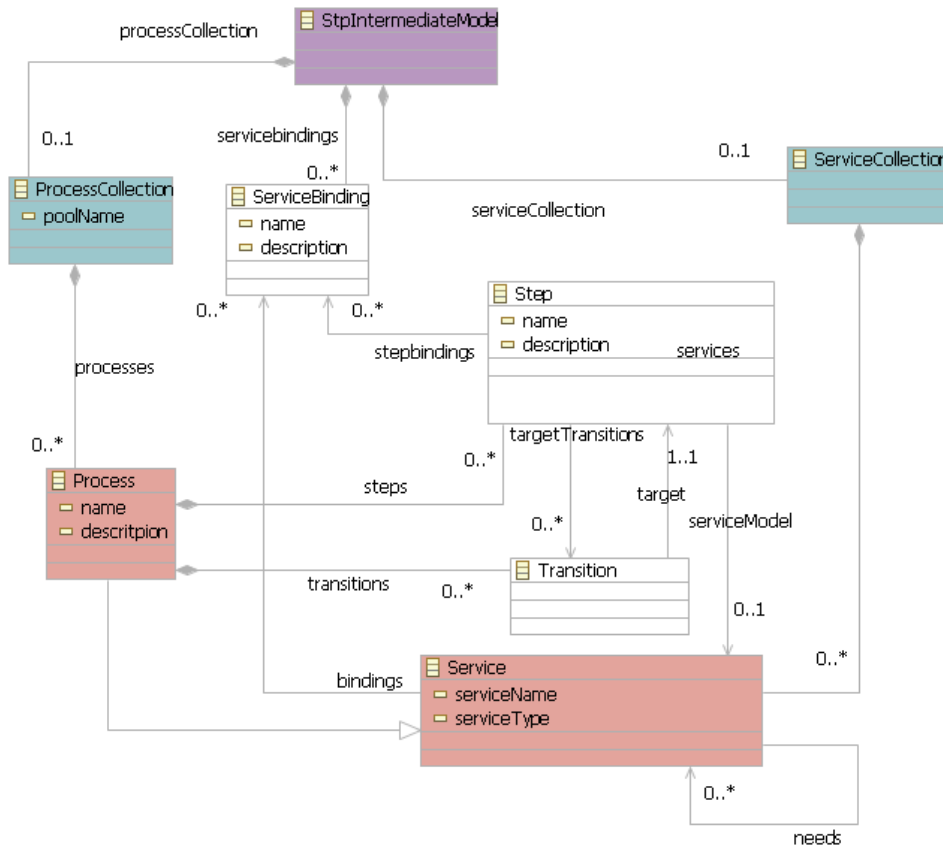


Figure 14: Simplified representation of the STP-IM Metamodel

The transformation from LPML to BPEL 2.0 will follow a generative approach relying on the IM metamodel, and consisting of the two main steps illustrated in Figure 14 and detailed below.

LPML to IM

When transforming an LPML model into an intermediate model, a plugin will parse the LPML model by iterating over all the elements of the business process and it will transform each of them into the appropriate IntermediateModel equivalent so that we get a generic representation of the LPML process expressed in the IM instance. In STP-IM, processes are defined as having steps, while each Step can be associated with a service to represent that its functionality is fulfilled by that particular service. Process steps are also associated with bindings because they need to use a specific binding of a service when executing a particular step (a service can have multiple bindings). So, for instance, an LPML activity will be represented by an IM step, while elements such as semantic annotations will be implemented as properties in the IM. It is possible that as part of the work for writing the transformation, the IM metamodel will be extended to better correspond to the needs of the LPML (perhaps a Step should point to a ServiceCollection in order to better represent the fact that one Activity in LPML can have a ServiceList. association).

IM to BPEL

Similarly to the first transformation, the “IntermediateModel to BPEL” phase is made up of a plugin which will iterate over all the elements of the IntermediateModel, and will enable to transform each element into BPEL 2.0 elements using the elements and the annotations previously added by the “LPML to IntermediateModel” transformation. For instance, an IM step will be either a BPEL basic activity such as an Invoke task, or a BPEL structured activity

such as a loop, depending on the corresponding annotation.

5.3 Summary and Remarks

LPML API provides complete programmatic support for the building and management of LPML process models, their serialization into RDF format for storage purposes and their transformation into SOA4ALL extended BPEL format for execution purposes. LPML API is build upon wide accepted MDD frameworks, such as Eclipse EMF and JenaBeans. The LPML transformation into BPEL also relies on EMF and STP-IM.

In addition to the LPML API framework specification, we have also described the BPEL 2.0 extensions provided by SOA4ALL to support LPML models execution and runtime adaptation features.

6. Language Evaluation

In this section we will evaluate the contents of the proposed process composition language by appealing to the concept of ontological completeness and coverage. The way in which target SOA4All users will perceive the language is mediated through the language representation developed in T2.6, so usability-style evaluations will be conducted within T2.5 in due course.

6.1 User categorization

Evaluating a language would be strongly dependent on both the target users and usage of the language, for example the best language for controlling either a real or a robotic pet would be much more limited in scope than the best language used to write fiction. The target usage of the language is covered by SOA4All use cases as developed by WP7, 8 and 9, and this document makes explicit reference to the requirements specified by each use case.

As far as the target users are concerned, in SOA4All we can categorise users as belonging to the following types:

1. IT Experts – people who have substantial IT education, or substantial experience of using and developing software (and possibly service) systems;
2. Domain Experts – people who have substantial education and experience in domains other than IT, and use the systems to achieve work objectives. These can be further divided according to the main use cases of SOA4All:
 - a. WP7 - Civil Servants / Town-Hall Administrators
 - b. WP8 - Telecom Engineers
 - c. WP9 - e-Marketing specialists
3. Casual Users – these are the closest match to “All” in “SOA4All”, and their needs are covered by the overall developments of SOA4All Studio in WP2. They are not expected to be IT nor domain experts, and even when they are, they would be accessing services for purposes different from their main line of work. Because the domain (or IT) expertise is no longer supporting the users in their use of SOA4All, we should use the profiles of these users as the “lowest common denominator” profile for SOA4All users. Students from management and social science disciplines are a good proxy for this target group of users.

6.2 Completeness and Expressiveness of the LPML

An accepted standard approach for evaluating the completeness of a language is the approach of measuring *the ontological completeness*, using the well-established Bunge–Wand–Weber (BWW) models, in particular the representation model.

Wand and Weber (Wand and Weber 1989) have studied the philosophical works of Mario Bunge (Bunge 1977) to seek an objective foundation for analysing the coverage of concepts when developing Information Systems. They define three different types of models: representation model, state-tracking model, and good decomposition model. Of these we will focus on the first, since it provides a set of constructs which are thought sufficient to represent the structure and the behaviour of the world. Annex D contains a plain English explanation of the concepts within the BWW representation model, adapted from (Green, Rosemann et al. 2007). It is claimed (Green, Rosemann et al. 2007) that the BWW representational model has been used in over thirty analysis projects spanning a number of

different representational grammars, so its use for such evaluations has been extensively validated.

In one such study, Green et al (Green, Rosemann et al. 2007) have mapped the BWW representation model onto the constructs of BPEL. In the table below, we have adapted this analysis for LPML as a starting point of our ontological completeness evaluation.

Table 1 . Mapping the BWW Representation model to LPML (based on LPML Metamodel)

Ontological construct	Explanation
Thing	
Property:	Name, Semantic Annotations (or Descriptions)
In general	
In particular	
Hereditary	
Emergent	
Intrinsic	Correlation set
Non-binding mutual	
Binding mutual	WSDL Service link, SA-WSDL Service link, WSMO-(Lite) Service link
Attributes	Names of properties
Class	Partner/Conversation (for semantic Web Service)
Kind	
State	Connector: A set of semantically annotated variables with value (syntactic and semantic) assignments
Conceivable state space	
State law	
Lawful state space	
Event	Message, Reply, Invoke, Receive, Create Instance (on Activity), InteractionActivity, ExtensionActivity, Connector
Conceivable event space	
Transformation	Control flow based: receive, assign, exclusive gateway, inclusive gateway, parallel gateway, FaultHandler, scope, flow Data flow based (Connector): merge, split, filter, reduction, sort, subdescription, aggregation, loop, count
Lawful transformation	If, While, Repeat-Until, For Each
Stability Condition	Expression (semantic annotation based)
Lawful event space	
History	
Acts-on	Role
Coupling: binding mutual property	Semantic Web Service link, Activity link, (Atomic or Composite) Goal link
System	Process Instance
System composition	Partner/Conversations (for Semantic Web Service)
System environment	
System structure	Partner/Conversations (for Semantic Web Service)
Subsystem	
System decomposition	
Level structure	
External event	Message
Stable state	
Unstable state	
Internal event	Receive, Invoke, Reply, Create Instance (on Activity), Invoke

Well-defined event	Create Instance (on Activity)
Poorly defined event	Message, Reply

A notable gap of the LPML coverage in the table is the lack of representation for ‘thing’. This is common amongst great many process specification / service composition languages, including BPEL. It has been extensively discussed elsewhere (Green, Rosemann et al. 2007), formulating propositions to be subjected to further testing. For example that the lack of representation of thing may cause lack of clarity in representing participants in the process, and confusion when some instances of class (things) participate in a relationship with other instances whilst other instances do not. Also missing are representations of ‘state law’ and the related ‘lawful state space’ and ‘lawful event space’, but perhaps more crucial is the lack of direct representation for ‘System environment’, which is expected to lead to users lacking clear distinction between things inside and outside of the system. The difficulty in identifying things outside of the system is expected to lead to difficulties in identifying which entities can generate significant external events (Green, Rosemann et al. 2007). Within SOA4All we envisage to model states and the system environment by preconditions and effects.

Also notable is the mapping of “event” and “transformation” BWV constructs to an LPML activity. Activities are used to represent events which may arise in a business process. In difference to BPEL, LMPL has no explicit “Wait” construct, so an activity has to be specified in order to directly represent time delay as event trigger.

‘Transformation’ is a core LPML construct, and this has been mapped to a number of LPML elements, divided into two groups: Control Flow group and Data Flow.

6.3 Pattern-based analysis of the LPML

Ontological coverage analysis using the BWV representation model from the previous section revealed that one specific BWV concept: ‘transformation’ maps to a number of LMPL constructs. The rationale for this is routed in the use of different LMPL constructs to express different patterns of control and communication flow. In this section we will analyse the coverage provided by LPML against a benchmark set of 20 control flow patterns found in workflow systems (van der Aalst, ter Hofstede et al. 2003b) and a set of six communication patterns found in Enterprise Application Integration systems (Ruh, Maginnis et al. 2001). The analysis is adapted from (Wohed, van der Aalst et al. 2003) where it has been applied to BPEL.

Pattern		Description	Implementation using LPML
Control flow patterns found in workflow /	Sequence	A cannot start before B completes	Sequence flow
	Parallel Split	At this point, concurrent execution of A and B is enabled	Parallel gateway
	Synchronization	C can only start once the concurrent A and B complete	Simulated by sequence flow and parallel gateway
	Exclusive Choice	At this point, one of ($A_1..A_n$) is chosen based on data	Inclusive gateway
	Simple Merge	C can only start once A or B completes, only A or B can be run	Simulated by Parallel gateway

	Multi Choice	At this point, two or more of $(A_1..A_n)$ are chosen based on data	Parallel gateway with conditional expressions expressed by means of semantic annotation
	Synchronizing Merge	C can only start once all the active A_i from $(A_1..A_n)$ complete	Simulated by Parallel gateway
	MultiMerge	C is started once for each completion of active A_i from $(A_1..A_n)$	Simulated by Parallel gateway
	Discriminator	C is started just once with the first completion from all active A_i ($i \in \{1..n\}$)	Parallel gateway with conditional expressions expressed by means of semantic annotation
	Arbitrary Cycles	Any portion of the process should be visited repeatedly	Cycles will be supported by the next LPML version for M30
	Implicit Termination	Process completes when nothing left to do, without explicit term. activity	Implicit not supported, explicit "End activity" used instead
	MI without Synchronization	A number of concurrent (sub)process instances are created	We will figure out by M30 whether multi-instances need to be supported.
	MI with a Priori Design Time Knowledge	A number of concurrent (sub)process instances are created and their completion synchronised, before proceeding with the rest of the process.	We will figure out by M30 whether multi-instances need to be supported.
	MI with/without a Priori Runtime Knowledge		
	Deferred Choice	Point of choosing A or B is reached before the decision data is available.	Parallel gateway with conditional expressions expressed by means of semantic annotation
	Interleaved Parallel Routing	Each A_i from $(A_1..A_n)$ is executed exactly once in an order determined just after the previous activity	Can be modelled through preconditions
	Milestone	C can only be started if A has finished but a subsequent B has not yet started	Can be modelled through preconditions
	Cancel Activity	Terminate activity	Lower level constructs
	Cancel Case	Terminate instance	Not supported

Communication Patterns (EAI)	Request/Reply	Sender waits for a reply before continuing	Depends on the transformation of the LPML in an executable language. In LPML the activity will handle this pattern.
	OneWay	Sender waits for an acknowledgment before continuing	Depends on the transformation of the LPML in an executable language. In LPML the activity will handle this pattern.

	Synchronous Polling	Sender polls for a response whilst receiving one.	Depends on the transformation of the LPML in an executable language. In LPML the activity will handle this pattern.
	Message Passing	Sender sends a message and continues processing	Depends on the service executing the activity. However we won't explicitly model that.
	Publish/Subscribe	Request sent to all receivers which have previously declared interest	Can be modelled implicitly by data split or control flow split.
	Broadcast	Request sent to all receivers in a network, each decides whether to act	Not supported

6.4 Recommendations

The analysis of the ontological coverage and the pattern-based coverage revealed satisfactory coverage of the exhaustive sets of concepts and patterns used for evaluation. The most important shortcomings seem to be common with BPEL in terms of lack of representation for environment and things interacting with the process.

Before recommending the increase of the coverage by introducing new language constructs, we should be aware of the lightweight nature of the language, which would motivate that the language is “fit for purpose” only and not complete. At any rate, the coverage of the language is the same as the underlying BPEL, which is much more complex technically.

7. Conclusions

This deliverable gave an insight into the advanced specification of the lightweight process modelling methodology and Lightweight Process Modelling Language (LPML). We have first described the three design principles abstraction of process models, the use of semantic annotations, and context-awareness. The LPML comprises new elements and has picked up selected concepts, elements, and artifacts of BPMN and BPEL. The selection was performed considering the creation of a lightweight language. A coherent metamodel of the LPML and its elements, properties, and relationships has been presented. On the programmatic perspective, the LPML is managed by an API. This deliverable described as well this LPML API, which provides programmatic process modeling and serialization support, either for storage as RDF and transformation into SOA4ALL extended BPEL 2.0⁷ or other executable languages.

The ability to be transformed into other process modelling languages, both executable and for documentation purposes, is key to the LPML. It is designed to be flexible enough in order to be easily transformed into various, existing process modelling languages. For the execution in SOA4All we selected to transform the LPML into BPEL.

A concept for the evaluation of the contents of the LPML has been covered as well by this deliverable in order to prove the applicability of the LPML. The evaluation approach is based on the concept of ontological completeness and coverage. The next step in evaluating the LPML is to extend and apply the evaluation concept. We have to let end-users model their processes by using the LPML. Any feedback given by these end-users will provide some ideas to further improve the language by M30.

The application of ontologies in process modeling languages has to be further evaluated in the future. In the LPML we have chosen a pragmatic proceeding. Ontologies are only applied for specifying activities. At the moment this approach seems to be most appropriate in order to achieve a working solution. However, as semantic descriptions will evolve and be attached to more and more artifacts, other full-fledged ontological approaches like BPMO can be applicable as well.

The usefulness of the new process modeling features as provided by the LPML has to be proved. These features will be implemented by the WP6 components, the composer, the optimizer, the template generator, and the execution engine. The final evaluation can only be done when these components are in use within SOA4All.

Another work to be done until M30 is the advancement of context-integration. The context reference can be implemented as semantic annotation. At the moment, a coherent concept for context-awareness in SOA4All is not yet provided. We envisage providing that concept by month 30.

In order to provide full benefit of the user support through patterns, templates, and goals a critical mass of existing artefacts have to be provided. Especially domain-specific patterns, templates and goals will provide an added value to the user. However, in the context of the research project SOA4All we will only provide a couple of patterns, templates, and goals based on the use cases. The provision of a large artefact repository is not focused in SOA4All.

SOA4All will provide a final version of the LPML specification in D6.3.3 which is due in M30.

⁷ We refer to the extended BPEL language used by D6.5.1 Execution Environment

8. References

- Akman, V. and M. Surav (1996). Steps towards formalizing context, American Association for Artificial Intelligence.
- Brezillon, P. and S. Abu-Hakima (1995). "Using knowledge in its context: Report on the ijcai-93 workshop." AI Magazine **16**(1).
- Bunge, M. (1977). "Treatise on Basic Philosophy." Ontology I: The furniture of the world **3**.
- Buschmann, F., K. Henney, et al. (2007). "Past, Present, and Future Trends in Software Patterns." IEEE Software **24**(7/8): 31-37.
- Coalition, W. M. (2005). Process Definition Interface -- XML Process Definition Language, WfMC Standards and Working Groups.
- Dey, A. K. (2000). Providing architectural support for building context-aware applications. Georgia Institute of Technology.
- Foundation, E. (2009). Eclipse Modelling Framework (EMF).
- Green, P., M. Rosemann, et al. (2007). "Candidate Interoperability Standards: An Ontological Overlap Analysis." Data & Knowledge Engineering **62**(2): 274-291.
- Keller, U., R. Lara, et al. (2004). WSMO Web Service Discovery.
- Koehler, J. and J. Vanhatalo (2007). "Process anti-patterns: How to avoid the common traps of business process modelling." IBM WebSphere Developer Technical Journal.
- McCarthy, J. (1993). Notes on formalizing context. IJCAI. M. Kaumann. San Mateo, California: 555-562.
- Medicke, J. and D. McDavid (2004). "Patterns for Business Process Modelling." Business Integration Journal **1**: 32-35.
- Nitzsche, J., T. van Lessen, et al. (2007a). BPEL for Semantic Web Services (BPEL4SWS). On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, Springer.
- Nitzsche, J., T. van Lessen, et al. (2007b). BPEL-Light. 5th International Conference on Business Process Management (BPM 2007), Springer.
- Oasis (2006). Web Services Business Process Execution Language Version 2.0, OASIS.
- Rosemann, M. and J. Recker (2006). Context-aware process design: Exploring the extrinsic drivers for process flexibility. 18th International Conference on Advanced Information Systems Engineering, Luxembourg, Namur University Press.
- Rosemann, M., J. Recker, et al. (2008). "Contextualisation of business processes." International Journal of Business Process Integration and Management (IJBPMIM) **3**(1): 47-60.
- Rosemann, M., J. Recker, et al. (2006). Understanding context-awareness in business process design. 17th Australasian Conference on Information Systems, Adelaide, Australia.
- Ruh, W. A., F. X. Maginnis, et al. (2001). Enterprise Application Integration: A Wiley Tech Brief, John Wiley and Sons Inc.
- Saidani, O. and S. Nurcan (2007). Towards context-aware business process modelling. 8th Workshop on Business Processes and Support Systems: Requirements for flexibility and the ways to achieve it.

Tran, H. N., B. Coulette, et al. (2007). Broadening the Use of Process Patterns for Modelling Processes. SEKE, Knowledge Systems Institute Graduate Schools.

van der Aalst, W. M. P., A. H. M. ter Hofstede, et al. (2003a). Workflow Patterns. Distributed and Parallel Databases: 5-51.

van der Aalst, W. M. P., A. H. M. ter Hofstede, et al. (2003b). "Workflow Patterns." Distributed and Parallel Databases **14**(1).

Vitvar, T., J. Kopecky, et al. (2008). WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web.

Wand, Y. and R. Weber (1989). An ontological evaluation of systems analysis and design methods. Information System Concepts: An Indepth Analysis. E. D. Falkenberg and P. Lindgreen: 79-107.

Wohed, P., W. M. P. van der Aalst, et al. (2003). Analysis of Web Services Composition Languages: The Case of BPEL4WS. Proceedings of Conceptual Modelling, Springer.

zur Muehlen, M. and J. C. Recker (2008). How Much Language is Enough? Theoretical and Practical Use of the Business Process Modeling Notation, Springer.

Annex A. Sample File

The following sample XML file describes the process “UpdateCatalog” out of WP9. The LPML model in java code has been serialized in XML code.

```
<org.SOA4All.lpml.impl.ProcessImpl>
  <startElement class="org.SOA4All.lpml.impl.ActivityImpl">
    <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ffe</iD>
    <flows>
      <org.SOA4All.lpml.impl.FlowImpl>
        <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fcb</iD>
        <flows/>
      </org.SOA4All.lpml.impl.FlowImpl>
    </flows>
    <name>Start</name>
    <startElement>true</startElement>
    <endElement>false</endElement>
  </startElement>
  <processElements>
    <org.SOA4All.lpml.impl.ActivityImpl reference="../../startElement"/>
    <org.SOA4All.lpml.impl.ActivityImpl>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ffd</iD>
      <flows>
        <org.SOA4All.lpml.impl.FlowImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fb8</iD>
        </org.SOA4All.lpml.impl.FlowImpl>
      </flows>
      <name>End</name>
      <startElement>false</startElement>
      <endElement>true</endElement>
    </org.SOA4All.lpml.impl.ActivityImpl>
    <org.SOA4All.lpml.impl.ParallelGatewayImpl>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ffc</iD>
      <flows>
        <org.SOA4All.lpml.impl.FlowImpl
reference="../../../../../startElement/flows/org.SOA4All.lpml.impl.FlowImpl"/>
        <org.SOA4All.lpml.impl.FlowImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fca</iD>
        </org.SOA4All.lpml.impl.FlowImpl>
        <org.SOA4All.lpml.impl.FlowImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fc7</iD>
        </org.SOA4All.lpml.impl.FlowImpl>
        <org.SOA4All.lpml.impl.FlowImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fbc</iD>
        </org.SOA4All.lpml.impl.FlowImpl>
      </flows>
      <split>true</split>
    </org.SOA4All.lpml.impl.ParallelGatewayImpl>
    <org.SOA4All.lpml.impl.ParallelGatewayImpl>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ffb</iD>
      <flows>
        <org.SOA4All.lpml.impl.FlowImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fc8</iD>
        </org.SOA4All.lpml.impl.FlowImpl>
        <org.SOA4All.lpml.impl.FlowImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fc0</iD>
          <condition class="org.SOA4All.lpml.impl.SemanticAnnotationImpl">
            <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fbd</iD>
          </condition>
        </org.SOA4All.lpml.impl.FlowImpl>
      </flows>
    </org.SOA4All.lpml.impl.ParallelGatewayImpl>
  </processElements>
</org.SOA4All.lpml.impl.ProcessImpl>
```

```

<referenceURI>http://www.SOA4All.eu/webshop#loopOnCollection</referenceURI>

<expression>http://www.SOA4All.eu/webshop#incompleted</expression>
  <type>SELECTION_CRITERIA</type>
  </condition>
  </org.SOA4All.lpmpl.impl.FlowImpl>
  <org.SOA4All.lpmpl.impl.FlowImpl>
    <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fb9</iD>
  </org.SOA4All.lpmpl.impl.FlowImpl>
  <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ActivityImpl[2]/flows/org.SOA4All
.lpmpl.impl.FlowImpl"/>
  </flows>
  <split>false</split>
</org.SOA4All.lpmpl.impl.ParallelGatewayImpl>
<org.SOA4All.lpmpl.impl.ExclusiveGatewayImpl>
  <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7ffa</iD>
  <flows>
    <org.SOA4All.lpmpl.impl.FlowImpl>
      <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fc6</iD>
    </org.SOA4All.lpmpl.impl.FlowImpl>
    <org.SOA4All.lpmpl.impl.FlowImpl>
      <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fc5</iD>
      <condition class="org.SOA4All.lpmpl.impl.SemanticAnnotationImpl">
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fc4</iD>
      </condition>
    </org.SOA4All.lpmpl.impl.FlowImpl>
  </flows>
</org.SOA4All.lpmpl.impl.ExclusiveGatewayImpl>
<org.SOA4All.lpmpl.impl.ParallelGatewayImpl>
  <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fb9</iD>
  <flows>
    <org.SOA4All.lpmpl.impl.FlowImpl>
      <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fc1</iD>
    </org.SOA4All.lpmpl.impl.FlowImpl>
    <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ParallelGatewayImpl[2]/flows/org.
SOA4All.lpmpl.impl.FlowImpl[2]"/>
    </org.SOA4All.lpmpl.impl.FlowImpl>
  </flows>
  <split>false</split>
</org.SOA4All.lpmpl.impl.ExclusiveGatewayImpl>
<org.SOA4All.lpmpl.impl.ActivityImpl>
  <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fe8</iD>
  <flows>
    <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ParallelGatewayImpl/flows/org.SOA
4All.lpmpl.impl.FlowImpl[2]"/>
  </flows>

```

```

    <org.SOA4All.lpm1.impl.FlowImpl>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fc9</iD>
    </org.SOA4All.lpm1.impl.FlowImpl>
  </flows>
  <name>Clothing provider: Update Catalog</name>
  <operation>updateCatalog</operation>
  <startElement>false</startElement>
  <endElement>false</endElement>
  <conversation class="org.SOA4All.lpm1.impl.ConversationImpl">
    <compositeGoal>true</compositeGoal>
    <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ff8</iD>
    <service class="org.SOA4All.lpm1.impl.ServiceImpl">

<serviceReference>http://www.clothingprovider.com/services/clothingProvider
.wsd1</serviceReference>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fed</iD>
    </service>
    <goal class="org.SOA4All.lpm1.impl.GoalImpl">

<goalReference>http://www.SOA4All.eu/webshop/clothingProviderGoal.wsmo</goal
lReference>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ff4</iD>
      <name>Clothing Provider</name>
    </goal>
  </conversation>
  <humanTask>false</humanTask>
  <synchronous>true</synchronous>
  <outputParameters>
    <org.SOA4All.lpm1.impl.ParameterImpl>
      <semanticAnnotations>
        <org.SOA4All.lpm1.impl.SemanticAnnotationImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fdf</iD>

<referenceURI>http://www.SOA4All.eu/webshop#listOfProducts</referenceURI>
          <type>META_DATA</type>
        </org.SOA4All.lpm1.impl.SemanticAnnotationImpl>
      </semanticAnnotations>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fe0</iD>
    </org.SOA4All.lpm1.impl.ParameterImpl>
  </outputParameters>
</org.SOA4All.lpm1.impl.ActivityImpl>
<org.SOA4All.lpm1.impl.ActivityImpl>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fe7</iD>
  <flows>
    <org.SOA4All.lpm1.impl.FlowImpl
reference="../../../../org.SOA4All.lpm1.impl.ParallelGatewayImpl/flows/org.SOA
4All.lpm1.impl.FlowImpl[3]"/>
    <org.SOA4All.lpm1.impl.FlowImpl
reference="../../../../org.SOA4All.lpm1.impl.ExclusiveGatewayImpl/flows/org.SO
A4All.lpm1.impl.FlowImpl"/>
  </flows>
  <name>Footwear provider: get product list</name>
  <operation>getProductList</operation>
  <startElement>false</startElement>
  <endElement>false</endElement>
  <conversation class="org.SOA4All.lpm1.impl.ConversationImpl">
    <compositeGoal>true</compositeGoal>
    <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ff7</iD>
    <service class="org.SOA4All.lpm1.impl.ServiceImpl">

```

```

<serviceReference>http://www.footwearprovider.com/services/footwearProvider
B.wsdl</serviceReference>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7feb</iD>
</service>
<goal class="org.SOA4All.lpml.impl.GoalImpl">
  <semanticAnnotations>
    <org.SOA4All.lpml.impl.SemanticAnnotationImpl>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ff2</iD>

<referenceURI>http://www.SOA4All.eu/webshop#footwear_provider</referenceURI>
>
    <type>FUNCTIONAL_CLASSIFICATION</type>
  </org.SOA4All.lpml.impl.SemanticAnnotationImpl>
  <org.SOA4All.lpml.impl.SemanticAnnotationImpl>
    <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ff1</iD>

<referenceURI>http://www.SOA4All.eu/webshop#oneweek_delivery</referenceURI>
  <type>NON_FUNCTIONAL_PROPERTY</type>
  </org.SOA4All.lpml.impl.SemanticAnnotationImpl>
</semanticAnnotations>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ff3</iD>
  <name>Footwear Provider</name>
</goal>
</conversation>
<humanTask>false</humanTask>
<synchronous>true</synchronous>
<outputParameters>
  <org.SOA4All.lpml.impl.ParameterImpl>
    <semanticAnnotations>
      <org.SOA4All.lpml.impl.SemanticAnnotationImpl>
        <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fdd</iD>

<referenceURI>http://www.SOA4All.eu/webshop#productlist</referenceURI>
  <type>META_DATA</type>
  </org.SOA4All.lpml.impl.SemanticAnnotationImpl>
</semanticAnnotations>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fde</iD>
  </org.SOA4All.lpml.impl.ParameterImpl>
</outputParameters>
</org.SOA4All.lpml.impl.ActivityImpl>
<org.SOA4All.lpml.impl.ActivityImpl>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fe6</iD>
  <flows>
    <org.SOA4All.lpml.impl.FlowImpl
reference="../../../../org.SOA4All.lpml.impl.ExclusiveGatewayImpl/flows/org.SO
A4All.lpml.impl.FlowImpl[2]"/>
    <org.SOA4All.lpml.impl.FlowImpl>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fc3</iD>
    </org.SOA4All.lpml.impl.FlowImpl>
  </flows>
  <name>Footwear provider: get product data</name>
  <operation>getProductData</operation>
  <startElement>false</startElement>
  <endElement>false</endElement>
  <conversation class="org.SOA4All.lpml.impl.ConversationImpl"
reference="../../../../org.SOA4All.lpml.impl.ActivityImpl[4]/conversation"/>
  <humanTask>false</humanTask>
  <synchronous>true</synchronous>
  <inputParameters>

```



```

        <org.SOA4All.lpml.impl.ParameterImpl>
        <semanticAnnotations>
        <org.SOA4All.lpml.impl.SemanticAnnotationImpl>
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fdb</iD>

<referenceURI>http://www.SOA4All.eu/webshop#productid</referenceURI>
        <type>META_DATA</type>
        </org.SOA4All.lpml.impl.SemanticAnnotationImpl>
        </semanticAnnotations>
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fdc</iD>
        </org.SOA4All.lpml.impl.ParameterImpl>
</inputParameters>
<outputParameters>
        <org.SOA4All.lpml.impl.ParameterImpl>
        <semanticAnnotations>
        <org.SOA4All.lpml.impl.SemanticAnnotationImpl>
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fd9</iD>

<referenceURI>http://www.SOA4All.eu/webshop#productdata</referenceURI>
        <type>META_DATA</type>
        </org.SOA4All.lpml.impl.SemanticAnnotationImpl>
        </semanticAnnotations>
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fda</iD>
        </org.SOA4All.lpml.impl.ParameterImpl>
</outputParameters>
</org.SOA4All.lpml.impl.ActivityImpl>
<org.SOA4All.lpml.impl.ActivityImpl>
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fe5</iD>
        <flows>
        <org.SOA4All.lpml.impl.FlowImpl
reference=" ../ ../ ../org.SOA4All.lpml.impl.ActivityImpl[5]/flows/org.SOA4All
.lpml.impl.FlowImpl[2]" />
        <org.SOA4All.lpml.impl.FlowImpl>
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fc2</iD>
        </org.SOA4All.lpml.impl.FlowImpl>
</flows>
<name>Footwear provider: get product price</name>
<operation>getProductPrice</operation>
<startElement>>false</startElement>
<endElement>>false</endElement>
<conversation class="org.SOA4All.lpml.impl.ConversationImpl"
reference=" ../ ../ ../org.SOA4All.lpml.impl.ActivityImpl[4]/conversation" />
        <humanTask>>false</humanTask>
        <synchronous>true</synchronous>
        <inputParameters>
        <org.SOA4All.lpml.impl.ParameterImpl
reference=" ../ ../ ../org.SOA4All.lpml.impl.ActivityImpl[5]/inputParameters/o
rg.SOA4All.lpml.impl.ParameterImpl" />
        </inputParameters>
        <outputParameters>
        <org.SOA4All.lpml.impl.ParameterImpl>
        <semanticAnnotations>
        <org.SOA4All.lpml.impl.SemanticAnnotationImpl>
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fd7</iD>

<referenceURI>http://www.SOA4All.eu/webshop#productdata</referenceURI>
        <type>META_DATA</type>
        </org.SOA4All.lpml.impl.SemanticAnnotationImpl>
        </semanticAnnotations>
        <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fd8</iD>

```

```

    </org.SOA4All.lpmpl.impl.ParameterImpl>
  </outputParameters>
</org.SOA4All.lpmpl.impl.ActivityImpl>
<org.SOA4All.lpmpl.impl.ActivityImpl>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fe4</iD>
  <flows>
    <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ParallelGatewayImpl/flows/org.SOA
4All.lpmpl.impl.FlowImpl[4]"/>
    <org.SOA4All.lpmpl.impl.FlowImpl>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fbb</iD>
    </org.SOA4All.lpmpl.impl.FlowImpl>
  </flows>
  <name>Accessories provider: get list of product descriptions</name>
  <operation>getListOfProductDescriptions</operation>
  <startElement>false</startElement>
  <endElement>false</endElement>
  <conversation class="org.SOA4All.lpmpl.impl.ConversationImpl">
    <compositeGoal>true</compositeGoal>
    <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ff6</iD>
    <service class="org.SOA4All.lpmpl.impl.ServiceImpl">

<serviceReference>http://www.accessoriesprovider.com/services/accessoriesPr
ovider.wsdl</serviceReference>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fea</iD>
    </service>
    <goal class="org.SOA4All.lpmpl.impl.GoalImpl">
      <semanticAnnotations>
        <org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fef</iD>

<referenceURI>http://www.SOA4All.eu/webshop#accessories_provider</reference
URI>
          <type>FUNCTIONAL_CLASSIFICATION</type>
        </org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
        <org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fee</iD>

<referenceURI>http://www.SOA4All.eu/webshop#special_offers</referenceURI>
          <type>NON_FUNCTIONAL_PROPERTY</type>
        </org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
      </semanticAnnotations>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7ff0</iD>
      <name>Accessories Provider</name>
    </goal>
  </conversation>
  <humanTask>false</humanTask>
  <synchronous>true</synchronous>
  <outputParameters>
    <org.SOA4All.lpmpl.impl.ParameterImpl>
      <semanticAnnotations>
        <org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
          <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fd5</iD>

<referenceURI>http://www.SOA4All.eu/webshop#listOfProductDescription</refer
enceURI>
          <type>META_DATA</type>
        </org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
      </semanticAnnotations>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fd6</iD>

```

```

    </org.SOA4All.lpmpl.impl.ParameterImpl>
  </outputParameters>
</org.SOA4All.lpmpl.impl.ActivityImpl>
<org.SOA4All.lpmpl.impl.ActivityImpl>
  <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fe3</iD>
  <flows>
    <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ActivityImpl[7]/flows/org.SOA4All
.lpmpl.impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl>
      <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fba</iD>
    </org.SOA4All.lpmpl.impl.FlowImpl>
  </flows>
  <name>Accessories provider: get list of product prices</name>
  <operation>getListOfProductPrices</operation>
  <startElement>false</startElement>
  <endElement>false</endElement>
  <conversation class="org.SOA4All.lpmpl.impl.ConversationImpl"
reference="../../../../org.SOA4All.lpmpl.impl.ActivityImpl[7]/conversation" />
  <humanTask>false</humanTask>
  <synchronous>true</synchronous>
  <outputParameters>
    <org.SOA4All.lpmpl.impl.ParameterImpl>
      <semanticAnnotations>
        <org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
          <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fd3</iD>
        </org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
      </semanticAnnotations>
    </org.SOA4All.lpmpl.impl.ParameterImpl>
  </outputParameters>
</org.SOA4All.lpmpl.impl.ActivityImpl>
<org.SOA4All.lpmpl.impl.ActivityImpl>
  <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7fe2</iD>
  <flows>
    <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ActivityImpl[3]/flows/org.SOA4All
.lpmpl.impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ParallelGatewayImpl[2]/flows/org.
SOA4All.lpmpl.impl.FlowImpl" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ActivityImpl[8]/flows/org.SOA4All
.lpmpl.impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference="../../../../org.SOA4All.lpmpl.impl.ParallelGatewayImpl[2]/flows/org.
SOA4All.lpmpl.impl.FlowImpl[3]" />
  </flows>
  <name>Webshop provider: aggregate catalog</name>
  <operation>aggregateCatalog</operation>
  <startElement>false</startElement>
  <endElement>false</endElement>
  <conversation class="org.SOA4All.lpmpl.impl.ConversationImpl">
    <compositeGoal>true</compositeGoal>
    <iD>11d1def534ealbe0:2a1172a4:1239522b160:-7ff5</iD>
    <service class="org.SOA4All.lpmpl.impl.ServiceImpl">

```

```

<serviceReference>http://www.SOA4All.eu/webshop/services/webshopProvider.ws
dl</serviceReference>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fe9</iD>
  </service>
</conversation>
<humanTask>false</humanTask>
<synchronous>true</synchronous>
<inputParameters>
  <org.SOA4All.lpml.impl.ParameterImpl>
    <semanticAnnotations>
      <org.SOA4All.lpml.impl.SemanticAnnotationImpl>
        <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fd1</iD>
      </org.SOA4All.lpml.impl.SemanticAnnotationImpl>
    </semanticAnnotations>
  </org.SOA4All.lpml.impl.ParameterImpl>
</inputParameters>
<referenceURI>http://www.SOA4All.eu/webshop#webshopcatalog</referenceURI>
  <type>META_DATA</type>
</org.SOA4All.lpml.impl.SemanticAnnotationImpl>
<org.SOA4All.lpml.impl.SemanticAnnotationImpl>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fcf</iD>
</org.SOA4All.lpml.impl.SemanticAnnotationImpl>
<referenceURI>http://www.SOA4All.eu/webshop#webshopproduct</referenceURI>
  <type>META_DATA</type>
</org.SOA4All.lpml.impl.SemanticAnnotationImpl>
<semanticAnnotations>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fd2</iD>
</org.SOA4All.lpml.impl.SemanticAnnotationImpl>
</inputParameters>
<outputParameters/>
</org.SOA4All.lpml.impl.ActivityImpl>
<org.SOA4All.lpml.impl.ActivityImpl>
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fe1</iD>
  <flows>
    <org.SOA4All.lpml.impl.FlowImpl
reference="../../../../org.SOA4All.lpml.impl.ActivityImpl[6]/flows/org.SOA4All
.lpml.impl.FlowImpl[2]"/>
    <org.SOA4All.lpml.impl.FlowImpl
reference="../../../../org.SOA4All.lpml.impl.ExclusiveGatewayImpl[2]/flows/org
.SOA4All.lpml.impl.FlowImpl"/>
  </flows>
  <name>Webshop provider: aggregate product to catalog</name>
  <operation>aggregateProductToCatalog</operation>
  <startElement>false</startElement>
  <endElement>false</endElement>
  <conversation class="org.SOA4All.lpml.impl.ConversationImpl"
reference="../../../../org.SOA4All.lpml.impl.ActivityImpl[9]/conversation"/>
  <humanTask>false</humanTask>
  <synchronous>true</synchronous>
  <inputParameters>
    <org.SOA4All.lpml.impl.ParameterImpl
reference="../../../../org.SOA4All.lpml.impl.ActivityImpl[9]/inputParameters/o
rg.SOA4All.lpml.impl.ParameterImpl"/>
  </inputParameters>
  <outputParameters/>
</org.SOA4All.lpml.impl.ActivityImpl>
<org.SOA4All.lpml.impl.FlowImpl
reference="../../../../startElement/flows/org.SOA4All.lpml.impl.FlowImpl"/>
  <org.SOA4All.lpml.impl.FlowImpl
reference="../../../../org.SOA4All.lpml.impl.ParallelGatewayImpl/flows/org.SOA4All.l
pml.impl.FlowImpl[2]"/>

```

```

    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ActivityImpl[3]/flows/org.SOA4All.lpmpl.
impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ParallelGatewayImpl[2]/flows/org.SOA4Al
l.lpmpl.impl.FlowImpl" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ParallelGatewayImpl/flows/org.SOA4All.l
pml.impl.FlowImpl[3]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ExclusiveGatewayImpl/flows/org.SOA4All.
lpmpl.impl.FlowImpl" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ExclusiveGatewayImpl/flows/org.SOA4All.
lpmpl.impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ActivityImpl[5]/flows/org.SOA4All.lpmpl.
impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ActivityImpl[6]/flows/org.SOA4All.lpmpl.
impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ExclusiveGatewayImpl[2]/flows/org.SOA4A
ll.lpmpl.impl.FlowImpl" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ParallelGatewayImpl[2]/flows/org.SOA4Al
l.lpmpl.impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ExclusiveGatewayImpl/flows/org.SOA4All.
lpmpl.impl.FlowImpl[3]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ParallelGatewayImpl/flows/org.SOA4All.l
pml.impl.FlowImpl[4]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ActivityImpl[7]/flows/org.SOA4All.lpmpl.
impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ActivityImpl[8]/flows/org.SOA4All.lpmpl.
impl.FlowImpl[2]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ParallelGatewayImpl[2]/flows/org.SOA4Al
l.lpmpl.impl.FlowImpl[3]" />
    <org.SOA4All.lpmpl.impl.FlowImpl
reference=" ../org.SOA4All.lpmpl.impl.ActivityImpl[2]/flows/org.SOA4All.lpmpl.
impl.FlowImpl" />
  </processElements>
<endElement class="org.SOA4All.lpmpl.impl.ActivityImpl"
reference=" ../processElements/org.SOA4All.lpmpl.impl.ActivityImpl[2]" />
  <iD>11d1def534ea1be0:2a1172a4:1239522b160:-8000</iD>
  <semanticAnnotations>
    <org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
      <iD>11d1def534ea1be0:2a1172a4:1239522b160:-7fff</iD>

<referenceURI>http://www.SOA4All.eu/webshop#transactional</referenceURI>
  <type>NON_FUNCTIONAL_PROPERTY</type>
</org.SOA4All.lpmpl.impl.SemanticAnnotationImpl>
</semanticAnnotations>
</org.SOA4All.lpmpl.impl.ProcessImpl>

```

Annex B. Terminology

Terminology for the LPML	
<i>process</i> or <i>process model</i>	A <i>process</i> or <i>process model</i> is a description of a business process in sufficient detail that it is able to be directly executed by the SOA4All process editor and service composition execution engine.
<i>process model</i>	Is composed of a number of <i>activities</i> which are connected in the form of a directed graph.
<i>process instance</i>	An executing instance of a process model is called a <i>process instance</i> . There may be multiple instances of a particular process mode running simultaneously. Each of them should have an independent existence. The instances typically execute without reference to each other.
<i>activity instance</i>	Each invocation of an activity that executes is termed an <i>activity instance</i> . An activity instance may initiate one or several task instances during completion. In Figure 1, activity instance “main check” is initiated when activity instance “pro-check” completes and “check” result is successful where the edge between activity instances “pre-check” and “main check” indicates the condition that must be satisfied for the subsequent activity instance to be started.
<i>atomic activity</i>	An <i>atomic activity</i> is one which has a simple, self-contained definition and only one instance of the activity executes when it is initiated.
<i>composite activity</i>	A <i>composite activity</i> is a complex action which has its implementation described in terms of a <i>sub-process</i> . When a composite task is started, it passes control to the first activity(ies) in its corresponding sub-process. This sub-process executes to completion and at its conclusion, it passes control back to the composite activity. For example, composite activity “registration” is defined in terms of the sub-process comprising activities, “search for tax office in charge & notify tax office”, “send invoice” and “send confirmation”.
<i>multi-instance activity</i>	A <i>multi-instance activity</i> is an activity that may have multiple distinct execution instances running concurrently within the same process instance. Each of these instances executes independently. Only when a nominated number of these instances have completed is the activity following the multiple instance activity initiated.
<i>multi-instance composite activity</i>	A <i>multi-instance composite activity</i> is a combination of the two previous constructs and denotes an activity that may have multiple distinct execution instances each of which is composite structured in nature (i.e. has a corresponding <i>sub-process</i>).
<i>control channel</i>	The control flow between activities occurs via the <i>control channel</i> which is indicated by a solid arrow between tasks. There may also be a distinct <i>data channel</i> between process activities which provides a means of communicating <i>data elements</i> between two connected

	<p>tasks. Where a distinct data channel is intended between activities, it is illustrated with a broken (dash-dot) line between them as illustrated in Figure 1 between activity instances “registration” and “archive”. In other scenarios, the control and data channels are combined, however in both cases, where data elements are passed along a channel between activities, this is illustrated by the pass() relation, e.g. in Figure 1 data element M is passed from activity instance “registration” and “archive”.</p>
--	---

Annex C. LPML and semantic annotation languages

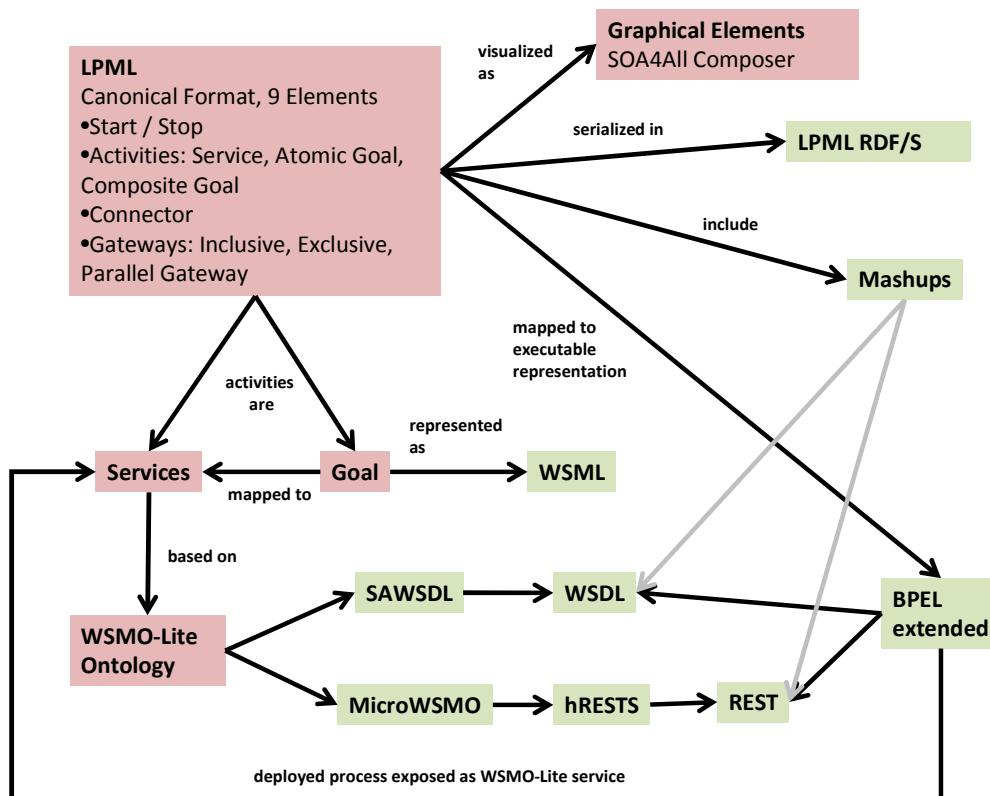


Figure 15: Relation of LPML to semantic annotation languages

Annex D. Evaluation information

Plain English definitions of the constructs of the BWW representation model

(adapted from (Green, Rosemann et al. 2007))

Ontological construct	Explanation
Thing ^a	A <i>thing</i> is the elementary unit in the BWW ontological model. The real world is made up of things. Two or more things (composite or simple) can be <i>associated</i> into a <i>composite</i> thing
Property ^a :	Things possess <i>properties</i> . A property is modelled via a function that maps the thing into some value. For example, the attribute “weight” represents a property that all humans possess. In this regard, weight is an
In general	attribute standing for a property <i>in general</i> . If we focus on the weight of a specific individual, however, we
In particular	would be concerned with a property <i>in particular</i> . A property of a composite thing that belongs to a component
Hereditary	thing is called an <i>hereditary</i> property. Otherwise it is called an <i>emergent</i> property. Some properties are inherent
Emergent	properties of individual things. Such properties are called <i>intrinsic</i> . Other properties are properties of pairs or
Intrinsic	many things. Such properties are called <i>mutual</i> . <i>Non-binding mutual</i> properties are those properties shared
Non-binding mutual	by two or more things that do not “make a difference” to the things involved; for example, order relations or
Binding mutual	equivalence relations. By contrast, <i>binding mutual</i> properties are those properties shared by two or more things
Attributes	that do “make a difference” to the things involved. <i>Attributes</i> are the names that we use to represent properties of things
Class	A <i>class</i> is a set of things that can be defined via their possessing a single property
Kind	A <i>kind</i> is a set of things that can be defined only via their possessing two or more common properties
State ^a	The vector of values for all property functions of a thing is the state of the thing
Conceivable state space	The set of all states that the thing might ever assume is the conceivable state space of the thing
State law	A state law restricts the values of the properties of a thing to a subset that is deemed lawful because of natural laws or human laws
Lawful state space	The lawful state space is the set of states of a thing that comply with the state laws of the thing. The lawful state space is usually a proper subset of the conceivable state space
Event	A change in state of a thing is an event
Conceivable event space	The event space of a thing is the set of all possible events that can occur in the thing
Transformation ^a	A transformation is a mapping from one state to another state
Lawful transformation	A lawful transformation defines which events in a thing are lawful
Lawful event space	The lawful event space is the set of all events in a thing that are lawful
History	The chronologically-ordered states that a thing traverses in time are the history of the thing
Acts-on	A thing acts on another thing if its existence affects the history of the other thing
Coupling:binding	Two things are said to be coupled (or interact) if one thing acts on the other. Furthermore, those two things are
mutual property	said to share a binding mutual property (or relation); that is, they participate in a relation that “makes a difference” to the things
System	A set of things is a system if, for any bi-partitioning of the set, couplings exist among things in the two subsets
System composition	The things in the system are its composition
System environment	Things that are not in the system but interact with things in the system are called the environment of the system
System structure	The set of couplings that exist among things within the system, and among things in the environment of the system and things in the system is called the structure
Subsystem	A subsystem is a system whose composition and structure are subsets of the composition and structure of another system
System decomposition	A decomposition of a system is a set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition
Level structure	A level structure defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself
External event	An external event is an event that arises in a thing, subsystem, or system by virtue of the action of some thing in the environment on the thing, subsystem, or system
Stable state ^a	A stable state is a state in which a thing, subsystem, or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event)
Unstable state	An unstable state is a state that will be changed into another state by virtue of the action of transformations in the system
Internal event	An internal event is an event that arises in a thing, subsystem, or system by virtue of lawful transformations in the thing, subsystem, or system
Well-defined event	A well-defined event is an event in which the subsequent state can always be predicted given that the prior state

	is known
Poorly defined event	A poorly defined event is an event in which the subsequent state cannot be predicted given that the prior state is known

^a A fundamental and core ontological construct