

D5.1 Exploratory Case Study

Emilia Cimpian², Schahram Dustdar¹, Jörg Hoffmann²,
Ta'id Holmes¹, Adina Sirbu², and Uwe Zdun¹

¹*Distributed Systems Group
Information Systems Institute
Vienna University of Technology
Austria
{lastname}@infosys.tuwien.ac.at*

²*Digital Enterprise Research Institute
Leopold – Franzens Universität Innsbruck
Austria
{firstname.lastname}@deri.at*

November 29, 2007



Abstract

This deliverable provides a detailed validation and evaluation of SemBiz technology. We propose a case study to clarify research topics and to investigate alternatives by evaluating and comparing the BPMO to other frameworks and by evaluating the theoretical approaches for querying, discovery and composition of business processes and comparing with related work.

Contents

1	Introduction	3
1.1	work package task 5.1	3
1.2	work package task 5.2	3
2	Evaluation of BP MO	3
2.1	BP MO Evaluation with OntoClean	4
2.1.1	OntoClean	4
2.1.2	BP MO Taxonomy	6
2.1.3	Metaproperties Assignment and Evaluation	7
2.2	Comparison with other Frameworks	16
2.2.1	OWL-S Process Ontology	16
2.2.2	SUPER BP MO	18
2.3	BP MO Evaluation Conclusions	19
3	Evaluation of Approaches	20
3.1	Query and Discovery	20
3.1.1	Semantic Query	20
3.1.2	Semantic Discovery	21
3.2	Functional Composition	22
3.2.1	Comparing to Related Work	22
3.2.2	Conclusions	24
3.3	Syntactic Composition	24
3.3.1	Comparing to Related Work	26
3.3.2	Conclusions	27
3.4	Deployment & Execution	27
4	Conclusion	28

1 Introduction

This first deliverable of the validation and evaluation workpackage of the SemBiz project wants to clarify research topics and investigate alternatives. This way the technology chosen and implementation realised can be validated and evaluated. Particularly we address the following work package tasks within this deliverable:

- T5.1: evaluate the BPMO and compare with other approaches
- T5.2: evaluate the techniques for querying, discovery, and composition of business processes and compare with related work

The next deliverable will

- T5.3: validate and evaluate the SemBiz technology on basis of the use case testing results.

1.1 work package task 5.1

Addressing T5.1 we have chosen OntoClean for evaluating the ontological adequacy of taxonomic relationships in BPMO.

1.2 work package task 5.2

To evaluate semantic query and discovery, where we adapt existing solutions, we provide the background that motivates our choices. In contrast, to evaluate our functional composition approach, entirely developed in the context of SemBiz, we provide an extensive comparison to existing solutions, and underline the novelty of our approach.

The framework for syntactic composition will be evaluated and compared to approaches in the field of service oriented computing. A framework for generic deployment and execution that will be presented in [7] will be introduced.

2 Evaluation of BPMO

The Business Process Modeling Ontology (BPMO) was designed for helping domain experts to semantically model their business processes. As such, its main focus is to provide the needed mechanisms for semantic business process modeling, based on a set of requirements previously defined.

This section provides an evaluation of the Business Process Modeling Ontology. As providing an evaluation methodology, or even a survey of the existing ones is out of the scope of this project, we are relying upon the already existing studies in choosing the most appropriate ontology evaluation technique, like [5] or [23]. Furthermore, these surveys analyze several aspects that need to be taken in consideration when evaluating an ontology, such as:

- what the evaluation is referring to: ontology's structure, architecture and design [16] [12]; lexical, vocabulary and data [6]; syntactic qualities [12].
- what is the purpose of the evaluation: to determine which ontology is more appropriate for a given domain, from a given set of ontologies[28]; to enhance an ontology definition during its construction phase [16].

Based on the already existing surveys and the ontology evaluation techniques classifications, the BPMO evaluation technique was chosen starting from the following premisses: (1) BPMO is still under development, so the evaluation should help us improve its design; (2) Most of the concepts and attributes names were taken from the existing business process modelling techniques, and they do not require a further evaluation; on the other hand, the ontology structure, architecture and design are new and innovative, and their evaluation is a must. The evaluation technique most appropriate for our objectives is OntoClean [16]. OntoClean is one of the most well-known and cited techniques for ontology evaluation¹.

2.1 BPMO Evaluation with OntoClean

This section provides the evaluation of BPMO using OntoClean. The first subsection presents a brief overview of OntoClean, for allowing the readers less familiar with this approach to understand the evaluation methodology; Section 2.1.2 provides the taxonomic structure of BPMO, needed during the evaluation. In Section (2.1.3) the actual ontology evaluation using OntoClean is performed.

2.1.1 OntoClean

OntoClean is a technique used for validating the ontological adequacy of taxonomic relationships [16]. If used during the design phase of an ontology, its results will guide towards possible improvements.

OntoClean has been described in several publications ([16][13][14][15]), and this deliverable is going to present only the minimum information needed for the evaluation. All the notations and definitions are taken from these publications. For more details please refer to the appropriate references.

The OntoClean evaluation is based on four parameters (metaproperties): Rigidity (**R**), Identity (**I**), Unity (**U**) and Dependence (**D**), having the following definitions:

Rigidity - a property is considered to be rigid if it is essential to all its possible instances; that is, an instance of a rigid property can not be an instance of another entity in a different world. In OntoClean a *property* is what it means to be a member of a class.

Identity - this parameter is referring to the problem of being able to recognize individual entities in the world as being the same. When a property carries its *Own* identity, the notation used is +O

¹see <http://esi-topics.com/erf/2004/june04-ChristopherWelty.html>

Unity - the property of classes all of whose individuals are wholes under the same relation (a class has unity if all its instances are the same type of whole, and is typically true of classes of natural objects).

Dependence -a property is dependent if each instance of it implies the existence of another entity.

Furthermore, all of these parameters can take three different values: positive (+), negative (-) or anti(\sim), which are explained in details in [16]. Identity and Dependency allow only the first two possibilities.

A set of subsumption constraints are further imposed, as presented in [16]. Considering two properties, p and q, where q subsumes p², the following constraints hold:

- C1: If q is anti-rigid (\sim), then p must be anti-rigid;
- C2: If q carries an identity criterion (+I or +O), then p must carry the same criterion;
- C3: If q carries a unity criterion (+U), then p must carry the same criterion;
- C4: If q has anti-unity (\sim U), then p must also have anti-unity.

If any of these four constraints does not hold for a given pair of properties p and q, the subsumption relation between them is semantically incorrect. Based on the parameters' values allocated for each class, some classes might move up or down in the class hierarchy, new classes may be added or existing classes might be removed, for correcting the modeling problems existing in an ontology [5].

²q subsumes p if and only if all instances of p are also instances of q

2.1.2 BPMO Taxonomy

The first step for evaluating an ontology using OntoClean method is to construct the taxonomic structure of that ontology. This structure will be further used for checking if the previously presented constraints hold.

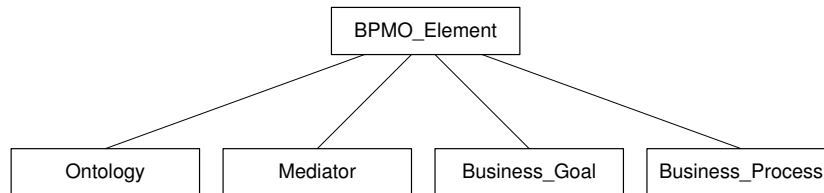


Figure 1: Main Elements - Taxonomy

As presented in Deliverable 2.1 [57] BPMO has one root element, `BPMO_Element`, which subsumes³ four main modeling elements: `Ontology`, `Mediator`, `Business_Goal` and `Business_Process` (see Figure 1).

Out of these four elements, the `Mediator` is out of the scope of this project, and we simply adopt the corresponding WSMO definition⁴. The `Business_Goal` is going to be defined in the next version of the BPMO, but we envision that its definition will contain a subset of the elements used in `Business_Process`'s definitions. Finally, the `Business_Process` and the process `Ontology` are the focus of the current BPMO, and they will be further analyzed in this section.

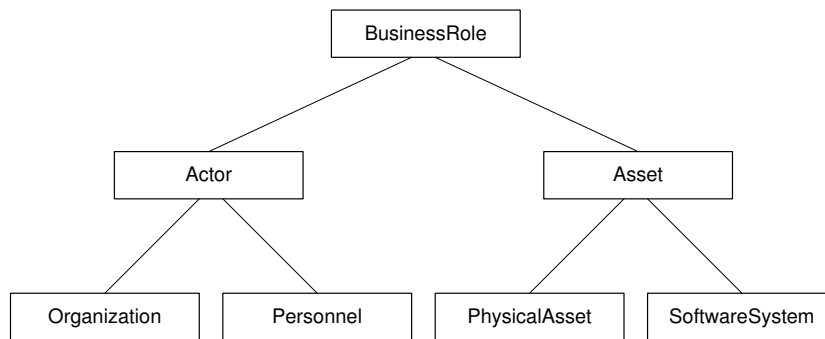


Figure 2: BusinessRole - Taxonomy

The BPMO Ontology has four main concepts: `BusinessRole`, `Event`, `Log` and `BusinessRule`. `Log` and `BusinessRule` are currently under-defined, they will be refined in the next BPMO version and evaluated in deliverable D5.2. The other two concepts, `BusinessRole` and `Event` have a number of subconcepts, structured in the taxonomies presented in Figures 2 and 3⁵.

³in this section the terminology used is the one from OntoClean description; in D1.2 the term *inherited (by)* was used instead of *subsumes*

⁴see: www.wsmo.org

⁵the rationale for creating all these subconcepts was presented in deliverable D1.2 [57]

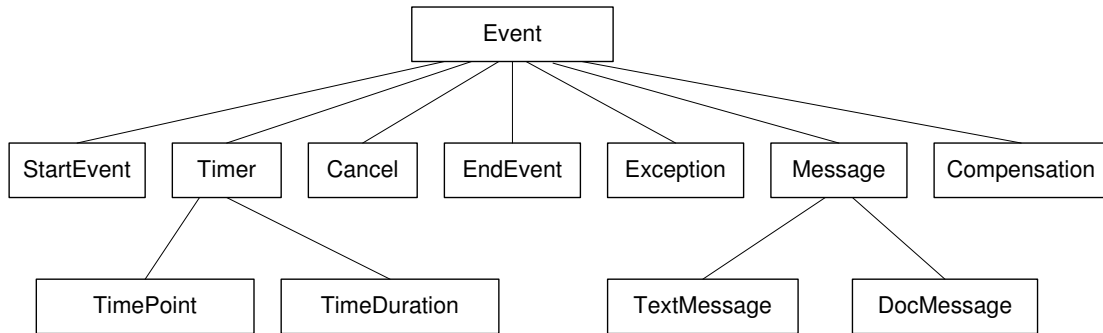


Figure 3: Event - Taxonomy

The `Business_Process` consists of five concepts, out of which only `BusinessProcess` and `CompositeProcess` have a number of subconcepts (see Figure 4) and will be further analyzed using OntoClean.

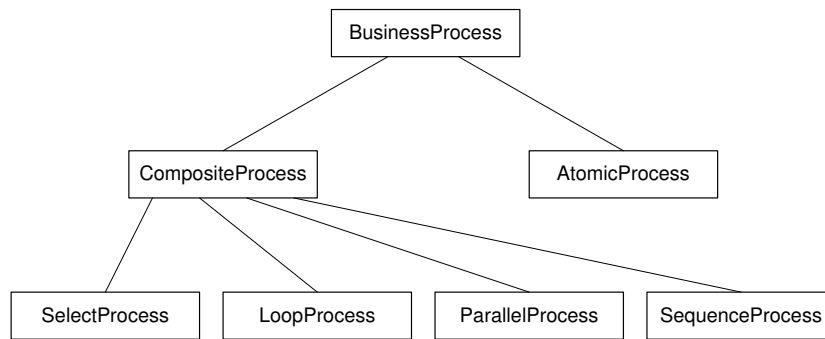


Figure 4: BusinessProcess- Taxonomy

2.1.3 Metaproperties Assignment and Evaluation

In this section the parameters' values (+, - or ~) will be assigned for the elements presented in the taxonomy, starting with the root.

BPMO_Element

Rigidity Everything is a `BPMO_Element`, so the Rigidity is Positive (+R).

Identity Every `BPMO_Element` carries its own identity criterion by using *namespaces* - this principle is inherited from `WSMO_Element` [10]. As a consequence, the parameter assignment is +O.

Unity This element is not carrying a common unity criterion, that is its instances are not the same type of wholes. The Unity parameter allocation is -U.

Dependence The existence of a `BPMO_Element` does not imply the existence of any other entity; as a consequence, this parameter allocation is -D.

Ontology

Rigidity Every instance of `Ontology` can only be an `Ontology` in every context (an ontology can not be considered to be a `Mediator`, `Business_Goal` or `Business_Process`); this parameter takes the value +R.

Identity +O (every `Ontology` has a namespace).

Unity This element is carrying a common unity criterion (every `Ontology` as a whole is an `Ontology`), the Unity parameter allocation is +U.

Dependence The existence of an `Ontology` does not imply the existence of any other entity; this parameter allocation is -D.

Mediator

Rigidity An instance of `Mediator` is not necessarily a `Mediator` in any other context (it can also be, for example a `Business_Process` providing a mediation solution); this parameter takes the value -R.

Identity +O (every `Mediator` has a namespace).

Unity Every `Mediator` as a whole is a `Mediator`, the Unity parameter allocation is +U.

Dependence The existence of a `Mediator` implies the existence of the entities it mediates between; this parameter takes the value +D.

Business_Goal

Rigidity Every instance of `Business_Goal` is necessarily a `Business_Goal` in every context, so the Rigidity is +R.

Identity +O (every `Business_Goal` has a namespace).

Unity Every `Business_Goal` as a whole is a `Business_Goal`, the Unity parameter allocation is +U.

Dependence The existence of a `Business_Goal` does not imply the existence of any other entity; this parameter allocation is -D.

Business_Process

Rigidity Every instance of `Business_Process` is necessarily a `Business_Process` in every context, so the Rigidity is +R.

Identity +O (every `Business_Process` has a namespace)

Unity Every `Business_Process` as a whole is a `Business_Process`, the Unity parameter allocation is +U.

Dependence The existence of a `Business_Process` does not imply the existence of any other entity; this parameter allocation is -D.

As shown in Figure 5 the constraints presented in Section 2.1.1 hold for this taxonomy.

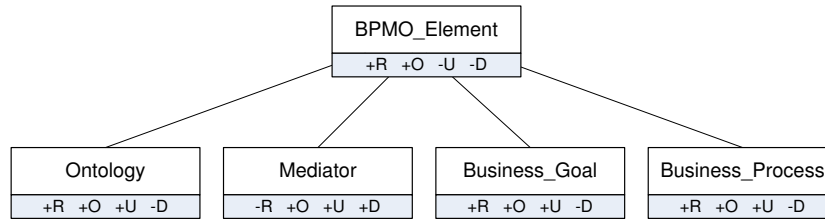


Figure 5: Metaproperties Allocation for the Main Elements

BusinessRole

Rigidity `BusinessRole` is rigid (+R), every instance of a `BusinessRole` is necessarily a `BusinessRole` in every context.

Identity Inside the ontology every element is identified by a unique id; outside the ontology, every element is identified by the concatenation of the ontology id and its own. As a consequence, this parameter takes the value +O.

Unity This element is not carrying a common unity criterion, that is its instances are not the same type of wholes. The Unity parameter allocation is -U.

Dependence -D. The existence of a `BusinessRole` does not imply the existence of any other entity.

Actor

Rigidity +R. Every instance of `Actor` is necessarily an `Actor` in every context.

Identity +O.

Unity This element is not carrying a common unity criterion, that is its instances are not the same type of wholes. The Unity parameter allocation is -U.

Dependence -D. The existence of an `Actor` does not imply the existence of any other entity.

Organization

Rigidity +R. Every instance of `Organization` is necessarily an `Organization` in every context.

Identity +O.

Unity +U. Every `Organization` as a whole is an `Organization`.

Dependence -D. The existence of an `Organization` does not imply the existence of any other entity.

Personnel

Rigidity +R. Every instance of `Personnel` is necessarily a `Personnel` in every context.

Identity +O.

Unity +U. Every `Personnel` as a whole is a `Personnel`.

Dependence -D. The existence of an instance of `Personnel` does not imply the existence of any other entity. In the next version of BPMO, a cardinality constraint will be included specifying that for every `Personnel` the `Organization` it belongs to has to be specified. This will change the Dependency to +D. As there are no constraints regarding the Dependency in OntoClean, this modification will not affect the evaluation results.

Asset

Rigidity +R. Every instance of `Asset` is necessarily an `Asset` in every context.

Identity +O.

Unity This element is not carrying a common unity criterion, that is its instances are not the same type of wholes. The Unity parameter allocation is -U.

Dependence -D. The existence of an `Asset` does not imply the existence of any other entity. The next version of BPMO will impose a cardinality constraint: the `Asset` must belong to an `Actor`, thus the parameter will change to +D.

PhysicalAsset

Rigidity +R. Every instance of `PhysicalAsset` is necessarily a `PhysicalAsset` in every context.

Identity +O.

Unity +U. Every `PhysicalAsset` as a whole is a `PhysicalAsset`.

Dependence -D. The existence of a `PhysicalAsset` does not imply the existence of any other entity. The next version of BPMO will impose a cardinality constraint: the `PhysicalAsset` must belong to an `Actor`, thus the parameter will change to +D.

SoftwareSystem

Rigidity +R. Every instance of **SoftwareSystem** is necessarily a **SoftwareSystem** in every context.

Identity +O.

Unity +U. Every **SoftwareSystem** as a whole is a **SoftwareSystem**.

Dependence -D. The existence of a **SoftwareSystem** does not imply the existence of any other entity. The next version of BP MO will impose a cardinality constraint: the **SoftwareSystem** must belong to an **Actor**, thus the parameter will change to +D.

As presented in Figure 6 the constraints presented in Section 2.1.1 hold for **BusinessRole** taxonomy.

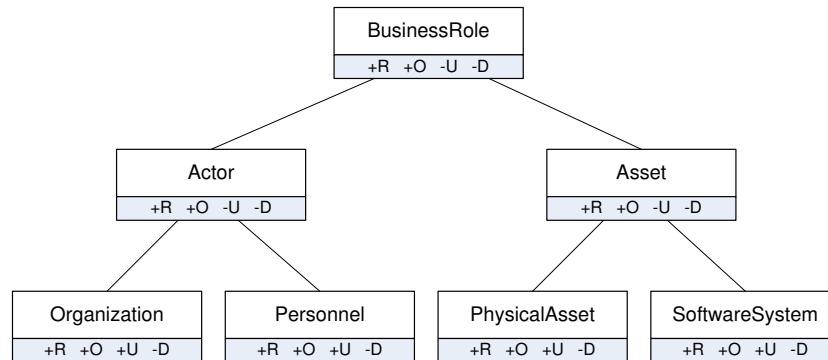


Figure 6: Metaproperties Allocation for the **BusinessRole** Taxonomy

Event

Rigidity -R. An instance of **Event** is not necessarily an **Event** in any other context.

Identity +O.

Unity -U. This element is not carrying a common unity criterion, that is its instances are not the same type of wholes.

Dependence -D. The existence of an **Event** does not imply the existence of any other entity.

StartEvent

Rigidity -R. The instance of **StartEvent** may play a different role in a different context.

Identity +O.

Unity +U. Every **StartEvent** as a whole is a **StartEvent**.

Dependence -D. The existence of a **StartEvent** does not imply the existence of any other entity.

Timer

Rigidity +R. Every instance of **Timer** is necessarily a **Timer** in every context.

Identity +O.

Unity -U. This element is not carrying a common unity criterion, that is its instances are not the same type of wholes.

Dependence -D. The existence of a **Timer** does not imply the existence of any other entity.

TimePoint

Rigidity +R. Every instance of **TimePoint** is necessarily a **TimePoint** in every context.

Identity +O.

Unity +U. Every **TimePoint** as a whole is a **TimePoint**.

Dependence -D. The existence of an **TimePoint** does not imply the existence of any other entity.

TimeDuration

Rigidity +R. Every instance of **TimeDuration** is necessarily a **TimeDuration** in every context.

Identity +O.

Unity +U. Every **TimeDuration** as a whole is a **TimeDuration**.

Dependence -D. The existence of an **TimeDuration** does not imply the existence of any other entity.

Cancel

Rigidity -R. The instance of **Cancel** may play a different role in a different context.

Identity +O.

Unity +U. Every instance of **Cancel** as a whole is the same.

Dependence -D. The existence of `Cancel` does not imply the existence of any other entity. In the next BPMO version, `Cancel` may specify the exact process it is referring to, which will modify this parameter value to +D.

EndEvent

Rigidity -R. The instance of `EndEvent` may play a different role in a different context.

Identity +O.

Unity +U. Every instance of `EndEvent` as a whole is the same.

Dependence -D. The existence of an `EndEvent` does not imply the existence of any other entity.

Exception

Rigidity -R. The instance of `Exception` may play a different role in a different context.

Identity +O.

Unity +U. Every instance of `Exception` as a whole is the same.

Dependence -D. The existence of an `Exception` does not imply the existence of any other entity.

Message

Rigidity +R. Instances of `Message` can only be `Messages` in any context.

Identity +O.

Unity -U. This element is not carrying a common unity criterion, that is its instances are not the same type of wholes.

Dependence -D. The existence of a `Message` does not imply the existence of any other entity.

TextMessage

Rigidity +R. Every instance of `TextMessage` is necessarily a `TextMessage` in every context.

Identity +O.

Unity +U. Every `TextMessage` as a whole is a `TextMessage`.

Dependence -D. The existence of a `TextMessage` does not imply the existence of any other entity.

DocMessage

Rigidity +R. Every instance of `DocMessage` is necessarily a `DocMessage` in every context.

Identity +O.

Unity +U. Every `DocMessage` as a whole is a `DocMessage`.

Dependence -D. The existence of a `DocMessage` does not imply the existence of any other entity.

Compensation

Rigidity +R. Every instance of `Compensation` is necessarily a `Compensation` in every context.

Identity +O.

Unity +U. Every instance of `Compensation` as a whole is a `Compensation`.

Dependence -D. The existence of an instance of `Compensation` does not imply the existence of any other entity. In the next version of BPMP every instance of this concept will have to contain information regarding the process for which it provides compensation.

As presented in Figure 7 the constraints presented in Section 2.1.1 hold for `Event` taxonomy.

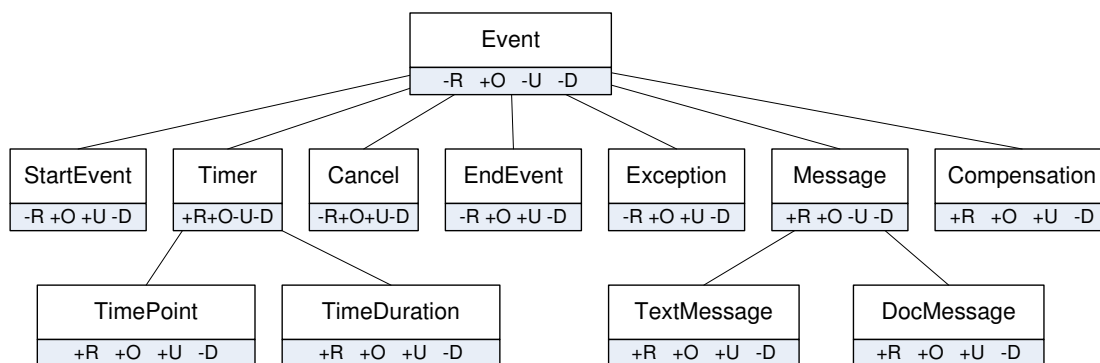


Figure 7: Metaproperties Allocation for the Event Taxonomy

BusinessProcess

Rigidity -R. The instance of `BusinessProcess` may play a different role in a different context.

Identity +O (from the same reason as for the previous elements).

Unity +U. Every instance of **BusinessProcess** as a whole is a **BusinessProcess**.

Dependence -D. The existence of an instance of **BusinessProcess** does not imply the existence of any other entity.

CompositeProcess

Rigidity -R. Instances of **CompositeProcess**s may play a different roles in different contexts.

Identity +O.

Unity +U. Every instance of **CompositeProcess**s as a whole is a **CompositeProcess**s.

Dependence +D. The existence of a **CompositeProcess** implies the existence of the composed processes (to the very least, the existence of the instance **Null** of the **AtomicProcess**).

SelectProcess

Rigidity +R. Every instance of **SelectProcess** is necessarily a **SelectProcess** in every context.

Identity +O.

Unity +U. Every instance of a **SelectProcess**s as a whole is a **SelectProcess**s.

Dependence +D. The **SelectProcess** automatically inherits the constraints from the **CompositeProcess**.

LoopProcess

Rigidity +R. Every instance of **LoopProcess** is necessarily a **LoopProcess** in every context.

Identity +O.

Unity +U. Every instance of a **LoopProcess**s as a whole is a **LoopProcess**s.

Dependence +D. The **LoopProcess** automatically inherits the constraints from the **LoopProcess**.

ParallelProcess

Rigidity +R. Every instance of **ParallelProcess** is necessarily a **ParallelProcess** in every context.

Identity +O.

Unity +U. Every instance of a `ParallelProcess` as a whole is a `ParallelProcess`.

Dependence +D. The `ParallelProcess` automatically inherits the constraints from the `ParallelProcess`.

SequenceProcess

Rigidity +R. Every instance of `SequenceProcess` is necessarily a `SequenceProcess` in every context.

Identity +O.

Unity +U. Every instance of a `SequenceProcess` as a whole is a `SequenceProcess`.

Dependence +D. The `SequenceProcess` automatically inherits the constraints from the `ParallelProcess`.

AtomicProcess

Rigidity +R. Every instance of `AtomicProcess` is necessarily a `AtomicProcess` in every context.

Identity +O.

Unity +U. Every instance of a `AtomicProcess` as a whole is a `AtomicProcess`.

Dependence -D. The existence of an instance of `AtomicProcess` does not imply the existence of any other entity.

As in the previous cases, this taxonomy also does not contain any inconsistencies based on OntoClean evaluation (see Figure 8).

2.2 Comparison with other Frameworks

In this section two of the most well-known ontologies for business process modeling will be analyzed, in comparison with BPMO: OWL-S⁶ Process Ontology and SUPER⁷ BPMO.

2.2.1 OWL-S Process Ontology

OWL-S is a OWL-based Web service ontology which provides a set of constructs for describing the Web Services. In [29] the authors propose a way for modeling the services as processes. The top level process ontology is depicted in Figure 9

A main difference between the two approaches is the fact that BPMO makes a clear distinction between what a process does (**capability**) and its **execution**, while

⁶<http://www.daml.org/services/owl-s/1.1/>

⁷<http://www.ip-super.org/>

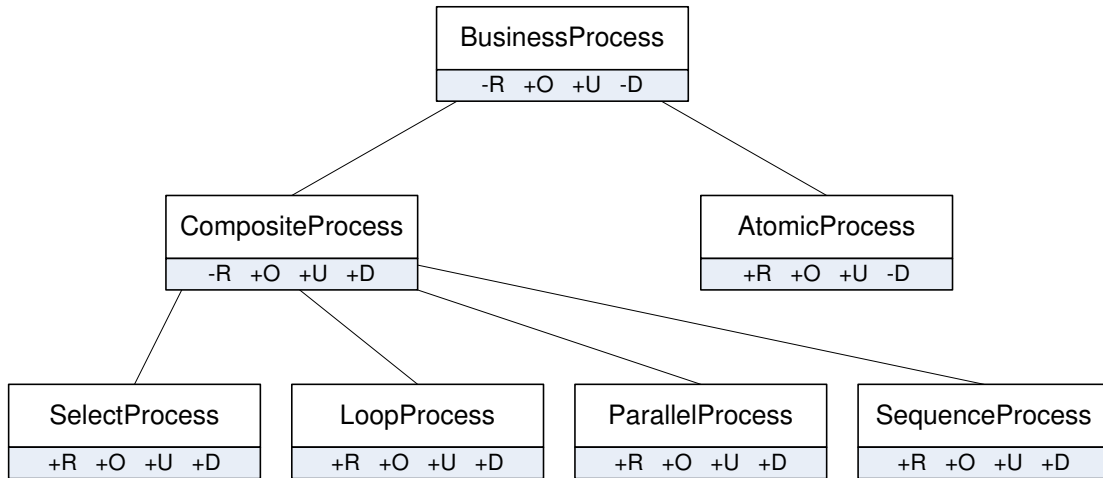


Figure 8: Metaproperties Allocation for the BusinessProcess Taxonomy

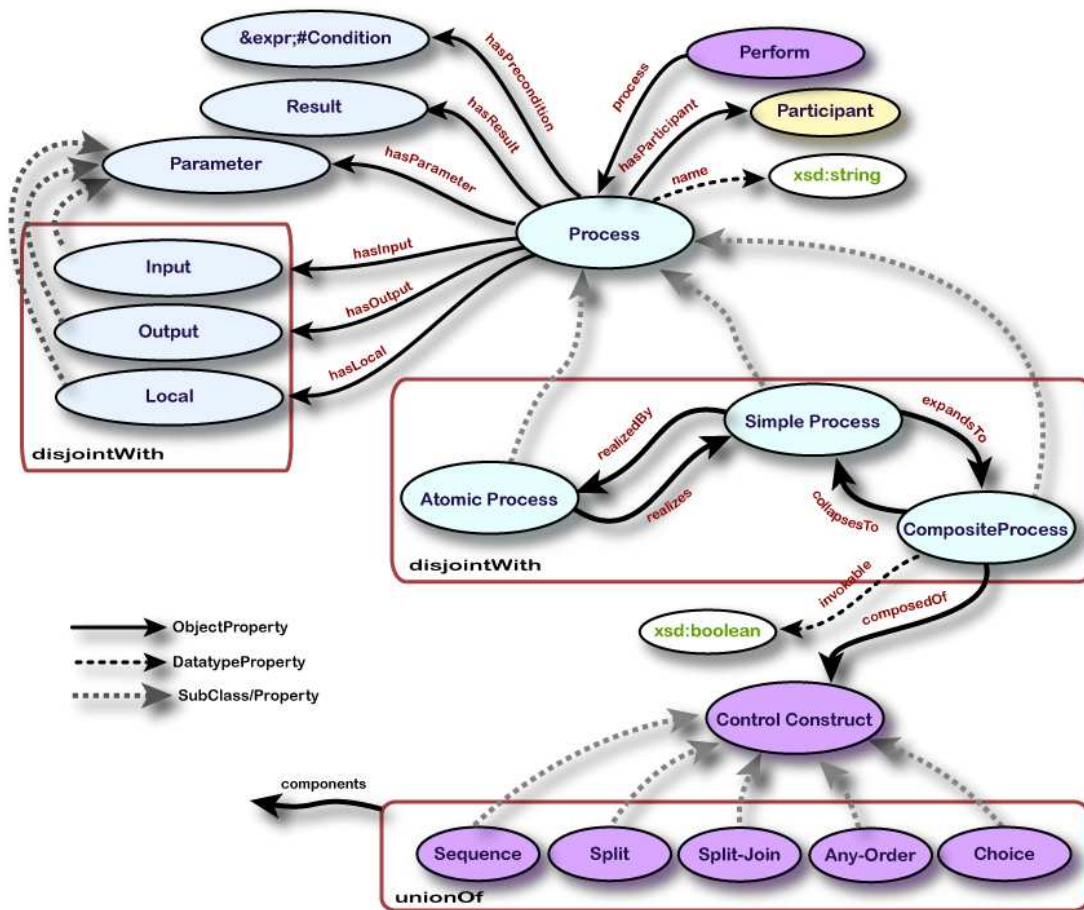


Figure 9: OWL-S Process Ontology(from [29])

in OWL-S this distinction is not done. The distinction is important from reusability perspective: several processes can have the same `capability`, but different execution paths.

An OWL-S process is characterized by the following attributes: `hasInput` (with the range of type `Input`), `hasOutput` (`Output`), `hasPrecondition` (`Condition`), `hasPostcondition` (`Condition`), `hasParticipants` (`Participant`), `hasResult` (`Result`) and `Local` (`Local`).

The first two parameters are part of the current version of BPMO `Business_Process` where the attributes have as range an `Event`, which triggers or is the result of executing a process. The next two parameters `precondition` and `postcondition` are going to be introduced in the second version of BPMO, being important for the discovery process. This four attributes are part of the specification of any Semantic Web Service or semantically described process and are usually refer to as IOPEs Inputs, Outputs, Preconditions, Effects. In BPMO these parameters are part of the `Capability` description.

The `hasParticipants` parameter also has direct correspondence in BPMO, this time in the process execution. In BPMO, we distinguish between two types of participants: the executor (given by the attribute `executedBy`) and participants (`interactsWith`). In OWL the participants are limited to the client and the server, while in BPMO we do not make this assumption. Any `BusinessRole` can participate in the execution of the process.

The last two parameters, `Local` and `Result` are used to bind the preconditions and to describe the outputs and effects associated with these conditions. In BPMO we consider that the preconditions and postcondition should be enough for this.

Regarding the process hierarchy, OWL-S introduces three sub-classes: `atomic`, `simple` and `composite processes`. The atomic and composite processes have exactly the same semantics as in SemBiz BPMO, while the simple processes are considered to be not invocable and not associated with a grounding. This distinction is not needed in BPMO where the `hasGrounding` attribute has the cardinality (0).

The `Control Constructs` from OWL-S can be considered the equivalent of the BPMO composite process types.

As a conclusion the BPMO `Business_Process` can, even on this intermediate stage, express every aspect on the OWL-S processes (with the exception of pre and postcondition, which are part of our planned future work) and also provides a clear execution vs capability separation. In addition, the BPMO provides an underlying ontology for specifying any process related aspects, like business roles, events or compensation.

2.2.2 SUPER BPMO

The SUPER approach for designing BPMO started with the premise that most of the existing standards and methodologies for process design can be modeled using ontologies, and can be after that abstracted to a higher level ontology, BPMO. As a consequence, several ontologies emerged for semantically representing EPC (sEPC) [50], BPEL (BPEL Ontology) [35] and BPMN. Specific ontologies are also used for modeling events (based on MXML) and for diverse knowledge required for mining Semantic Business Processes [4]. Any of this ontologies may be imported and reused by the SemBiz BPMO if needed.

The conclusion drawn from that comparison is that BPMO is at least as expressive and complete as OWL-S process model.

A more detailed analysis and evaluation of BPMO will be done after the ontology is finished, and it will be presented in deliverable 5.2 Evaluation case study implementation.

3 Evaluation of Approaches

Within this section we will evaluate our approaches for semantic query, discovery, functional and syntactic composition and deployment and execution by comparing them with related work.

3.1 Query and Discovery

For both semantic query and discovery, the main effort is to adapt existing WSMO solutions to BPMO. Within this section, we will therefore focus on approaches that can be straightforwardly adapted to BPMO. Our interest here is not to provide an extensive comparison with state of the art reasoners, respectively discovery engines, but rather to motivate our choices.

3.1.1 Semantic Query

The SemBiz approach to semantic query is already introduced in [9]: we use the WSML2Reasoner framework, with the Integrated Rule Inference System (IRIS)⁸ as underlying Datalog engine. The main motivation is that WSML2Reasoner and IRIS support query answering for WSML-Core and WSML-Flight, and reasoning support for WSML-Flight in one of the main requirements, since BPMO is defined as a WSML-Flight ontology.

WSML2Reasoner is a flexible architecture for easy integration of external reasoning components. It translates ontologies described in WSML to the syntax of the underlying reasoning engines. KAON2, Pellet, MINS, IRIS are reasoners that have been integrated with WSML2Reasoner. In the following, we give briefly introduce each of these tools.

Pellet⁹ is a sound and complete DL reasoner for *SHOIN(D)* Description Logics. This Java-based reasoner provides an option to enable the Unique Name Assumption (UNA) and includes reasoning about nominals. It is based on the tableaux algorithms developed for expressive Description Logics and incorporates the decision procedure for *SHOIQ* (the expressivity of OWL-DL plus qualified cardinality restrictions). More details about Pellet's features, its architecture and implemented optimization techniques, as well as its future directions can be found in [45]. WSML2Reasoner and Pellet support reasoning tasks for WSML-DL. However, this combination does not offer support for the Logic Programming branch of WSML, and in particular for the language of BPMO, WSML-Flight.

⁸<http://iris-reasoner.org/>

⁹<http://www.mindswap.org/2003/pellet/index.shtml>

KAON2 ¹⁰ is an infrastructure for managing OWL-DL, SWRL and F-Logic ontologies. It provides a hybrid reasoner that allows Datalog-style rules to interact with structural Description Logics knowledge bases. It supports the *SHIQ(D)* Description Logic and disjunctive Datalog programs. Unlike most currently available DL reasoners, the Java-based KAON2 reasoner does not implement the tableaux calculus. It implements algorithms which reduce a *SHIQ(D)* knowledge base to a disjunctive datalog program. These algorithms allow applying well-known deductive database techniques, such as magic sets or join-order optimizations, to DL reasoning. The theoretical work underlying KAON2 can be found in [52]. Although reasoning with WSML-Flight is possible when using WSML2Reasoner with KAON2, one major drawback is the licensing: KAON2 is not open source, and its usage is free only for academic purposes.

MINS (Mins Is Not Silri) ¹¹ is a reasoner for Datalog programs with negation and function symbols, based on the SILRI ¹² inference engine. One drawback of using MINS is its license (in this case GPL). Another issue is that there is currently no active support for MINS.

Our chosen inference engine, IRIS, is designed to meet requirements for large scale reasoning. Apart from the state-of-the-art deductive methods, the system utilizes database techniques and extends them for implicit knowledge in order to effectively process large datasets. Currently, IRIS is a WSML-Flight reasoner. However, the system is extensively being developed to support reasoning with WSML-Rule and to integrate a permanent storage system designed for distributed scalable reasoning. Further, IRIS will implement novel techniques for reasoning with integrating frameworks based on classical first-order logic and non-monotonic logic programming as well as techniques for Description Logics reasoning.

3.1.2 Semantic Discovery

Our approach to semantic discovery is already introduced in [9]: we adapt the existing WSMO discovery [24] to BPMO. The main reason is that the adaptation is straightforward, since BPMO business processes and business goals have their capabilities described semantically, in a very similar way to WSMO Web services and goals. Another reason is that the WSMO discovery combines a pure keyword-based discovery, which serves as a pre-filtering step, with semantic discovery.

From the implementation point of view, we adapt the WSMX discovery implementation. The current WSMX implementation does not contain the most general type of WSMO discovery, namely discovery based on rich semantic descriptions. However, it has some advantages over the existing solutions. The most relevant in the context of SemBiz is that it contains several implementations for semantic discovery based on simple descriptions of web services (or "lightweight" discovery), for WSML-DL as well as for WSML-Flight semantic descriptions.

A brief summary of the existing literature on semantic discovery of web services is as follows. There exist various approaches to web service discovery which consider only DL

¹⁰<http://kaon2.semanticweb.org>

¹¹<http://dev1.deri.at/mins/>

¹²<http://ontobroker.semanticweb.org/silri/>

descriptions, the solutions being often tailored for multi-agent systems ([39, 49, 27]). Most of the existing literature in this field refers to detecting matches by comparing the inputs and outputs of requested, respectively provided, web services. For example, the matching algorithms described in [39] and [27] depend on the logical relation between the concepts associated with those inputs and outputs.

Only few approaches take into consideration in the matching process also the relationship between inputs and outputs. From these, the largest part deal specifically with web services expressed using Description Logics. One relevant example in this direction is [19], which presents a means to compare service descriptions defined in terms of inputs, outputs and conjunctive queries. These conjunctive queries are ABox assertions that relate the inputs and outputs with known objects (individual names) and existentially quantified unknown objects.

In comparison, WSMO discovery does not focus on a particular language, but on an adequate (language-independent) mathematical model of the objects of investigation. It is therefore compatible with various concrete representation languages such as Description Logics, First-order Logics, etc.

3.2 Functional Composition

Functional level composition plays an important role in the achievement of (semi)automatic composition of business processes. Namely, it serves to identify an appropriate (partially ordered) list of BPMO business processes from the possibly very large number of BPMO business processes returned by the discovery component. Composition at this level gives the guarantee that the selected BPMO business processes can achieve the user goal from a functional perspective. Therefore, functional composition does not result in an executable process. Rather, it forms an effective filtering method for more accurate techniques that take into account the exact protocols by which the business processes communicate.

Because functional composition is performed at the semantic level, one crucial element is to provide support for background ontologies. These ontologies model the underlying domain of BPMO business processes. Reasoning over ontologies is necessary in two situations: (1) to understand which BPMO business processes can be used, and (2) to test whether a given composition is actually a solution. Most of the existing approaches on functional composition ignore the background ontologies and assume exact matching of input/output types. In contrast, our notion of functional composition is very general: the composition semantics includes powerful background ontologies, and we use the most general notion of matching, *partial matches*, where several business processes can cooperate each covering only a part of a requirement.

3.2.1 Comparing to Related Work

The SemBiz approach to functional composition naturally builds on the work presented in [18]. The latter already introduces the formal basis of this approach, which includes the very general notion of composition considered here, as well as the special case of forward effects. However, [18] focuses on a restriction of the forward effects case, which

can be compiled into AI Planning formalism, and therefore be solved by already existing tools for planning under uncertainty.

In this work, we address the forward effects special case. In difference to [18], we do not solve the composition problem by compiling it into a planning formalism. Instead, we have developed a dedicated tool, by adapting to functional composition the underlying techniques of a scalable tool for planning under uncertainty. By solving the problem in its natural form we expect to obtain a more efficient tool, optimized for problems that are composition-specific. One example of such problems is that, in comparison to the planning tasks which involve a relatively small number of predefined actions, functional composition has to deal with a large number of BPMO business processes, this having a direct effect on the branching factor of the actual search.

There exist many other approaches to functional composition (e.g., [42, 44]) that compile the composition problem into a planning formalism. These approaches also assume exact matches of input/output types, therefore ignoring the background ontologies. In contrast, we use the most general notion of matching, *partial matches*. In effect, the fact that we allow partial matches differentiates our work from almost all other approaches to composition.

Some works focus on various other aspects of the composition problem, e.g.: [34] obtains a simple composition ability as a side-effect of verifying SWS properties using Petri Nets; [30] treats the actual interaction (communication) with a web service as a planning problem; [26, 2] focus on information gathering at composition time (rather than at plan execution time).

Two approaches explore how to adapt formalisms from hand-tailored planning, namely Golog [31], respectively HTN planning [47], for web service composition. Both approaches are capable of composing services involving control constructs (loops, branches). There is one fully automatic approach handling control constructs [41, 40]. There, search techniques based on BDDs (Binary Decision Diagrams) are exploited to obtain complex solutions; input/output type matches are assumed to be exact.

The requirements on matches are relaxed in the following works. In [1], techniques are introduced that disambiguate concept names during web service composition.

[46] extends the classical HTN (Hierarchical Task Network) planning to work with OWL-S processes, and therefore performs matching with respect to OWL-DL ontologies. However, the type of matches considered is plug-in, i.e. a task will match only if all its preconditions are fulfilled. This approach further differs from ours in that, similar to [31, 47], it uses workflow templates, describing the outline of activities that need to be performed, to achieve composition.

Partial matches are addressed in [8], a fully automatic approach to web service composition. The approach uses numeric intervals to encode a sub-concept hierarchy, and then performs matching (as well as service discovery) based on those intervals, simply by asking whether one interval (a service output) is fully contained in the union of a set of intervals (service inputs). Both forward and backward search are performed (however, for backward search only complete matches are considered). There are several differences to our approach. First, in [8] the functionality of web services is described only with typed inputs and outputs, without preconditions and effects. The main consequence is that it is not possible to encode the relation of outputs with respect to inputs, feature

present in the forward effects case. Second, our background theories are more general, taking arbitrary clausal form instead of a sub-concept hierarchy. Third, our next steps will be to design heuristic functions that guide the search, instead of performing a blind depth-first search.

3.2.2 Conclusions

Many of the existing approaches to functional composition assume a very restrictive matching of input/output types. To the best of our knowledge, partial matches are considered only in [8]. We emphasize again the importance of partial matches: consider a BPMO business process that outputs an instance of concept Event. According to BPMO, Event has several sub-concepts, for example: Message, Timer, Compensation etc. Since the BPMO business process does not specify the exact sub-category, applying it results in an ambiguous situation. Such situations are very likely to appear in an open, or at least large, environment.

In [9] we have introduced our formalism for functional composition with partial matches, and identified the special case of forward effects. The latter has a simpler semantics, while still covering many composition scenarios from literature and real case studies. We have also presented the details of adapting a tool for planning under uncertainty to the forward effects case. The tool, namely Conformant-FF [8], is based on CNF reasoning. We were able to naturally adapt this approach to our setting because forward effects beliefs can also be represented as propositional CNFs. Note that this is not possible in the general WSC case, for complexity reasons. The result of our work so far is the basis of a tool capable to automatically compose BPMO business processes that conform to the forward effects restrictions. The main challenge up to this point has been to define the search space of the tool, and optimize the basic procedures to the extent that they are sufficiently efficient.

Our goal is to achieve scalability of functional composition with partial matches. This is also the problem in [8], which performs a blind search in the space of possible compositions. Therefore, while we already have a prototype able to solve forward effects tasks, a lot of effort is still necessary in order to make the approach scalable. Our future work will be to control the search via heuristic solution distance and search node filtering techniques, and to implement and evaluate these techniques.

3.3 Syntactic Composition

Service-oriented computing is an emerging paradigm that made an important shift from traditional tightly coupled, hard-to-adapt software development to more platform neutral, loosely coupled software development. The interoperable and platform independent nature of services supports a novel approach to business process development by using processes, running in a process engine, to invoke existing services from their process activities (aka process tasks, steps). We call this kind of architecture *process-driven, service-oriented architecture* [17]. In this approach, a typical business process consists of many activities, the control flow, and the process data. Each activity is correspondent to a communication task (e.g., invoking other services, processes, or an interaction with

a human), or a data processing task. The control flow describes how these activities are ordered and coordinated to achieve the business goals. Being well considered in both research and industry, this approach has led to a number of standardisation efforts, such as WS-BPEL [36] [20], XPDL [56], BPMN [37], and WS-CDL [55].

As the number of services or processes involved in a business process grows, the complexity of developing and maintaining the business processes also increases along with the number of invocations and data exchanges. It is error-prone and time consuming for developers to work with large business processes that implement numerous concerns, such as the process control flow, the participants involved, the data dependencies and messages exchanged, the service invocations, etc. This problem occurs because business process descriptions integrate these concerns in a compound way without a clear separation. In addition, this problem also occurs at different abstraction levels [17]. For instance, the business process is relevant for different stakeholders: Business experts require a high-level business-oriented understanding of the various process elements (e.g., the relations of processes and activities to business goals and organisation units), whereas the technical experts require the technical details (e.g., deployment information or communication protocol details for service invocations).

In addition to this complexity, business experts and technical experts alike have to deal with a constant need for change. On the one hand, process-driven SOA aims at supporting business agility. That is, the process models should enable a quicker reaction on business changes in the IT by manipulating business process models instead of code. On the other hand, the technical infrastructure (technologies, platforms, etc.) constantly evolves.

One of the successful approaches to manage complexity is *separation of concerns* [11]. Process-driven SOAs use a specific realisation of this principle, *modularisation* [11]: Services expose standard interfaces to processes and hide unnecessary details for using or reusing. This helps in reducing the complexity of process-driven SOA models, but from the modellers' point of view this is often not enough to cope with the complexity challenges explained above, because modularisation only exhibits a single perspective of the system focusing on its (de-)composition. Other – more problem-oriented – perspectives, such as a business-oriented perspective or a technical perspective (used as an example above), are not exhibited to the modeller. In the field of software architecture, *architectural views* have been proposed as a solution to this problem. An *architectural view* is a representation of a system from the perspective of a related set of *concerns* [21]. The architectural view concept offers a separation of concerns that has the potential to resolve the complexity challenges in process-driven SOAs, because it offers more tailored perspectives on a system, but it has not yet been exploited in process modelling languages or tools.

In [9], we introduced a view-based framework that (semi-)formally defines various concerns of the process model and uses those (semi-)formalised models to capture a particular perspective of the business process.

3.3.1 Comparing to Related Work

Our work regarding syntactic composition is closely related to existing process modelling languages. There are several standardisation efforts for process modelling languages, such as WS-BPEL [36], BPMN [37], XPDL [56], WSCI [54] and WS-CDL [55]. They can be categorised into different dimensions, for instance, textual and graphical languages, or abstract and executable languages, and so on. The abstract modelling languages (e.g., abstract BPEL, or WSCI/WS-CDL) are working at the same abstraction level as our abstract models (i.e., orchestration, information, or collaboration models) while the executable language are more or less similar to our refined models. The aforementioned modelling languages consider the business process model as a whole. They do not support the separation of the process model's concerns. Moreover, there is no explicit relationship between an abstract and an executable modelling language. So it requires additional effort to maintain the integrity and consistency of the models, or to validate models [32, 38]. All these modelling languages can be integrated into our approach using extension models.

To the best of our knowledge, there is only a few view-based approaches to business process modelling. The most related work in this area is the approach by Mendling et al. [33] inspired by the idea of schema integration in database design. Process models based on Event-driven Process Chains (EPCs) are investigated, and the pre-defined semantic relationships between model elements such as *equivalent*, *sequence*, and merge operations are performed to integrate two distinct views. Semantics-based merging is a promising approach to model integration, but it is difficult to apply to integrate two different types of models, for instance, to merge a control model with a data model. Thus, the authors mainly focus on integrating process models without any data element or any collaboration.

The Amfibia [3, 25] approach focuses on formalising different aspects of business process modelling, and/or develop an open framework to integrate various modelling formalisms through the *interface* concept. Akin to our approach, Amfibia has the main idea of providing a modelling framework that does not depend on a particular existing formalism or methodology. The major contribution in Amfibia is to exploit dynamic interaction of those aspects. Like our approach, Amfibia's framework also has a core model with a small number of important elements, which are referred to, or refined in other models. The distinct point to our framework is that in Amfibia the interaction of different 'aspects' is only performed by event synchronisation at run-time when the workflow management system executes the process. Using extension and integration mechanisms in our framework, the integrity and consistency between models can be verified earlier at the model level.

The ISO Reference Model for Open Distributed Processing (RM-ODP) [22] is a standardised reference model, which defines a set of different *view points* such as enterprise, information, computational, engineering, and technology viewpoints. Each viewpoints has its own language and clear semantics. The consistency among viewpoints is ensured by the common architecture and the *common object model*. These concepts, similar to those in Amfibia and our approach, are defined based on the principle of separation of concerns to help stake-holders thinking from different perspectives in order to manage complexity of distributed applications. The advantage of our approach compared

to these approaches is that our view-based model-driven framework does not only separate process model concerns but also separate process model into different levels of abstraction, for instance, business level, and technical level.

Our work also shares some concepts with the approach described in [53]. van der Aalst et al. develop a conceptual SOA-based architecture framework around the idea of modularisation. The key concept in [53] is the *component* that is more or less equivalent to our *process* concept, and the relationships between components. The authors emphasise the separation of activities from data elements, but do not mention the capability of extending or integrating other concerns that could be part of a business process.

Skogan et al. [48] offer another approach for process-based modelling in UML. A tool-chain is devised to extract and formalise WSDL descriptions using UML models. Service compositions are captured by UML activity diagrams with special stereotypes. Finally, code in executable languages is generated from a composition model. The authors neither consider separation of concerns in service composition nor integration of other concerns except service interfaces and the control flow.

Schmidt et al. [43] proposes an interesting approach to web service transaction modelling. Even though the approach is only considering one concern of a business process model, the view based modelling framework also realises the separation of views into layers and maintaining references between various layers. Our work has not yet focused on other concerns, such as transactions, security, etc., but our model-driven framework in general can be extended into these dimension using the approach presented in [9]. Consequently, the transaction model in [43] can be seen as a complement to our work to develop the meta-model for the transaction concerns of the business process.

3.3.2 Conclusions

Existing modelling approaches lack sufficient support to manage the complexity of developing large business processes with many different concerns because most of them consider the process model as a whole.

The view-based modelling framework [51] not only helps to manage the development complexity by the separation of the processes' concerns, but also to cope with both business and technical changes using the separation of abstraction levels.

This study also raises a number of research questions which are only answered by further work. The modelling framework should be extended with other concerns of the business process such as transactions, security, event handling, etc. In addition, the view integration algorithms can be enhanced by the validation of possible constraint conflicts between various integration points.

3.4 Deployment & Execution

As the evaluation of different BPEL engines, their performance, their degree of compliance and ease of use, would not be a significant contribution to the project we have designed a framework that permits generic validation, deployment and execution (VDE) using an arbitrary BPEL engine. This way *flexibility* concerning the BPEL engine of

choice is given and a unique interface for validation, deployment and execution can be used. The VDE framework will be presented in [7].

4 Conclusion

Within this deliverable we have evaluated the BPMO as well as techniques for querying, discovery, and composition of business processes and compared it with other approaches and related work.

The OntoClean methodology proved the correctness of BPMO from the hierarchical perspective (*subsumes* relationship), but it does not provide any results regarding the expressivity or the completeness of the ontology. The comparison with the related work section analyze these two aspects by comparing BPMO with two existing ontologies. The conclusion drawn from that comparison is that BPMO is at least as expressive and complete as OWL-S process model.

For semantic query and discovery, we have provided a brief summary of existing approaches, and motivated our choice: tools that can be straightforwardly adapted to BPMO.

Concerning functional composition of semantic descriptions, we have underlined the main features of our approach, which we then used as a basis for comparison with existing compositions solutions.

We compared a view based modelling framework for syntactic composition with related work that we consider appropriate for meeting the projects requirements and goals. An abstracting layer for validation, deployment and execution of BPEL processes will be presented in [7].

References

- [1] AKKIRAJU, R., SRIVASTAVA, B., ANCA-ANDREEA, I., GOODWIN, R., AND SYEDA, T. Semaplan: Combining planning with semantic matching to achieve web service composition. In *ICWS (2006)*.
- [2] AU, T.-C., AND NAU, D. The incompleteness of planning with volatile external information. In *17th European Conference on Artificial Intelligence (ECAI-06) (2006)*.
- [3] AXENATH, B., KINDLER, E., AND RUBIN, V. An open and formalism independent meta-model for business processes. In *Proceedings of the Workshop on Business Process Reference Models (2005)*, pp. 45–59.
- [4] BELECHEANU, R., CABRAL, L., DOMINGUE, J., GAALOUL, W., HEPP, M., FILIPOWSKA, A., KACZMAREK, M., KACZMAREK, T., NITZSCHE, J., NORTON, B., PEDRINACI, C., ROMAN, D., STOLLBERG, M., AND STEIN, S. Business Process Ontology Framework. SUPER Deliverable 1.1, April 2007.

- [5] BRANK, J., AND GROBELNIK, M. A survey of ontology evaluation techniques. In *Proc. of the 8th Int. multi-conference Information Society IS-2005* (2005).
- [6] BREWSTER, C., ALANI, H., DASMAHAPATRA, S., AND WILKS, Y. Data-driven ontology evaluation. In *Proceedings of Int. Conf. on Language Resources and Evaluation*, (2004).
- [7] CONSORTIUM, S. D2.3 prototype implementation. SemBiz Deliverable, May 2008.
- [8] CONSTANTINESCU, I., FALTINGS, B., AND BINDER, W. Large scale, type-compatible service composition. In *2nd International Conference on Web Services (ICWS-04)* (2004), pp. 506–513.
- [9] DUSTDAR, S., HOFFMANN, J., HOLMES, T., SIRBU, A., TRAN, H., AND ZDUN, U. D2.2 semantic querying, discovery, and composition framework. SemBiz Deliverable, August 2007.
- [10] FEIER, C., AND DOMINGUE, J. D3.1v0.1 wsmo primer. WSMO Final Draft, April 2005.
- [11] GHEZZI, C., JAZAYERI, M., AND MANDRIOLI, D. *Fundamentals of Software Engineering*. Prentice Hall, 1991.
- [12] GOMEZ-PEREZ, A. Some ideas and examples to evaluate ontologies. Knowledge Systems Laboratory, Stanford University, 1994.
- [13] GUARINO, N., AND WELTY, C. A formal ontology of properties. In *Proceedings of EKAW-2000: The 12th International Conference on Knowledge Engineering and Knowledge Management* (Berlin, 2000).
- [14] GUARINO, N., AND WELTY, C. Identity, unity, and individuation: Towards a formal toolkit for ontological analysis. In *Proceedings of ECAI-2000: The European Conference on Artificial Intelligence* (Amsterdam, 2000).
- [15] GUARINO, N., AND WELTY, C. Evaluating ontological decisions with ontoclean. *Communications of the ACM*. 45(2):61-65., 2002.
- [16] GUARINO, N., AND WELTY, C. An overview of ontoclean. In *Handbook on Ontologies*, S. Staab and R. Studer, Eds. Springer, New York, 2004, pp. 151–159.
- [17] HENTRICH, C., AND ZDUN, U. Patterns for Process-Oriented Integration in Service-Oriented Architectures. In *Proceedings of 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006)* (Irsee, Germany, July 2006).
- [18] HOFFMANN, J., BERTOLI, P., AND PISTORE, M. Web service composition as planning, revisited: In between background theories and initial state uncertainty. In *AAAI* (2007).
- [19] HULL, D., ZOLIN, E., BOVYKIN, A., HORROCKS, I., SATTLER, U., AND STEVENS, R. Deciding semantic matching of stateless services. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)* (Boston, Massachusetts, USA, 2006), AAAI Press.

- [20] IBM, BEA SYSTEMS, MICROSOFT, SAP AG, SIEBEL SYSTEMS. Business Process Execution Language for Web Services (BPEL4WS). <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>, 05 2003.
- [21] IEEE. Recommended Practice for Architectural Description of Software Intensive Systems. Tech. Rep. IEEE-std-1471-2000, IEEE, 2000.
- [22] ISO. Open Distributed Processing Reference Model (IS 10746). http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm, 1998.
- [23] J. HARTMANN, Y. S., GIBOIN, A., MAYNARD, D., DEL CARMEN SUREZ-FIGUEROA, M., AND CUEL, R. Methods for ontology evaluation. KWeb Deliverable, December 2004.
- [24] KELLER, U., LARA, R., LAUSEN, H., POLLERES, A., AND FENSEL, D. Automatic location of services. In *Proceedings of the 2nd European Semantic Web Symposium (ESWS2005)* (Heraklion, Crete, 5 2005).
- [25] KINDLER, E., AXENATH, B., AND RUBIN, V. AMFIBIA: A Meta-Model for the Integration of Business Process Modelling Aspects. In *The Role of Business Processes in Service Oriented Architectures* (2006), no. 06291 in Dagstuhl Seminar Proceedings.
- [26] KUTER, U., SIRIN, E., NAU, D., PARSIA, B., AND HENDLER, J. Information gathering during planning for web service composition. *J. Web Semantics* 3, 2-3 (2005), 183–205.
- [27] LI, L., AND HORROCKS, I. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web* (Budapest, Hungary, May 2003).
- [28] LOZANO-TELLO, A., AND GOMEZ-PEREZ, A. Ontometric: A method to choose the appropriate ontology. *J of DB Mgmt. Spec. Issue on Ontological analysis, Evaluation, and Engineering of Business Systems Analysis Methods* (April-June 2004).
- [29] MARTIN, D., BURSTEIN, M., HOBBS, J., LASSILA, O., MCDERMOTT, D., MCILRAITH, S., NARAYANAN, S., PAOLUCCI, M., PARSIA, B., PAYNEM, T., SIRIN, E., SRINIVASAN, N., AND SYCARA, K. OWL-S: Semantic Markup for Web Services. W3C Member Submission, available at: <http://www.w3.org/Submission/OWL-S>, November 2004.
- [30] MCDERMOTT, D. Estimated-regression planning for interactions with web services. In *6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)* (2002).
- [31] MCILRAITH, S., AND SON, T. C. Adapting Golog for composition of semantic Web services. In *Proc. of the 8th Int. Conf. on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France* (2002).

- [32] MENDLING, J., AND HAFNER, M. From Inter-organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *OTM Workshops* (2005), pp. 506–515.
- [33] MENDLING, J., AND SIMON, C. Business Process Design by View Integration. In *Business Process Management Workshops* (2006), vol. 4103 of *LNCS*, Springer, pp. 55–64.
- [34] NARAYANAN, S., AND MCILRAITH, S. Simulation, verification and automated composition of web services. In *WWW* (2002).
- [35] NITZSCHE, J., AND WUTKE, D. An ontology for executable business processes. In *Workshop on Semantic Business Process and Product Lifecycle Management* (Innsbruck, Austria, 2007).
- [36] OASIS WEB SERVICES BUSINESS PROCESS EXECUTION LANGUAGE (WSBPEL) TC. *Web Service Business Process Execution Language Version 2.0*, January 2007.
- [37] OMG. Business Process Modeling Notation (BPMN). <http://www.bpmn.org/Documents/OMG-02-01.pdf>, 02 2006.
- [38] OUYANG, C., DUMAS, M., TER HOFSTEDE, A. H. M., AND VAN DER AALST, W. M. P. From BPMN Process Models to BPEL Web Services. In *ICWS* (2006), pp. 285–292.
- [39] PAOLUCCI, M., KAWAMURA, T., PAYNE, T., AND SYCARA, K. Semantic matching of web services capabilities. In *Proceedings of The First International Semantic Web Conference (ISWC2002)* (Sardinia, Italy, 2002).
- [40] PISTORE, M., MARCONI, A., BERTOLI, P., AND TRAVERSO, P. Automated composition of web services by planning at the knowledge level. In *19th International Joint Conference on Artificial Intelligence (IJCAI-05)* (2005).
- [41] PISTORE, M., TRAVERSO, P., AND BERTOLI, P. Automated composition of web services by planning in asynchronous domains. In *ICAPS* (2005).
- [42] PONNEKANTI, S., AND FOX, A. SWORD: A developer toolkit for web services composition. In *WWW* (2002).
- [43] SCHMIT, B. A., AND DUSTDAR, S. Model-driven Development of Web Service Transactions. *International Journal Enterprise Modelling and Information Systems Architectures* 1, 1 (10 2005), 46–.
- [44] SHESHAGIRI, M., DESJARDINS, M., AND FININ, T. A Planner for Composing Services Described in DAML-S. In *Proc. AAMAS’03* (2003).
- [45] SIRIN, E., PARSIA, B., GRAU, B., KALYANPUR, A., AND KATZ, Y. Pellet: A practical owl-dl reasoner. *Submitted for publication at "Journal of Web Semantics"*.
- [46] SIRIN, E., PARSIA, B., AND HENDLER, J. Template-based composition of semantic web services. In *AAAI Fall Symposium on Agents and Search* (2006).

- [47] SIRIN, E., PARSIA, B., WU, D., HENDLER, J., AND NAU, D. HTN planning for web service composition using SHOP2. *J. Web Semantics* 1, 4 (2004).
- [48] SKOGAN, D., GRØNMO, R., AND SOLHEIM, I. Web Service Composition in UML. In *Enterprise Distributed Object Computing Conference, 2004* (2004), pp. 47–57.
- [49] SYCARA, K., WIDOFF, S., KLUSCH, M., AND LU, J. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems* (2002), 173–203.
- [50] THOMAS, O., AND FELLMANN, M. Semantic epc: Enhancing process modeling using ontology languages. In *Workshop on Semantic Business Process and Product Lifecycle Management* (Innsbruck, Austria, 2007).
- [51] TRAN, H., ZDUN, U., AND DUSTDAR, S. View-based and model-driven approach for reducing the development complexity in process-driven soa. In *Int. Conf. on Business Process and Services Computing (BPSC)* (Leipzig, Germany, Sept. 2007).
- [52] U. HUSTADT, B. MOTIK, U. S. Reasoning for description logics around shiq in a resolution framework. Technical Report 3-8-04/04, FZI, 2004.
- [53] VAN DER AALST, W., BEISIEGEL, M., VAN HEE, K., KÖNIG, D., AND STAHL, C. A SOA-Based Architecture Framework. In *The Role of Business Processes in Service Oriented Architectures* (2006), no. 06291 in Dagstuhl Seminar Proceedings.
- [54] W3C. Web Service Choreography Interface (WSCI). <http://www.w3.org/TR/wsci>, 08 2002.
- [55] W3C. Web Services Choreography Description Language (WSCI). <http://www.w3.org/TR/ws-cdl-10>, 11 2005.
- [56] WFMC. XML Process Definition Language (XPDL). <http://www.wfmc.org/standards/XPDL.htm>, 10 2005.
- [57] YAN, Z., CIMPIAN, E., AND HOLMES, T. D1.2 Business Process Modeling Ontology (BPMO) version 1. SemBiz Deliverable, September 2007.