Semantic Annotation of Web Service

Ivan Di Pietro, Francesco Pagliarecci, Luca Spalazzi

January 2008

Abstract

In this technical report, we propose a new approach to the discovery, the selection, and the automated composition of distributed processes described as semantic web services through a new semantic annotation.

In existing approaches, the importance of describing web services at the *process-level* is widely recognized, as witnessed by standard languages for describing business processes, like BPEL, and standards for semantic web services, like OWL-S and WSMO. The approaches that do not deal with semantics cannot exploit the ability to do reasoning about what services do. For this reason, in our opinion, the role of semantics is very important in order to solve the above problems. On the other hand, current semantic web services, such as those based on OWL-S and WSMO, in spite of their expressive power, are hard to use in practice. Indeed, they require comprehensive and usually large ontological descriptions of the processes, and rather complex (and often inefficient) reasoning mechanisms.

In our approach, we still have a representation of the service at process level including the concept of state (definitely, a web service can be mapped into a State Transition System), but we reduce to the minimum the usage of ontological descriptions of processes. As a consequence, we can perform a limited, but efficient and still useful semantic reasoning for verifying, discovering, selecting, and composing web services, at the process level.

The key idea is to keep separate the procedural and the ontological descriptions, and to link them through semantic annotations. We define the formal framework, and propose a technique that can exploit simple reasoning mechanisms at the ontological level, integrated with effective reasoning mechanisms devised for procedural descriptions of web services.

1 Introduction

In this technical report, we propose a new approach to discovery, selection, and automated composition of distributed processes described as semantic web services through a new semantic annotation of processes.

The importance of describing web services at the *process-level* is widely recognized, a witness being the standard languages for describing business processes, like BPEL [3], and the most popular standards for semantic web services, like OWL-S [9] and WSMO [1]. In a process-level description, a web service is not simply represented as an "atomic" component - with its inputs, outputs, preconditions, and effects - that can be executed in a single step. Instead, the interface of the service describes its behavior, i.e., a process that interacts with other services in different steps, and which can have different control constructs, e.g., sequence, condition, and iteration. We can exploit the service's behavioral model to perform verification, discovery, selection, and composition tasks and to express requirements at process level (e.g., see [2]). This is especially true for composition, as witnessed by [15, 12, 4, 23, 19]. Some approaches do not deal with semantic web services, and cannot thus exploit the ability to do reasoning about what services do. This is the case of techniques for composing BPEL processes [19] and of theoretical frameworks for the composition of services represented as finite state automata [12, 4]. From the other side, the approaches that have been proposed so far to exploit semantics (see, e.g., [15, 23, 1]) describe processes with comprehensive ontologies. They have the practical disadvantage to require long descriptions that are time- and effort- consuming, and that are very hard to propose in practice for industrial applications. Such semantic descriptions of web services are based on expressive languages such as OWL [14] or WSMO [1] that require complex reasoning mechanism. Indeed, for instance, the OWL family of languages are based on the description logics $\mathcal{SHIQ}$ and $\mathcal{SHIOQ}$, that have reasoning services that are EXPTime and NEXPTime, respectively [22].

In this technical report, we formally define the notion of enriching the representation of web services with a semantically annotated behavioral description and the grounding algorithm that is needed for discovery, selection, verification, and composition. According to the line described in [20, 17], the key idea is to keep separate the procedural description of processes and their ontological descriptions, and adding semantic annotations that link the two. Therefore, the main aspects of this technical report are the following:

- The procedural behavior of a web service is described with languages that have been designed to describe processes; in this chapter we refer to BPEL [3]. A BPEL process can be formally modeled as a State Transition Systems (STS) [23].
- The data exchanged among processes are described in a standard WSDL file.
- The semantics of exchanged data is described in a separate ontological language. The language we use is WSML [1] that belongs to the

Description Logic family [22].

- We define an annotation language that allows us to link data (WSDL) and behavioral (BPEL) definitions of the process with ontology elements (WSMO). The language is based on XML and, from a theoretical point of view, it belongs to the assertional part of the a Description Logic. Annotations are necessary to give semantics to the exchanged data (e.g., which relations exist between the data given in input to the service and the data received as answers from the service), as well as to define the effects and outcomes of the service executions (e.g., to identify the successful executions of the service and distinguish them from the failures, and to describe the effects associated to the successful executions). This approach allows us to annotate only what we need and leave BPEL the duty of describing the behavior.
- We define a language that can express requirements on the behavior of the service that has to be verified, selected, or composed. The language is a temporal logic based on CTL (*Computation Tree Logic*) [11], enriched with concept and role assertions of a Description Logic.
- We propose a grounding algorithm that includes semantic annotations in the STS that models the web services. This allows us to obtain an STS model processable by existing model checkers (e.g.[8]) and planners (e.g. [5]) to solve verification, selection, and composition problems.

The technical report is structured as follows. In Section 2 we expound the overview of our approach. Section 3 shows the schema (XSD) of the annotation file. In Section 4, we define semantically annotated state transition systems that describe BPEL processes. In Section 5, we report the language for describing semantically annotated conditions. Section 6 shows an algorithm for the tranformation of annotated STS and temporal specifications into their propositional versions. Finally, Section 7 reports some concluding remarks.

2 Overview of the Approach: The Methodology

A web service can be characterized in terms of its data and its behavior. Data description is the definition of the data types used within the service and this can be done by means of the standard language WSDL (*Web Services Description Language*). This is not enough, since WSDL only represents the static part of a service. With WSDL, we are not aware of the actual control and data flows in the process: we only know the interface of the web service and the data structures it uses. The behavioral aspects of a service can be represented with several languages. One of the most promising languages, which is going to become a *de facto* standard in process representation, is BPEL (*Business Process Execution Language*). As a consequence, we will refer to BPEL in our running example.

WSDL plus BPEL are the "classical", purely syntactical representation of a process. This provides us with a set of powerful tools to solve several problems, but we need to add semantics to this representation. The use of semantics allows us to use some reasoning techniques that help us solve several problems related to services in a pervasive computing environment, such as selection, discovery and composition.

All these problems have in common the need of verifying certain conditions over the behavior and the semantics of a given service. For instance, let us consider process selection: given a set of processes and a user specification, selection consists in recognizing which processes satisfy the given specification. Thanks to the use of semantics, we can express both processes and specifications as semantically enriched models.

Our approach can be described in four steps:

1. annotation,
2. model translation,
3. grounding,
4. model checking.

Annotation: it is the phase in which both data and behavioral definitions of the process are enriched with links to the ontology. The ontology should be a commonly accepted formalization of a certain domain. It is difficult to have such an ontology, indeed every organization may have its own one. Ontology matching is a parallel problem and we will not delve into it, therefore we assume we have a general shared ontology for our domain.

There are different approaches to process annotation (e.g., see SAWSDL [25]). We propose a novel one that aims mainly at preserving the original syntax of the BPEL and WSDL files. Therefore, in our approach the annotation is put in a different file with links to BPEL and WSDL through XPath expressions.

Model Translation: it consists in expressing our process in a different form that can be model checked easily and automatically. As model checkers usually deal with some kind of State Transition Systems, we translate an annotated BPEL process into an *Annotated State Transition Systems* (ASTS). This step can be done automatically.

Grounding: it is the procedure by which the semantic annotations are "lowered" to a purely syntactic form; roughly speaking, concept and role assertions must be transformed into propositions. From a technical point of view, each annotated state is a knowledge base, therefore we need to use the *query answering service* [16] of Description Logics in order to know which assertions hold in that state. This ensures that our annotations can be treated by existing model checkers that work only with propositions. The grounding is applied to both the Annotated STS and the goal specification, which is expressed in *Annotated CTL*. After the execution of the grounding algorithm, we obtain a gound (propositional) STS and a ground (propositional) CTL specification.

Model checking: it consists in checking whether the ground CTL specification is verified by the ground STS, if not it provides a counter-example. In our experiments, we used NuSMV ([8]), a well-known state-of-the-art model checker. As a consequence, in order to be processed by NuSMV, the ground STS is described into the SMV language.

3 Annotation Technique

The technologies we are going to refer to are:

- WSDL: the “stateless” description of a web service. It gives information about the data used by the service and the exposed interface.
- BPEL: the procedural description of a web process.
- WSML: the language used to build up an ontology.

Our approach is driven by several requirements:

- To annotate web processes with semantics.
- To keep the schema of the technologies safe and computable by existing tools.
- To develop an annotation strategy to cope both with static (“data oriented”) and procedural (“process oriented”) annotations.

All these requirements can be accomplished by keeping the annotation separate from WSDLs and BPELs. Therefore, we will use an annotation file and we will link it to the related files through XPath queries. Our annotation file is defined over XML syntax, so we are going to come up with an XML schema for annotation.

According to Description Logic jargon, within a given ontology we have a terminological component (called *T-BOX*) and an assertional component (called *A-BOX*). The *T-BOX* contains concept definitions as well as generalization and aggregation relationships among them. This part of the ontology is directly provided by a WSML file. The T-BOX simply matches the set of the declared concepts and their structure (i.e. their attributes/roles).

On the other hand, the *A-BOX* contains assertion definitions of two different types:

- concept assertions,

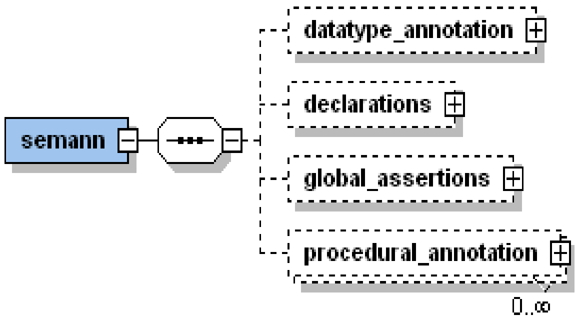

Figure 1: Annotation core components

- role assertions.

Concept assertions have the form *a:C*. Such an expression tells us that the individual *a* is an instance of the concept *C*. For example, *joe:Person* states that Joe is a (instance of) Person.

Role assertions specify the value that a certain role of an individual has. They have the form *a.R=b*. Intuitively, such an expression means that the value of the attribute *R* of the individual *a* is *b*, where *b* is another individual or a literal. For example, *joe.Mother=mary* states that Joe has a mother whose name is Mary. Mary is the identifier of another individual. As second example: *joe.Surname='Smith'* states that the surname of Joe is 'Smith', where 'Smith' is a string.

The roles used in role assertions are defined in the terminological part of the ontology. For example, if the two assertions *joe:Person* and *joe.Surname='Smith'* hold, this means that the concept *Person* has a role *Surname* whose type is string (formally: $Person = \forall\ Surname\ .\ String \sqcap \ldots$).

Basically, in our approach, we have a T-BOX for all the services we have to annotate in our domain — it contains all the concepts we need to represent our application domain — and we associate an A-BOX for each state of each service — it describes what are the effects of a given action in terms of concept and role assertions. Nevertheless, there are some assertions that do not depend on any action, but hold everywhere. For example, we could say that *male* and *female* are individuals of the concept *Gender*. This is always true, no matter how our process evolves. This will become clearer with the introduction of global assertions in our annotation.

Let us now analyze the schema of an annotation file. The core components are shown in Fig 1.

3.1 Datatype Annotation

In this section of an annotation file, there are several assertions that map WSDL elements into WSML concepts. There is no direct mapping between data and concepts at structural level. In other words, it is perfectly legal to annotate an element with a concept whose attributes do not match with those of the element neither in number nor in type.

Data type annotations simply impose some restrictions on the procedural annotations, i.e., the A-BOXes related to the State Transition System. As

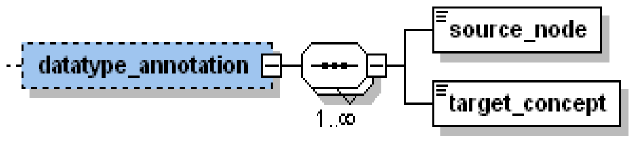

Figure 2: Datatype annotation

shown in Fig 2, datatype annotation is composed of two elements. Source node is a XPath string pointing to a WSDL part (message, element, complex type, simple type or one of their sub-parts). Target concept is a WSML identifier pointing to a concept.

An example of datatype annotation is the following:

3.2 Declarations

We make this assumption: **individuals with the same name (id) refer to the same object**. So, if we had a pair of assertions like:

$god.Exists = True$
$god.Exists = False$

we would refer to the same instance 'god', no matter if the A-BOX would be inconsistent. In the declarations section we simply list all the individuals used in the assertions.

$< source\_node > /definitions/message[7] < /source\_node >$
$< target\_concept >$
$http : //www.diiga.univpm.it/ontologies/virtualStore\#Checkout$
$< /target\_concept >$

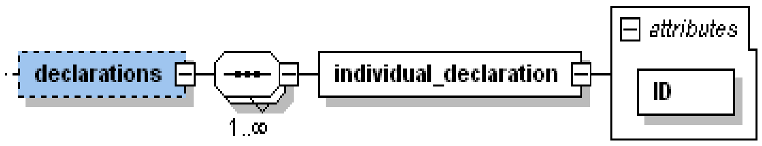

Figure 3: Declarations

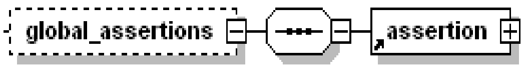

Figure 4: Global assertions

3.3 Global Assertions

Global assertions are concept and role assertions that hold in every state of our process. They are a sintactic sugar, since they have been inserted here just to avoid to repeat them in every state (i.e., in the procedural annotations - see further). Instances directly declared in a WSMO ontology are mapped into global assertions. This is the case of the instances of the concept *Gender*.

3.4 Procedural Annotation

A procedural annotation (see Fig. 5) contains sets of concept and role assertions. A procedural annotation is always referred to a BPEL activity. This activity is identified by the activity attribute, whose value is an XPath expression pointing into the BPEL. Intuitively, a procedural annotation contains all the assertions that hold after that the related BPEL activity has been executed.

3.5 Assertions

Let us now delve into the structure of concept and role assertions. Assertions are used both in global assertions and in procedural annotations. An

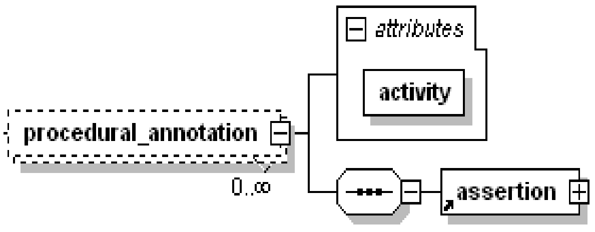

Figure 5: Procedural annotation

assertion is composed of a set of concept assertions and role assertions (Fig. 6).

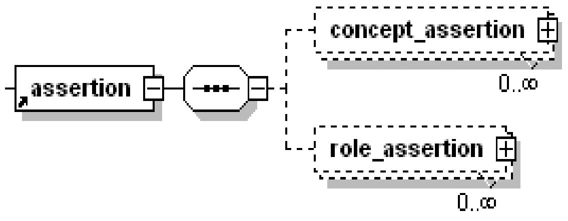

Figure 6: Assertions

A **concept assertion** tells us that in a given activity (state of the STS), there exists an individual of a certain concept. Fig. 7 shows the structure of a concept assertion within the proposed XML schema.

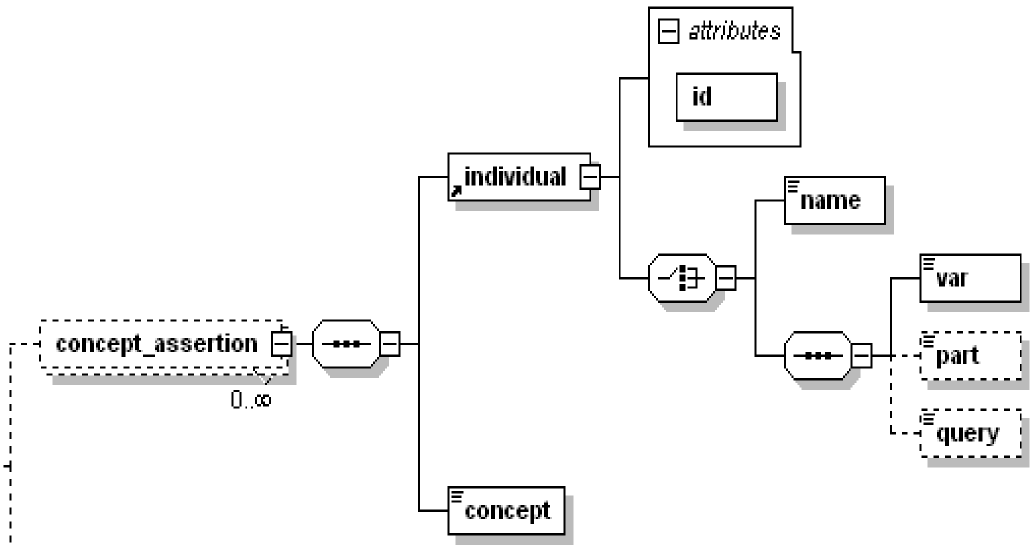

Figure 7: Concept assertion

The individual can be a brand new one, with a name chosen by the user. In the other case, a new individual is associated to a BPEL variable. The variable is identified by its name. It is also possible to refer to a part of a variable and, in case the part is a complex type, we can use an XPath query to delve into it. In this case datatype annotations play their role, because they specify a constraint on this type of annotation. If a datatype annotation has previously mapped variable x into a concept C, then a concept assertion over the same variable x must have the concept C or one of its derived

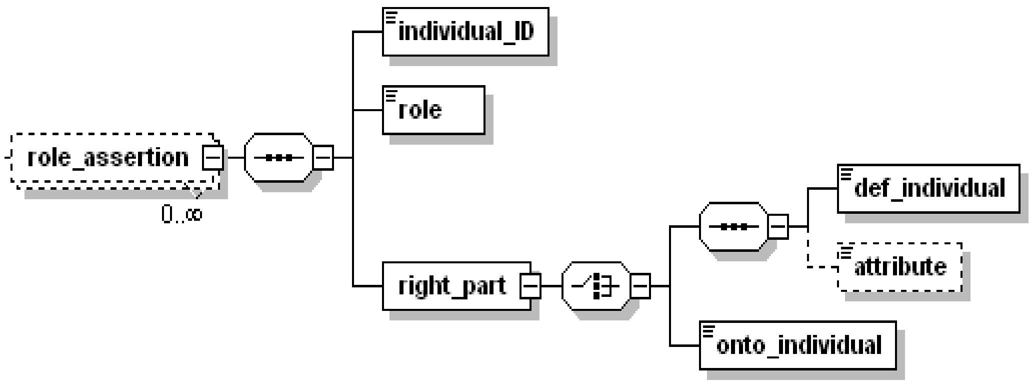

Figure 8: Role assertion

concepts as the right part. In every case, the individual used in the assertion is identified by the value of the id attribute, which refers to a previously declared individual in the declaration section.

Role assertions express a relation between an attribute (role) of a certain individual and another individual.

The left part is an individual previously declared. The role points to one of the attributes of the concept to which the individual belongs to. The role is identified by a WSML URI. The right part of a role assertion may be another individual. The optional attribute element is not currently supported and is intended for future use. It would allow us to create role assertions like the following one:

$$x.R = y.S$$

where S is one of the attributes of y's concept. Alternatively, the right part may be an individual directly declared in the ontology. This kind of individuals are considered as global ones.

4 BPEL Processes as Annotated STSs

We encode BPEL processes (extended with semantic annotations) as *annotated state transition systems*. State transition systems (STS) describe dynamic systems that can be in one of their possible *states* (some of which are marked as *initial states*) and can evolve to new states as a result of performing some *actions*. We distinguish actions in *input actions*, *output actions*, and τ. *Input actions* represent the reception of messages, *output actions* represent messages sent to external services, and τ is a special action, called *internal action*, that represents internal evolutions that are not visible to external services. In other words, τ represents the fact that the state of the system can evolve without producing any output, and without consuming any input (this is a consequence of the fact we use *abstract* BPEL, where the internal actions are "opaque"). A *transition relation* describes how the state can evolve on the basis of inputs, outputs, or of the internal action τ.

In an *Annotated* STS, we associate to each state a set of *concept assertions* and *role assertions*. This configures a state as the assertional component (or ABox) of a knowledge representation system based on a given description logic where the ontology plays the role of the terminological component (or TBox). Therefore, *concept assertions* are formulas of the form $a : C$ (or $C(a)$) and state that a given individual a belongs to (the interpretation) of the concept C. *Role assertions* are formulas of the form $a.R = b$ (or $R(a, b)$) and state that a given individual b is a value of the role R for a. As a consequence, each action can be viewed as a transition from a state consisting in an ABox in a different state consisting in a different ABox.

Definition 1 ***Annotated State Transition System***
An annotated state transition system *defined over a state transition system* Σ *is a tuple* $\langle \Sigma, \mathcal{T}, \Lambda \rangle$ *where:*

- $\Sigma = \langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{P}, \mathcal{X} \rangle$ *is the state transition system,*
- $\mathcal{S}$ *is the finite set of states;*
- $\mathcal{S}^0 \subseteq \mathcal{S}$ *is the set of initial states;*
- $\mathcal{I}$ *is the finite set of input actions;*
- $\mathcal{O}$ *is the finite set of output actions;*
- $\mathcal{R} \subseteq \mathcal{S} \times (\mathcal{I} \cup \mathcal{O} \cup \{\tau\}) \times \mathcal{S}$ *is the transition relation.*
- $\mathcal{P}$ *is the set of propositions that in an annotated STS is empty ($\mathcal{P} = \emptyset$);*
- $\mathcal{X} : \mathcal{S} \to 2^{\mathcal{P}}$ *is the observation function, that in an Annotated STS is undefined.*
- $\mathcal{T}$ *is the terminology (TBox) of the annotation,*
- $\Lambda : \mathcal{S} \to 2^{\mathcal{A}_{\mathcal{T}}}$ *is the annotation function, where $\mathcal{A}_{\mathcal{T}}$ is the set of all the concept assertions and role assertions defined over $\mathcal{T}$,*

5 Conditions on Annotated STSs

In order to express process-level verification, selection, and composition requirements, we need to express conditions on *Annotated STSs*, i.e., conditions on concept and role assertions that hold in given states. In order to do that, we first give some definitions. Let us start with the definition of conjunctive query over a description logic as reported in [16]

Definition 2 (Conjunctive Query)

A conjunctive query q *over* $\langle \mathcal{T}, \Lambda(s) \rangle$ *is a set of atoms* $\{p_1(\overline{x_1}), \ldots, p_n(\overline{x_n})\}$ *where each* $p_i(\overline{x_i})$ *is either* $p_i(x_i)$ *or* $p_i(x_{i,1}, x_{i,2})$ *and* $\overline{x_i}$ *is a tupla of variables or individuals:*

$$p_i(x_i) = x_i : C_i \qquad\qquad p_i(x_{i,1}, x_{i,2}) = x_{i,1}.R_i = x_{i,2}$$

$\mathsf{V}(q)$ denotes the set of variables of q and $\mathsf{C}(q)$ denotes the set of individuals of q. Therefore, $\mathsf{VC}(q) = \mathsf{V}(q) \cup \mathsf{C}(q)$ denotes the set of variables and individuals of q. When $\mathsf{V}(q) = \emptyset$ we have a ground conjunctive query, i.e. each x_i, $x_{i,1}$, or $x_{i,2}$ is an individual. A concept assertion in a propositional condition intuitively denotes a typical description logic problem: the *retrieval inference problem*. Let $x : C$ a goal concept assertion, the retrieval inference problem is the problem of finding for each state s all individuals mentioned in the ABox $\Lambda(s)$ that are an instance of the concept C w.r.t. the given TBox $\mathcal{T}$. A non-optimized algorithm for a retrieval can be realized by testing for each individual occurring in the ABox whether it is an instance of the concept C. Once we have retrieved a set of instances $\{a\}$ for the concept assertion $x : C$, we can substitute x in the propositional condition with the retrieved instances and check whether the condition holds. Therefore, *a conjunctive query denotes in fact a set of specifications* to be checked instead of a single one.

A temporal specification for an Annotated STS is a CTL formula containing conjunctive queries, as defined in the following:

Definition 3 (Temporal Specification of Annotated STS)
A Temporal Specification $\phi(q_1, \ldots, q_m)$ *over* $\langle \Sigma, \mathcal{T}, \Lambda \rangle$ *is a formula defined over the set of conjunctive queries* $\{q_1, \ldots, q_m\}$ *as follows:*

$$\begin{aligned}\phi \;\; = \;\; & q_i \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathsf{AF}\ \phi \mid \mathsf{AG}\ \phi \mid \mathsf{EF}\ \phi \mid \mathsf{EG}\ \phi \mid \mathsf{AX}\ \phi \mid \mathsf{EX}\ \phi \mid \\ & \mathsf{A}\ (\phi\, \mathcal{U}\ \phi) \mid \mathsf{E}\ (\phi\, \mathcal{U}\ \phi) \mid \mathsf{A}\ (\phi\ \mathcal{B}\ \phi) \mid \mathsf{E}\ (\phi\ \mathcal{B}\ \phi)\end{aligned}$$

We can extend to temporal specifications the definition of V as follows: $\mathsf{V}(\phi(q_1, \ldots, q_m)) = \mathsf{V}(q_1) \cup \mathsf{V}(q_2) \cup \ldots \mathsf{V}(q_m)$. The definition of C and VC can be extended in a similar way. A ground temporal specification is a formula without variables, i.e., such that $\mathsf{V}(\phi(q_1, \ldots, q_m)) = \emptyset$. Obviously, we can annotate other temporal languages, as for example EAGLE [10], but for the goal of this chapter, the annotation of CTL formulas is enough.

CTL is a propositional, branching-time, temporal logic. Intuitively, according to our extension, a temporal condition must be verified along all possible computation paths (state sequences) starting from the current state. Concerning the temporal operators (i.e., AF, EF, AX, and so on), they maintain the same intuitive meaning that they have in standard CTL. In other words, $\mathsf{EX}\phi$ is true in a state s if and only if s has at least a successor t such that ϕ is true at t. $\mathsf{E}[\phi\mathcal{U}\psi]$ is true in a state s if and only if there exists a path starting at s and an initial prefix of the path such that ψ holds at the last state of the prefix and ϕ holds at all other states along the prefix. EGϕ is true at state s if there is at least a path starting at s such that ϕ holds at each state on the path. As a consequence of the fact that a temporal specification has concept and role assertions, *a temporal condition denotes a set of specifications* to be checked instead of a single one, as well as for a conjunctive query.

The semantics of a temporal specification is defined in three steps: first, we define the semantics of a ground conjunctive query (for the sake of space, we refer the reader to [16] for details); then, we define the semantics of a ground temporal specification; finally, we define the semantics of temporal specification with variables.

Definition 4 (Semantics of Ground Temporal Specifications of Annotated STS)

Let $\langle \Sigma, \mathcal{T}, \Lambda \rangle$ *be an* annotated state transition system. *Let* q *be a ground conjunctive query over* $\langle \mathcal{T}, \Lambda(s) \rangle$. *Let* ϕ *and* ψ *be ground temporal specifications. Then defined*

- $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models q$ *iff* $\langle \mathcal{T}, \Lambda(s) \rangle \models q$
- $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models \phi \wedge \psi$ *iff* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models \phi$ *and* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models \psi$
- $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models \neg\phi$ *iff* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \not\models \phi$
- $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models$ *EX* ϕ *iff* $\exists t \in \mathcal{S}, \exists \alpha \in \mathcal{I} \cup \mathcal{O} \cup \{\tau\}$ *such that* $R(s, \alpha, t)$ *and* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, t \models \phi$
- $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models$ *EG* ϕ *iff* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models \phi$ *and* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models$ *EX EG* ϕ
- $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models$ *E* $\phi \mathcal{U} \psi$ *iff* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models \psi$ *or* *[* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models \phi$ *and* $\langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models$ *EX E* $\phi \mathcal{U} \psi$ *]*

Definition 5 (Semantics of Temporal Specifications of Annotated STS) *Let* $\phi(q_1, \ldots, q_m)[\overline{x}]$ *be a temporal specification such that* $V(\phi(q_1, \ldots, q_m)) = \{\overline{x}\}$ *is the corresponding set of variables. Then*

$$\phi(q_1, \ldots, q_m)[\overline{x}]^{\mathcal{I}}(s) = \{\overline{a} \mid \langle \Sigma, \mathcal{T}, \Lambda \rangle, s \models \phi(q_1, \ldots, q_m)[\overline{x}/\overline{a}]\}$$

is the interpretation of the temporal specification.

6 The Grounding Algorithm

A temporal specification can be efficiently checked by means of modal checking [6, 8]. Concerning annotated temporal specifications, the basic idea consists of using model checking, as well. However, the traditional model checkers cannot be used, as they are not able to deal with the ontological reasoning necessary to cope with state annotations. In this section, we discuss an approach that makes it possible to reuse existing model checkers

```
Algorithm: Ground Process Generation
input : $\Gamma = \langle\langle \mathcal{S}, \mathcal{S}^0, \mathcal{A}, \mathcal{R}, \_, \_\rangle, \mathcal{T}, \Lambda\rangle$
        $\phi = \phi(q_1, \ldots, q_m)$
output: $\Sigma_o = \langle \mathcal{S}, \mathcal{S}^0, \mathcal{A}, \mathcal{R}, \mathcal{P}, \mathcal{X}\rangle$
        $\Phi$ /* Set of ground CTL formulae */
{
    $\mathcal{P} = \emptyset$
    for each $q_i$ ($1 \leq i \leq m$) do  $\mathcal{A}sser(q_i) = \emptyset$;
    for each $s \in \mathcal{S}$ do {
        $\mathcal{X}(s) = \Lambda(s)$;
        for each $q_i$ ($1 \leq i \leq m$) do {
            $\mathcal{X}(s) = \mathcal{X}(s) \cup$ cq_answer($q_i, \langle \mathcal{T}, \Lambda(s)\rangle$);
            $\mathcal{P} = \mathcal{P} \cup$ cq_answer($q_i, \langle \mathcal{T}, \Lambda(s)\rangle$);
            $\mathcal{A}sser(q_i) = \mathcal{A}sser(q_i) \cup$ cq_answer($q_i, \langle \mathcal{T}, \Lambda(s)\rangle$);
        }
    }
    $\Phi$ = ground_ts( $\{q_1, \ldots, q_m\}, \phi(q_1, \ldots, q_m)$,
                     $\{\mathcal{A}sser(q_1), \ldots, \mathcal{A}sser(q_m)\}$);
    return $\Sigma_o$ , $\Phi$;
}
```

```
Algorithm: ground_ts
/* Ground Spec. Generation */
input : $L$ /* Set of conjunctive queries */
        $\phi$ /* Partially ground CTL formula */
        $A$ /* Set of sets of assertions */
output: $\Phi$ /* Set of ground CTL formulae */
{
    $\Phi := \emptyset$;
    if $L \neq \emptyset$ {
        $a :=$ head($L$);
        for each $a\prime \in \mathcal{A}sser(a)$ do {
            $\varphi :=$ substitute $a$ with $a\prime$ in $\phi$;
            $\Phi := \Phi \cup$
                ground_ts(rest($L$), $\varphi$, rest($A$));
        }
    }
    return $\Phi$
}
```

Figure 9: The algorithm for building a ground STS and related set of ground specifications.

(e.g., NuSMV [8]), in order to exploit their very efficient and optimized verification techniques. This approach is based on the idea to solve the problem of knowing in which states the assertions contained in the temporal specification hold *before the model checking task*. The algorithm is based on the query answering service (e.g., the algorithm reported in [16]). Therefore, the algorithm consists of three steps.

Ground process generation (see Figure 9): it aims to map assertions (of the annotated STS) into propositions in order to obtain a ground STS. It consists of applying the conjunctive query ansuering service for each conjunctive query in the temporal specification and for each state of the annotated STS. Therefore, for each conjunctive query, we have a set of assertions to map into state propositions. The complexity of this algorithm is polynomial in the complexity of the query answering algorithm. The query answering algorithm has been shown to be PTIME-Hard with respect to data complexity for extended *DL-Lite* languages [7]. The result is that the ground process generation has a polynomial complexity with respect to data complexity.

Ground spec. generation (Figure 9): it aims to transform assertions contained in the annotated temporal specification into conditions on state propositions in order to obtain a ground temporal specification. It consists of generating a set of specifications by setting variables in the temporal specification with instances retrieved in the previous step.

Model checking: it consists of applying the model checking algorithm to each ground temporal specification generated in the previous step.

7 Related work and conclusions

This work is based on and extends the work reported in [20, 17]. In this chapter we focused more on the semantic annotation and the grounding process. The WSMO [1] framework recognizes the importance of interfaces that describe behaviors of services, and proposes the use of mediators to match service behaviors against (discovery) goals. However, the WSMO framework includes services representations in its ontological model. We chose to use a bottom-up approach, not dismissing the existing and widespread technologies like BPEL. Indeed, BPEL provides us with the behavioral features of a service, whereas in WSMO we would need to express them in the orchestration and the coreography properties. The work on WSDL-S and METEOR-S [21, 18, 24] provides semantic annotations for WSDL. It is close in spirit to ours, but does not deal with semantically annotated (BPEL) process-level descriptions of web services. The work in [13] is also close in spirit to our general objective of bridging the gap between the semantic web framework and the business process technologies. However, [13] focuses on the problem of extending BPEL with semantic web technology to facilitate web service interoperation, while the problem of automated composition is not addressed.

Recently, an increasing amount of work is dealing with the problem of composing semantic web services taking into account their behavioral descriptions [15, 26, 23, 1]. In this context, research is following two related but different main approaches: OWL-S [9] and WSMO [1]. Approaches based on OWL-S [15, 26, 23] are different from the one proposed in this chapter, since, in OWL-S, even processes are described as ontologies, and therefore there is no way to separate reasoning about processes and reasoning about ontologies. In the approach undertaken in WSMO, processes are represented as Abstract State Machines, a well known and general formalism to represent dynamic behaviors. The idea underlying WSMO is that the variables of Abstract State Machines are all defined with terms of the WSMO ontological language. Our processes work instead on their own state variables, some of which can be mapped to a separated ontological language, allowing for a minimalist and practical approach to semantic annotations and for effective reasoning to discover, select, or compose services automatically. Indeed, the aim of the work on WSMO is to propose a general language and representation mechanism for semantic web services, while we focus on the practical problem of providing effective techniques for selecting and composing semantic web services automatically. It would be interesting to investigate how our approach can be applied to WSMO Abstract State Machines rather than BPEL processes, and how the idea of minimalist semantic annotations can be extended to work with the rest of the WSMO framework. This task is in our research agenda.

In [2] the author proposes combination of the $\mathcal{SHIQ}$(D) DL and μ-calculus. In our approach, we use CTL to express temporal specifications. CTL is subsumed by μ-calculus, according with the minimalist nature of our approach.

References

[1] The Web Service Modeling Framework - http://www.wsmo.org/.

[2] Sudhir Agarwal. A goal specification language for automated discovery and composition of web services. In *International Conference on Web Intelligence (WI '07)*, Silicon Valley, USA, NOV 2007.

[3] T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana. Business Process Execution Language for Web Services (version 1.1), 2003.

[4] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of E-Services that export their behaviour. In *Proc. ICSOC'03*, 2003.

[5] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. In *IJCAI-2001 workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[6] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2), June 1992.

[7] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data Complexity of Query Answering in Description Logics. AAAI, 2006.

[8] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4), 2000.

[9] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services, 2003.

[10] U. Dal Lago, M. Pistore, and P. Traverso. Planning with a Language for Extended Goals. In *Proc. AAAI'02*, 2002.

[11] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 14, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, New York, N.Y., 1990.

[12] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: A Look Behind the Curtain. In *Proc. PODS'03*, 2003.

[13] D. Mandell and S. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In *Proc. of 2nd International Semantic Web Conference (ISWC03)*, 2003.

[14] D. L. McGuinness and Editors F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[15] S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. WWW'02*, 2002.

[16] M. Ortiz, D. Calvanese, and T. Eiter. Characterizing Data Complexity for Conjunctive Query Answering in Expressive Description Logics. AAAI, 2006.

[17] F. Pagliarecci, M. Pistore, L. Spalazzi, and P. Traverso. Web service discovery at process-level based on semantic annotation. In *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems (SEBD 2007), 17-20 June, Torre Canne, BR, Italy*, 2007.

[18] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S Web Service Annotation Framework. In *WWW04*, 2004.

[19] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. In *Proc. IJCAI'05*, 2005.

[20] M. Pistore, L. Spalazzi, and P. Traverso. A minimalist approach to semantic annotations for web processes compositions. In *Proc. of the 3rd European Semantic Web Conference (ESWC 2006)*, Budva (Montenegro), 11–14 June, 2006. Springer–Verlag, Berlin, Germany.

[21] A. Sheth, K. Verna, J. Miller, and P. Rajasekaran. Enhacing Web Service Descriptions using WSDL-S. In *EclipseCon*, 2005.

[22] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.

[23] P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *Proc. ISWC'04*, 2004.

[24] K. Verma, A. Mocan, M. Zarembra, A. Sheth, and J. A. Miller. Linking Semantic Web Service Efforts: Integrationg WSMX and METEOR-S. In *Semantic and Dynamic Web Processes (SDWP)*, 2005.

[25] W3C Semantic Annotations for Web Service Description Language Working Group. Semantic Annotations for WSDL and XML Schema, 2007.

[26] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition using SHOP2. In *Proc. ISWC'03*, 2003.