2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology

Semantic Web Service Selection at the Process-level: the eBay/Amazon/PayPal Case Study

Ivan Di Pietro, Francesco Pagliarecci, Luca Spalazzi
Università Politecnica delle Marche – Ancona, Italy
{dipietro — pagliarecci — spalazzi}@diiga.univpm.it

Annapaola Marconi, Marco Pistore
Fondazione Bruno Kessler – Trento, Italy
{marconi—pistore}@fbk.eu

Abstract

Several approaches have been proposed to tackle the selection of distributed processes described as semantic web services. However, their practical applicability in real composition scenarios is still an open question. Addressing this problem requires on the one hand to deal with services described as stateful business processes and, on the other hand, to consider complex selection requirements concerning both the service interface and its behavior. In fact, in most existing approaches the selection is performed on the basis of the "functional" description of a service, i.e. in terms of its inputs, outputs, preconditions and effects. In this paper, we present our approach for the process-level service selection and evaluate it on a real world scenario that entails a high level of complexity: the eBay Web Services, the Amazon E-Commerce Services and the e-payment service offered by PayPal. The approach is based on a representation of services at the process level that is based on BPEL *and* WSDL *specifications and that extends these standard specifications with minimal semantic annotations that permit to perform an efficient and yet useful, semantic reasoning for the process-level selection of web services.*

1 Introduction

The importance of describing web services at the *process-level* is widely recognized. Consider, for instance, the standard languages for describing business processes, like BPEL [3], and the most popular standards for semantic web services, like OWL-S [8] 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. These information on the process model can be exploited to perform sophisticated kind of automated verification, discovery, selection, and composition of services and to express requirements at process level (e.g., see [2]). This is especially true for composition, as witnessed by [14, 11, 4, 23, 19]. Most approaches that deal with complex behavioral descriptions do not deal with semantic annotations of 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 [11, 4]. From the other side, most of the approaches that have been proposed so far to exploit semantics (see, e.g., [14, 23, 1]) can deal only with atomic services. The few exceptions have the disadvantage of requiring a comprehensive semantic description of the processes. Such descriptions, based on expressive languages such as OWL [13] or WSMO [1], are time- and effort- consuming, and are very hard to propose in practice for industrial applications.

The key idea of our approach is to keep the procedural description of processes separate from their ontological descriptions, and to link the two through semantic annotations. The main characteristics of our approach can be summarized as follows:
• The procedural behavior of a web service is described in BPEL [3]. A BPEL process can be formally modeled as a State Transition System (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 [18] that allows us to link data (WSDL) and behavioral (BPEL) definitions of the process with ontology elements (WSML). The language is based on XML and, from a theoretical point of view, it belongs to the assertional part of a Description Logic. 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*) [10], 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.[7]) and planners (e.g. [5]) to solve verification, selection, and composition problems.

In our previous work we have proposed a discovery algorithm [16] and a composition algorithm [20] that exploit a minimalist semantic annotation of web services. In this paper, we present our comprehensive approach for the process-level service selection and evaluate it on a real world scenario: the eBay Web Services, the Amazon E-Commerce Services and the e-payment service offered by PayPal.

The paper is structured as follows. In Section 2 we give an overview of our approach. In Section 3 we present the Amazon-eBay-PayPal reference example and show the ontology and the annotations used in this case study. Then we show in details the different phases of our approach at work: in Section 4 we show how the annotated BPEL processes are translated into an abstract STS; in Section 5, we describe the language for defining semantically annotated requirements for service selection, verification and composition; in Section 6 we present the automated selection through Model Checking on our case study. Finally, Section 7 reports some concluding remarks.

2 Overview of the Approach

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 this paper. WSDL plus BPEL are the "classical", purely syntactical representation of a process. Enriching these descriptions with semantic annotations allows us to use automated reasoning techniques that help us solve several problems related to services in a pervasive computing environment, such as selection, discovery and composition.

Our approach can be described in four steps: annotation, model translation, grounding, 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. In particular, 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, concepts and role assertions are transformed into boolean propositions. From a technical point of view, each annotated state is a knowledge base modeling the assertions that 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 ground (propositional) STS and a ground (propositional) CTL specification.

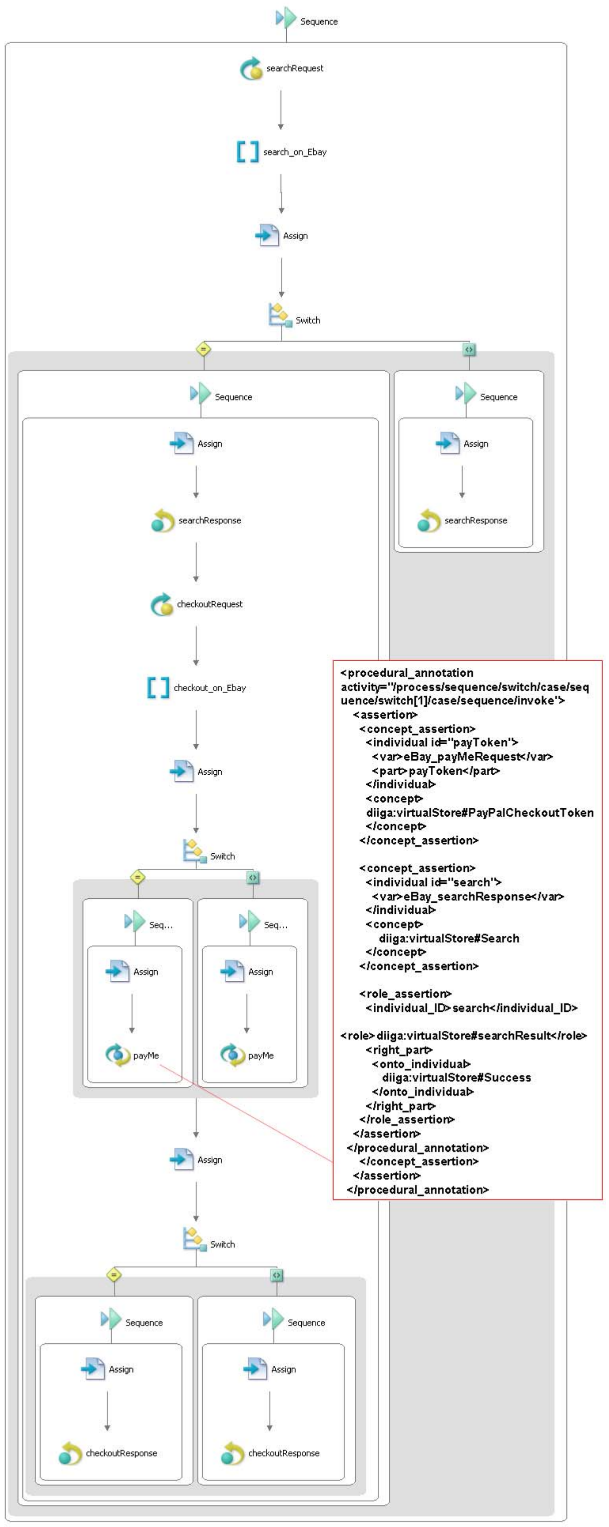
Model checking: it consists in checking whether the ground CTL specification is verified by the ground STS (if not, a counter-example is provided). In our experiments, we used NuSMV ([7]), a well-known state-of-the-art model checker.

3 Case Study Overview

As our case study, we refer to two real services provided by the two major e-commerce sites: eBay and Amazon. For the sake of readability, let us represent the abstract BPEL of the case study graphically[1] (see Fig. 1 and Fig. 2).

Both services allow user to search for an object. Several search mechanisms are possible, but for space's sake we will consider the keyword-based search only.
In case the user found the item he was looking for and he is satisfied with its price, he may proceed with a checkout phase. Of course eBay has a bidding mechanism, but we will not consider it, rather focussing on a "buy now" interaction or the checkout on the Express site. The checkout

[1]All the files are available at
http://leibniz.diiga.univpm.it/ spalazzi/SWS/eCommerceCS.zip



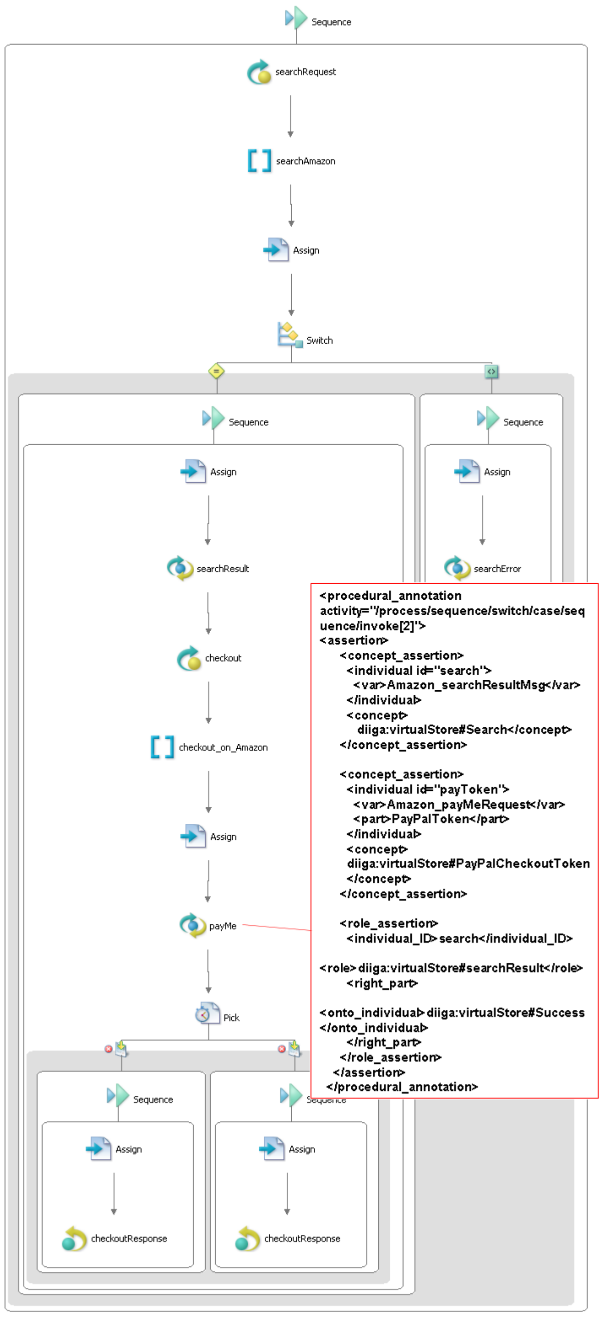
```
<procedural_annotation
activity="/process/sequence/switch/case/seq
uence/switch[1]/case/sequence/invoke'>
  <assertion>
   <concept_assertion>
    <individual id="payToken'>
     <var>eBay_payMeRequest</var>
     <part>payToken</part>
    </individual>
    <concept>
    diiga:virtualStore#PayPalCheckoutToken
    </concept>
   </concept_assertion>

   <concept_assertion>
    <individual id="search'>
     <var>eBay_searchResponse</var>
    </individual>
    <concept>
      diiga:virtualStore#Search
    </concept>
   </concept_assertion>

   <role_assertion>
    <individual_ID>search</individual_ID>

<role>diiga:virtualStore#searchResult</role>
    <right_part>
     <onto_individual>
       diiga:virtualStore#Success
     </onto_individual>
    </right_part>
   </role_assertion>
  </assertion>
 </procedural_annotation>
   </concept_assertion>
  </assertion>
 </procedural_annotation>
```

Figure 1. BPEL of the eBay e-commerce service



```
<procedural_annotation
activity="/process/sequence/switch/case/seq
uence/invoke[2]'>
<assertion>
   <concept_assertion>
    <individual id="search'>
     <var>Amazon_searchResultMsg</var>
    </individual>
    <concept>
      diiga:virtualStore#Search</concept>
   </concept_assertion>

   <concept_assertion>
    <individual id="payToken'>
     <var>Amazon_payMeRequest</var>
     <part>PayPalToken</part>
    </individual>
    <concept>
    diiga:virtualStore#PayPalCheckoutToken
    </concept>
   </concept_assertion>

   <role_assertion>
    <individual_ID>search</individual_ID>

<role>diiga:virtualStore#searchResult</role>
    <right_part>

<onto_individual>diiga:virtualStore#Success
</onto_individual>
    </right_part>
   </role_assertion>
  </assertion>
 </procedural_annotation>
```

Figure 2. BPEL of the Amazon e-commerce service

process has a payment phase that can be accomplished in different ways. The most interesting part is the payment phase, which has a different flow in the two services. In Amazon, we assume that the payment (the *payMe* operation) is performed by PayPal only. Instead, in eBay the payment can be performed over two branches: on the left the payment is done with PayPal, whereas on the right one it is done by means of credit card. How can we express the fact that we have two different payment methods and what these methods are? This information can be needed for selecting or composing these services. Can this information be derived from data types? Generally speaking, the answer is no. Data type definition does not help figuring out what is the specific payment method offered. In our example, this means that the same messages can be used for both the PayPal and credit card payment methods, creating an ambiguity for selection. The usage of semantic annotations, referring an e-commerce ontology, allow to clearly define this kind of information.

According to our approach [20, 16], for each Web service we have: (1) an ontology defining the relevant terminology; (2) an interface process defining the interactions necessary to execute the service; and (3) its annotation that defines (partial) correspondences between the ontology and the process [2]. Our ontology (see Fig. 3) is very simple, but it expresses the main concepts useful to model the e-commerce domain. In the definition of *Checkout* and *Search* we have the attributes *result* and *searchResult* respectively, whose values are restricted to belong to the concept *Result*. The possible values (instances) of *Result* are *CustomCode*, *Failure*, *Success*, *Warning*, ..., as depicted in Fig. 3.

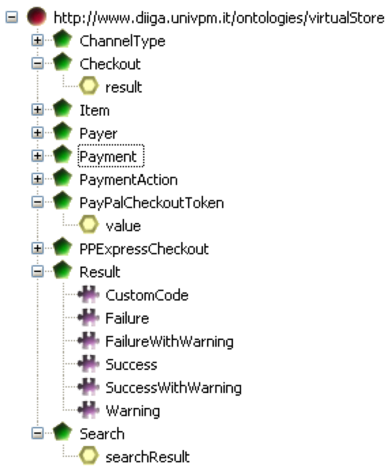


Figure 3. WSML e-commerce ontology

As a second step, we annotate the significant variables and activities of our processes. For the sake of space, let us skip the annotation of the WSDL file (annotation of data types) and go directly to the annotation of the BPEL files (annotation of behavior). Let us suppose we have to say that, after activity *searchResponse* and *searchResult* (see Fig. 1 and Fig. 2, respectively), a search has been performed with a successful result. This means adding the following concept and role assertions to those activities:

search : Search   search.Result = Success

Similarly, in the other branch of the BPEL processes, we have to annotate a failure. For the activities named *checkoutResponse* in the final two branches of the BPEL processes the annotation is similar; indeed, in the left branch we have:

checkout : Checkout   checkout.Result = Success

whereas, in the right branch we have:

checkout : Checkout″   checkout.Result = Failure″

An important annotation must be associated to the payment actions where PayPal is used. This is specified by putting a concept assertion for the concept *PayPalCheckoutToken* in the annotation of the activity *payMe* in the left branch in Fig. 1. The same annotation is used for Amazon in its unique payment action (see Fig. 2).

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. We assume that infinite loops of τ transitions cannot appear in the system. In an *Annotated* STS, we associate a set of *concept assertions* and *role assertions* to each state . 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

[2]We assume that the ontologies provided by the different services have been mapped into a common ontology that defines all the relevant concepts of the scenario — the literature on ontologies defines many powerful techniques to achieve this mapping among the local ontologies of processes.

consisting in a different ABox. Concerning loops, the minimalistic approach allows us to use invariants to annotate actions that must be iterated.

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:*
- $\Sigma = \langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R} \rangle$ *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;*
- $\mathcal{T}$ *is the terminology (TBox) of the annotation;*
- $\Lambda : \mathcal{S} \rightarrow 2^{\mathcal{A}_{\mathcal{T}}}$ *is the annotation function, where* $\mathcal{A}_{\mathcal{T}}$ *is the set of all the concept assertions and role assertions defined over* $\mathcal{T}$.

The representation of the above processes as Annotated STSs has the terminology $\mathcal{T}$ given by the WSML ontology. The corresponding list of concepts is simply reported into the CONCEPTS section of the ASTS.

```
CONCEPTS
 Search, PayPalCheckoutToken, Checkout,
 Result, PaymentAction, ChannelType
```

The Λ function contains, for each state, the set of global assertions (included the concept assertions declared in the ontology) and the assertions, if any, associated to that state. As an example, we report a Λ function for only one annotated state, namely the state in which the payment by PayPal is performed (see Figures 1 and 2).

```
ANNOTATION FUNCTION
 LAMBDA(payMe˙Sync)
  Failure : Result,   Success : Result, Warning : Result,
  SuccessWithWarning : Result, FailureWithWarning : Result,
  CustomCode : Result, Sell : PaymentAction, Order : PaymentAction,
  Authorization : PaymentAction, eVendorItem : ChannelType,
  search : Search,   search.searchResult = Success,
  payToken : PayPalCheckoutToken
```

The last three assertions have been added during the annotation phase, whereas the other ones derive directly from the ontology and are present in every state, since they are global.

5 Conditions on Annotated STSs

In order to express verification, selection, and composition requirements, we need to express conditions on an *Annotated STS*, i.e., conditions on concept and role assertions that hold in given states. Let us start with the notion of conjunctive query over a description logic as defined in [15].

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

$\mathsf{V}(q)$ 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 $x : C$ be 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:*

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

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 [9], but for the goal of this paper, 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. 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.

Now, let us suppose we want to check some extra requirements over the services we found. These requirements may help us selecting the most suitable service among the services returned by a preliminary, keyword-based search. This is the so called *service selection.*

Requirement 1: I want to find an e-commerce service that makes me perform a search and then, if some items are returned, it *guarantees* the possibility to pay through my PayPal account.

In this requirement we can find something different from the classical search engine-fashion search. Indeed, we have a *procedural specification* that looks for a payment *after* a search has been successfully performed. It is clear that we need the concept of time or sequence of actions to express the concept "after". Second, we have a *semantic component* that lets us refer to a PayPal account as described in an appropriate e-commerce ontology. The use of semantics eliminates possible misunderstanding about the terms used. As long as we refer to a commonly accepted ontology, what we mean with search, checkout and account is clear. We can express our requirement in annotated CTL:

```
AF (!(x:Search & x.searchResult=Failure) ->
(x:Search & x.searchResult=Success & y:PayPalCheckoutToken))
```

6 The Selection of Annotated STSs

A temporal specification can be efficiently verified by means of model checking [6, 7]. 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. We adopt an approach that makes it possible to reuse existing model checkers (e.g., NuSMV [7]), 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* and the algorithm is based on the query answering service (e.g., the algorithm reported in [15]).

For the sake of space, we do not report a formal description of the grounding algorithm (the reader can refer to [18]). Here we present the algorithm at work through our reference case study. The idea is to automatically select the service (among eBay and Amazon) that satisfy *Requirement 1*. Let us separate the conjunctive queries that compose the requirement:

$$\begin{aligned} q_1 &= \{\mathsf{x{:}Search}, \mathsf{x.searchResult{=}Failure}\} \\ q_2 &= \{\mathsf{x{:}Search}, \\ &\quad \mathsf{x.searchResult{=}Success}, \mathsf{y{:}PayPalCheckoutToken}\} \end{aligned}$$

The corresponding sets of variables and the individuals are:
$\mathsf{V}(\phi(q_1, q_2)) = \{\mathsf{search}, \mathsf{payToken}\}$
$\mathsf{C}(\phi(q_1, q_2)) = \{\mathsf{Success}, \mathsf{Failure}\}$.

If we apply the query answering algorithm to the previous conjunctive queries and to our eBay ASTS, we obtain in which states the assertions reported in the temporal condition hold. The state *searchResponse2* corresponds to the failure of the search (conjunctive query q_1) and the state *payMe_Sync* corresponds to the payment through PayPal (conjunctive query q_2). The corresponding ground CTL specification is the following:

```
AF (!(search:Search & search.searchResult=Failure) ->
   (search:Search & search.searchResult=Success &
   payToken:PayPalCheckoutToken))
```

At this point, we can verify the satisfiability of the CTL specification for the ground representation of the Amazon and eBay processes, using the NuSMV model checker. As result, we have that Amazon satisfies the specification whereas eBay does not. Indeed, eBay has an alternative payment branch and a counter-example that covers this path is provided.

7 Related work and conclusions

This work is based on and extends the work reported in [20, 16]. In this paper, we present the overall approach for process-level web service selection and evaluate it on a real world case study. 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 choreography properties. The work on WSDL-S and METEOR-S [21, 17, 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 [12] is also close in spirit to our general objective of bridging the gap between the semantic web framework and the business process technologies. However, [12] 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 [14, 26, 23, 1]. In this context, research is following two related but different main approaches: OWL-S [8] and WSMO [1]. Approaches based on OWL-S [14, 26, 23] are different from the one proposed in this paper, 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] S. 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. D. 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] 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.

[8] T. O. S. Coalition. OWL-S: Semantic Markup for Web Services, 2003.

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

[10] 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.

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

[12] 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.

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

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

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

[16] 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, Torre Canne, BR, Italy*, 2007.

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

[18] I. D. Pietro, F. Pagliarecci, and L. Spalazzi. Semantic Annotation of Web Services. Technical report, DIIGA — Università Politecnica delle Marche, 2008. http://leibniz.diiga.univpm.it/∼spalazzi/reports/TR-2008-01.pdf.

[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.