Ad Hoc Composition of User Tasks in Pervasive Computing Environments

Sonia Ben Mokhtar, Nikolaos Georgantas, Valérie Issarny

Ad Hoc Composition of User Tasks in Pervasive Computing Environments

Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny

INRIA Rocquencourt,
78153 Le Chesnay, France
{Sonia.Ben_Mokhtar,Nikolaos.Georgantas,Valerie.Issarny}@inria.fr
http://www-rocq.inria.fr/arles/

Abstract. Due to the large success of wireless networks and portable devices, the pervasive computing paradigm is becoming a reality. One of the most challenging objectives to be achieved in pervasive computing environments is to allow a user to perform a task by composing on the fly the environment's service and resource components. However, existing approaches commonly assume that networked components have been developed to integrate in terms of interfaces and conversations, which restricts the user's ability to fully exploit the diversity of the pervasive computing components. In order to overcome this constraint, we propose a solution for ad hoc composition of pervasive computing components, based on the Web services and Semantic Web paradigms. The main feature of our solution is the ability to integrate on the fly a number of Web services' conversation fragments to reconstruct a conversation enabling the target user task.

1 Introduction

The paradigm of pervasive computing has opened new perspectives to the enactment of human everyday activities related to accessing information and computation. Pervasive computing enables user-centric retrieval and consumption of information, compared to the conventional computer-centric approach. Systemically, this is realized as a synergistic combination of intelligent human-machine interfaces and ubiquitous computing and networking. The ubiquitous property implies a useful, pleasant and unobtrusive presence of the system everywhere around us.

A pervasive computing environment is populated with networked, both computing and input/output devices providing the environment's components. Within this environment, users perform tasks which integrate the functionalities offered by the environment's components. A pervasive computing environment is characterized by a number of features, such as: (i) the highly dynamic character of the computing and networking environment due to the intense use of the wireless medium and the mobility of users; (ii) the resource constraints of mobile devices, in terms of CPU, memory, storage, display capabilities, battery power and bandwidth; and (iii) the high heterogeneity of integrated technologies in terms of networks, devices and software infrastructures. In such an environment, satisfying the functional requirements of user tasks becomes extremely hard. Specifically, the integration of the environment's components to support a user task shall: (i) be dynamic, according to available components at the specific time and place;

T. Gschwind, U. Aßmann, and O. Nierstrasz (Eds.): SC 2005, LNCS 3628, pp. 31–46, 2005.

(ii) satisfy the functional requirements in an effective way within the bounds posed by the resource constraints of mobile devices; and (iii) accommodate the heterogeneity of technologies. However, existing approaches assume that pervasive computing components have been developed to integrate. This means that components being integrated shall perfectly match with the task specification in terms of supported interfaces and conversations.

Our aim is to allow a user who carries an abstract task description on his/her device, i.e., a description without any reference to existing component instances, to perform this task by integrating on the fly the environment's components that are available around him/her, without any preliminary knowledge about these components. This is what we call "ad hoc composition of user tasks". This requires building upon an architectural style that deals with the heterogeneity of pervasive computing components.

Service Oriented Architecture (SOA) is an architectural style that offers solutions to the interoperability problem among distributed applications. In the SOA approach, software resources that are available on the network are abstracted as services. These services are described in a declarative manner that is independent from their implementation; they are loosely coupled, and communicate using standard protocols. This architectural style is most convenient to the pervasive computing environment, as it enables homogeneous use of the heterogeneous software components that populate the pervasive computing environments.

Web services is one of the realizations of the SOA architectural style. Web services introduces loosely coupled services that communicate using the standard technologies that made the success of the Web. More precisely, a Web service is a software component that is developed using any programmation language and deployed on any platform, and that is accessible via the Web. It exposes an XML interface that describes its public operations and its access details; this interface is specified using the Web Services Description Language (WSDL)[1]. A Web service communicates using the Simple Object Access Protocol (SOAP)[2] on top of Internet protocols (HTTP, SMTP). Furthermore, Web services have already been used in pervasive computing environments [13] and have proved to be efficient when deployed on mobile, resource-constrained devices. All these features make Web services an excellent candidate to support application-level interoperability in pervasive computing environments. Thus, by representing each pervasive computing component as a Web service, the problem of dynamic integration of components can be treated as a problem of dynamic composition of Web services.

Dynamic composition of Web services involves automating the discovery and selection of services, ensuring semantic and data-type compatibility [16], but also automating service invocation. The automation of service discovery and selection can be achieved only if service descriptions are machine-interpretable. A key issue is that the two main standards for describing and publishing Web services (WSDL and UDDI[3]) are mainly syntactic. Thus, to discover and select services a strong matching (i.e. a syntactic comparison) have to be made between the required functionalities and the ad-

[1] WSDL: Web Services Description Language. http://www.w3c.org/TR/wsdl

[2] SOAP: Simple Object Access Protocol. http://www.w3c.org/TR/SOAP

[3] UDDI: Universal Description, Discovery, and Integration of Business for the Web. http://www.uddi.org

vertised ones. The semantic representation of service descriptions' content will allow machines to understand and process this content, and to support dynamic discovery and integration. Thus, a number of research efforts have been made to apply to Web services the solutions that have been proposed by the Semantic Web community for semantically annotating Web pages. This leads the emergence of semantic Web services. In this area, Ontology Web Language for Services (OWL-S)[4] is one of the most promising languages for describing semantic Web services.

On the other hand, the automation of service invocation involves the use of conversation languages (e.g., WSCI [27], WS-CDL[26], WSCL[28]) to describe the external behavior of the service rather than describing only the service operations as supported by WSDL. This conversation description prescribes how to interact with the service so that it behaves in a specific desired manner.

Our work presented in this paper is a part of the effort of the IST Amigo project[5] that investigates solutions to realize the full potential of home networking to improve people's lives. In this paper, we present a solution to the ad hoc composition of user tasks from available pervasive computing components. Both the user task and the environment's components are represented as semantic Web services described using OWL-S. The main feature of OWL-S that we exploit, is the ability to describe semantic conversations. More precisely, OWL-S allows the description of the external behavior of a Web service by using a semantic model, in which each operation involved is described semantically in terms of inputs/outputs. The difference between the user task description and the environments' services descriptions, is that the task description is abstract and does not contain any concrete reference to existing Web services.

Our solution introduces a matching algorithm that attempts to reconstruct the abstract task process description by integrating fragments from the environments' services process descriptions. The result obtained is a concrete task description that contains references to available environments' services and that is executable by invoking those services.

The remainder of the paper is structured as follows. First, we present the general context of our work (Section 2). We then present our approach to the ad hoc composition of Web services (Section 3). Further, we review related research efforts in the area of matching algorithms (Section 4). Finally, we conclude with a summary of our contribution and discuss our future work (Section 5).

2 Background

2.1 Service Description

Web services can be described using three description levels that are : (i) the interface-level; (ii) the process-level; and (iii) the binding-level. The interface-level description contains the signatures of the service operations. The process-level description is the specification of the external behavior of the service, which is also called the service's

[4] OWL-S: Semantic Markup for Web Service. http://www.daml.org/services/owl-s

[5] Amigo: ambient intelligence for the networked home environment. http://www.extra.research.philips.com/euprojects/amigo/

conversation. Finally, the binding-level description contains the low level information necessary to communicate with the service, including protocols, addressing and message formats.

WSDL describes services using the interface-level description and the binding-level description. While these two levels are sufficient when the service has a simple behavior (i.e., invocation of a single operation), it remains insufficient when the service has a complex behavior. In this case, the key issue is that we do not have any indication about when and which service operations have to be invoked in order to lead the service to behave as we want. Moreover, for dynamic service invocation, this has to be automated. Thus, by having a conversation description, a requester dynamically can derive the sequence of information to exchange with the service, which in turn correspond to the interaction protocol of the service [22]. This is what makes the process-level description an important requirement to achieve dynamic service composition.

2.2 Semantic Web Services

Semantic Web services is a research area at the intersection of two important research domains that concern Web technologies : the Semantic Web and Web services. The Semantic Web aims at describing the static information provided by Web pages in a non-ambiguous manner, in order to allow machines to understand and process their content [3]. Web services standards, such as WSDL and UDDI, use XML structures to describe Web services functionalities and their exchanged data. However, these structures are syntactic which makes it impossible for a machine to automatically understand Web services descriptions. A semantic Web service is then a Web service described using a well defined semantic language, which provides the service with a non-ambiguous interface, facilitating the automation of certain tasks, such as discovery, invocation and composition.

There are two main approaches to semantically describe Web services. The first approach adds semantics on top of WSDL. The second approach uses ontologies[6] that have been developed specifically to describe Web services.

A number of research efforts have been undertaken in order to bring semantics to WSDL [25,7,11,21,1]. These introduce different models to semantically annotate WSDL descriptions, and some of them propose solutions for mapping XSD structures to ontologies and/or classify services into categories. However, as they are build upon WSDL, these models lack in describing Web services' conversation, which is an important requirement to achieve dynamic service composition.

2.3 OWL-S

OWL-S (previously DAML-S), is a Web service ontology based on the Ontology Web Language (OWL)[7], used to describe Web services properties and capabilities. Using OWL-S a service description is composed of three parts : the service profile, the process model and the service grounding.

[6] Ontologies describe structured vocabularies, containing useful concepts for a community who wants to organize and exchange information in a non-ambiguous manner.

[7] OWL: Web Ontology Language. http://www.w3.org/TR/owl-ref/

The benefit of using OWL-S rather than another approach to semantically describe Web services is that OWL-S offers a generic model to describe Web services. This model is easily extensible and allows the different levels of service description that we have identified earlier. Furthermore, OWL-S brings semantics to the conversation description, and thus allows flexible conversation matching.

OWL-S Service Profile. The service profile gives a high level description of a service and its provider. It is generally used for service publication and discovery. The service profile is composed of three parts: (i) an informal description of the service oriented to a human user; (ii) a description of the service's capabilities, in terms of Inputs, Outputs, Pre-conditions and Effects (IOPE); and (iii) a set of attributes describing complementary information about the service.

OWL-S Process Model. The process model is a representation of the external behavior of the service as a process. This description contains a specification of a set of sub-processes that are coordinated by a set of control constructs (e.g., Sequence or Parallel constructs); these sub-processes are atomic or composite. The atomic processes correspond to WSDL operations. The composite processes are decomposable into other atomic or composite processes by using a control construct.

OWL-S Service Grounding. The service grounding specifies the information that is necessary for the service invocation, such as the protocol, message formats, serialization, transport and addressing information. It is a mapping between the abstract description of the service and the concrete information necessary to interact with the service. The OWL-S service grounding is based on WSDL.

3 Ad Hoc Composition of Semantic Web Services

Ad hoc composition of Web services translates into the integration on the fly of a set of services to perform a user task. Our objective is to allow a user entering into a pervasive computing environment, in which services and resources publish an OWL-S description, to perform a task. A description of this task is available on the user's device as an abstract OWL-S conversation. In order to select the set of services that are suitable to be integrated to perform the user's task, and to integrate this set of services, a matching algorithm is needed. In our approach, we propose a matching algorithm that enables reconstructing the task's conversation using fragments from the conversations of the environment's services. Towards this goal, we first introduce formal modeling of the conversations of both the environment's services and the task as finite state processes (FSP). Other approaches for formalizing Web services conversation and composition have been proposed in the literature, generally based on process algebras (e.g. π-calculus, CCS)[8,15,6], or Petri nets[24,19,17]. FSP is generally used as a textual notation for concisely describing and reasoning about concurrent programs, such as workflows of Web service compositions[10]. These processes can be represented graphically using finite state automata.

In the following, we describe our dynamic composition approach. First, we present the notion of abstract task (Section 3.1). Then, we present our model to map OWL-S

conversations to finite state automata (Section 3.2). Finally, we describe our matching algorithm (Section 3.3).

3.1 Abstract Task Description

While Web services of the pervasive computing environment are described as OWL-S processes with a WSDL grounding, the user task is described as an abstract OWL-S process without any reference to existing services. An abstract OWL-S process involves abstract atomic and composite processes.

An abstract atomic process is defined as an elementary entity that has a set of inputs/outputs. Those inputs/outputs are specified with logical names. They carry semantic definitions, and have to be matched to inputs/outputs of a concrete OWL-S atomic process contained in the description of an environment's service. An abstract composite process is composed of a set of abstract, either composite or atomic, processes, and uses a control construct from those offered by the OWL-S process model. These control constructs are: Sequence, Split, Split + Join, Choice, Unordered, If-Then-Else, Repeat-While, and Repeat-Until.

In addition to the description of the task's conversation, we allow the definition of a set of atomic conversations, that are fragments of the task conversation that must be executed by a single Web service.

3.2 Modeling OWL-S Processes as Finite State Automata

Formally, an automaton is represented by the 5-tuple $< Q, \sum, \delta, S_0, F >$ [12], where:

- Q is a finite set of states.
- $\sum$ is a finite set of symbols that define the alphabet of the language the automaton accepts. ϵ is the empty symbol.
- δ is the transition function, that is $\delta : Q \times \sum \rightarrow Q$
- S_0 is the start state, that is, the state in which the automaton is when no input has been processed yet (Obviously, $S_0 \in Q$).
- F a subset of Q (i.e. $F \subset Q$), called final states.

In our modeling approach, the symbols correspond to the atomic processes involved in the conversation. The initial state corresponds to the root composite process, and a transition between two states is performed when an atomic process is executed.

Each process, either atomic or composite, that is involved in the OWL-S conversation, is mapped to an automaton and linked together with the other ones in order to build the conversation automaton. This is achieved following the OWL-S process description and the mapping rules shown in Figure 1. In this Figure we can see that an atomic process ap is modeled as an automaton $< Q, \sum, \delta, S_0, F >$, where :

- $Q = \{S_0, S_1\}$;
- $\sum = \{ap\}$;
- $\delta(S_0, ap) = S_1$;
- S_0 is the start state ;
- $F = \{S_1\}$.

A composite process C that involves a set of processes $P_1, P_2, ..., P_n$,represented by the automata $< Q_1, \sum_1, \delta_1, S_{0,1}, F_1 >, < Q_2, \sum_2, \delta_2, S_{0,2}, F_2 >, ..., < Q_n, \sum_n, \delta_n, S_{0,n}, F_n >$, respectively, is represented by the automaton $< Q, \sum, \delta, S_0, F >$ according to the control construct it uses, as follows.

- If C=Repeat-While(P_1)[8]
  - $Q = Q_1$;
  - $\sum = \sum_1$;
  - $\delta : Q_1 \times \sum_1 \rightarrow Q_1$
    $(x, y) \mapsto \delta(x, y) = \delta_1(x, y) when (x, y) \in Q_1 \times \sum_1$ and $\delta(x, y) = S_0$ when $x \in F_1$ and $y = \epsilon$;
  - $S_0 = S_{0,1}$;
  - $F = F_1 \cup \{S_0\}$.
- If C=Choice($P_1, P_2, ..., P_n$), then:
  - $Q = (\bigcup Q_i) \cup S_{Init}$ (S_{Init} is a new start state);
  - $\sum = \bigcup \sum_i$;
  - $\delta : \bigcup (Q_i \times \sum_i) \rightarrow \bigcup Q_i$
    $(x, y) \mapsto \delta(x, y) = \delta_i(x, y) when (x, y) \in Q_i \times \sum_i$ and $\delta(x, y) = S_{0,i}$ when $x = S_{Init}$ and $y = \epsilon$;
  - $S_0 = S_{Init}$;
  - $F = \bigcup F_i$.
- If C=Sequence($P_1, P_2, ..., P_n$) then:
  - $Q = \bigcup Q_i$;
  - $\sum = \bigcup \sum_i$;
  - $\delta : \bigcup (Q_i \times \sum_i) \rightarrow \bigcup Q_i$
    $(x, y) \mapsto \delta(x, y) = \delta_i(x, y) when (x, y) \in Q_i \times \sum_i$ and $\delta(x, y) = S_{0,i+1}$ when $x \in F_i$ $(i \neq n)$and $y = \epsilon$;
  - $S_0 = S_{0,1}$;
  - $F = F_n$.
- If C= Split(P_1, P_2) then C is treated as Choice(Sequence(P_1, P_2), Sequence (P_2, P_1)), as we process parallelism as non determinism. The Split+Join and the Unordered constructs are treated as the Split construct.
- If C=If-Then-Else(P_1, P_2) then C is treated as Choice(P_1, P_2).

The conditions involved in the constructs Repeat-While, Repeat-Until and If-Then-Else are not visible in our automata model. However, these conditions will be taken into consideration during the matching process. The OWL class Condition that defines those conditions, is actually a placeholder for further work, and will be defined as a class of logical expressions. Thus, we consider that during our matching algorithm a comparison between those logical expression will be made. More information about this comparison will be given in future work.

[8] If C=Repeat-Until(P_1) then it is treated as the Repeat-While(P_1) but with removing the initial state from the set of final states. The only difference between the Repeat-While construct and the Repeat-Until construct is that the process being repeated is executed at least once in the case of the Repeat-Until construct.

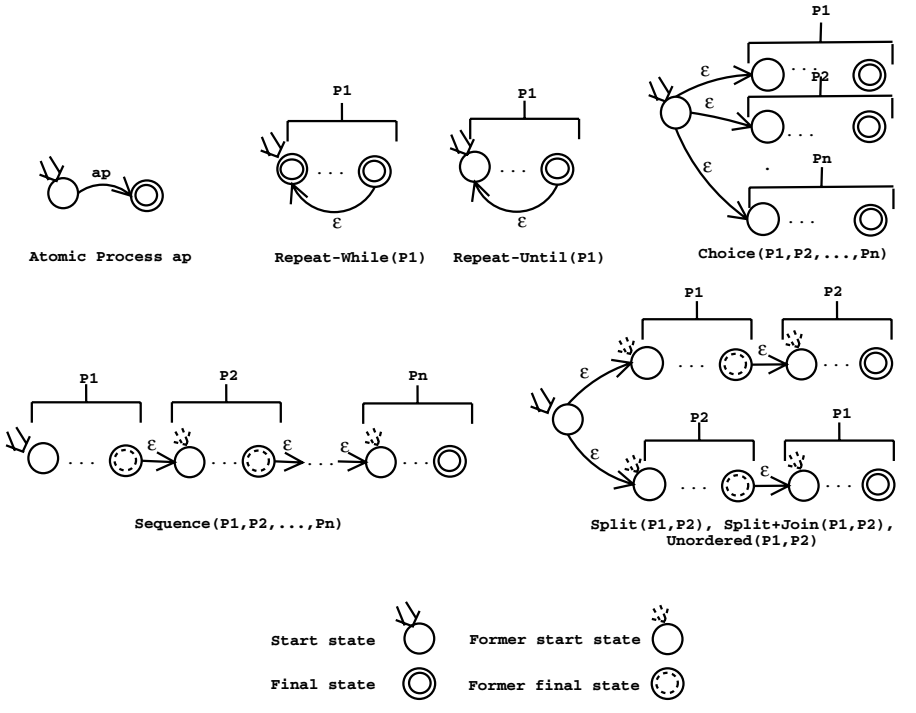

Fig. 1. Modeling OWL-S processes as finite state automata

3.3 Matching Algorithm

One of the most important features of a dynamic service composition approach is the matching algorithm being used. Following the definition given by Trastour et al. in [23], the matching is the process by which parties that are interested in having exchange of economic value are put in contact with potential counterparts. The matching process is carried out by matching together features that are required by one party and provided by another. Thus, the matching allows the selection of the most suitable services to respond to the users' requirements. In our approach, matching depends on two important features : (i) the services' advertisements; and (ii) the task's description. A service advertisement is composed of the information published by the service provider. This description could be quite simple, for example, a set of keywords describing the service, or more complex, describing for example the service's operations, conversation, functional and non-functional capabilities. This description could further be syntactic (by using XML-based standards for Web services' description) or semantic (by using semantic Web languages). In our approach, services are advertised by means of their provided behavior, i.e., conversation, while the user task is described by the behavior it requires from services.

The matching algorithm we propose aims at reconstructing the task's behavior by using fragments of the services behaviors. This algorithm is performed in two steps:

(i) semantic operation matching, and (ii) conversation matching, which are detailed bellow. Semantic operation matching aims at selecting a set of services that may be integrated to compose the target task. Our selection criterion is the provision by the service of at least one semantically equivalent operation from those that are involved in the task. Conversation matching then compares the structure of the task's conversation with those of selected services and attempts to compose fragments from the services' conversations to reconstruct the task's conversation.

Semantic Operation Matching. The objective of the semantic matching step is to compare semantically described operations involved in the task's conversation with those involved in the services' conversations. This kind of matching is more powerful and more flexible than syntactic matching, as it allows the use of inference rules enabled by ontologies to compare elements, rather than comparing syntactically their names.

To perform semantic operation matching, we build upon the matching algorithm proposed by Paolucci et al. in [18,22]. This algorithm is used to match a requested service with a set of advertised ones. The requested service has a set of provided inputs in_{Req}, and a set of expected outputs out_{Req}, whereas each advertised service has a set of expected inputs in_{Ad} and a set of provided outputs out_{Ad}. In our case, we propose to use this matching algorithm to compare atomic processes, i.e., operations, rather that high-level Web services' capabilities. This matching algorithm defines four levels of matching.

- Exact : if $out_{Req} = out_{Ad}$ or out_{Req} is a direct subclass of out_{Ad}
- Plug in : if out_{Ad} subsumes[9] out_{Req}, in other words, out_{Ad} could be used in the place of out_{Req}
- Subsumes : if out_{Req} subsumes out_{Ad}, in this case, the service does not completely fulfill the request. Thus, another service may be needed to satisfy the rest of the expected data.
- Fail : failure occurs when no subsumption relation between advertisement and request is identified.

This matching algorithm also applies between the inputs of the request and the inputs of the advertisement. A match between an advertisement and a request is recognized when all outputs of the request are matched against all outputs of the advertisement; and all the inputs of the advertisement are matched against all the inputs of the request.

Furthermore, we propose to use the two first levels of matching : Exact and Plug in matches, as we consider that a subsumes match cannot guarantee that the required functionality will be provided by the advertised service[16]. Furthermore, as we match operations we don't want to split them between two or more services.

The matching process we are building upon is a complex mechanism that may lead to costly computations. However, the algorithm uses a set of strategies that rapidly prune advertisements that are guaranteed not to match the request[18]. For example, if one of the out_{Req} cannot be matched by any of the out_{Ad} the match directly fails. Furthermore, the fact that we use only the first two levels of matching considerably reduces the cost of the matching.

[9] Subsumption means the fact to incorporate something under a more general category.

The main logic of our semantic operation matching algorithm is that Process model descriptions of services are parsed, and once all inputs/outputs of a task's operation are matched against all inputs/outputs of a service's operation the service is recorded. More precisely, all the operations of this service that are semantically equivalent to task's operations are recorded. This allows the selection of a set of services that offer semantically equivalent operations with those of the user's task. The conversations offered by those selected services are then used to reconstruct the task's conversation.

Conversation Matching. The objective of the conversation matching is to compare the structure of the task's process with the structure of the selected services processes, in terms of control constructs involved. In this algorithm, we use the automaton model describing each service that has been selected and the one describing the user task. The first step is to connect the selected services' automata to form a global automaton. This is achieved by adding a new initial state and an ϵ-transition from this state to each of the initial states of the selected services. Other ϵ-transitions are also added to link each final state of the selected services with the new initial state.

```
Check(TaskState, EnvState){
  if(TaskState is a final state and EnvState is a final state)
     success;
  else{
     if(EnvState.followingSymbols don't include TaskState.followingSymbols )
        fail;
     else{
        forall Symbol in TaskState.followingSymbols, state1 in EnvState.nextState(Symbol),
               state2 in TaskState.nextState(Symbol){
                Check(state1,state2);
        }}}}
```

Fig. 2. Main logic of the conversation matching algorithm

The next step of our conversation matching algorithm is to parse each state of the task's automaton by starting with the initial state and following the automaton transitions. Simultaneously, a parsing of the global automaton is done in order to find at each step of the parsing process, an equivalent state to the current one in the task. An equivalence is detected between a task's automaton state and a global automaton state, when for each input symbol of the former there is at least a semantically equivalent input symbol[10] of the latter. Each state of the task's automaton is parsed just once. We have implemented this algorithm in a recursive form. This algorithm checks whether we can find a sub-automaton in the global automaton that behaves like the task's automaton. The main logic of this algorithm is described in Figure 2.This algorithm gives a list of sub-automata of the global automaton that behave like the task automaton. Figure 3 shows an example of the conversation matching algorithm. In this figure the abstract task automaton (on the left higher corner) that involves the operations op1, op2, op3 and op4 is going to be matched against the global automaton (on the right) which connects together the automata of the services S1, S2, S3, S4 and S5. The operations involved in the task have already been matched semantically against the services' operations during the semantic operation matching step. Thus, in the global automaton we have just

[10] We remind that equivalence relationship between symbols is a semantic equivalence that have already been checked during the semantic matching step.

represented a selected set of operations that are semantically equivalent to the task's operations. After browsing simultaneously the two automata as specified in the algorithm, the sub-automaton of the global automaton represented in the left lower corner of the figure is found. This automaton behaves exactly as the task's automaton.

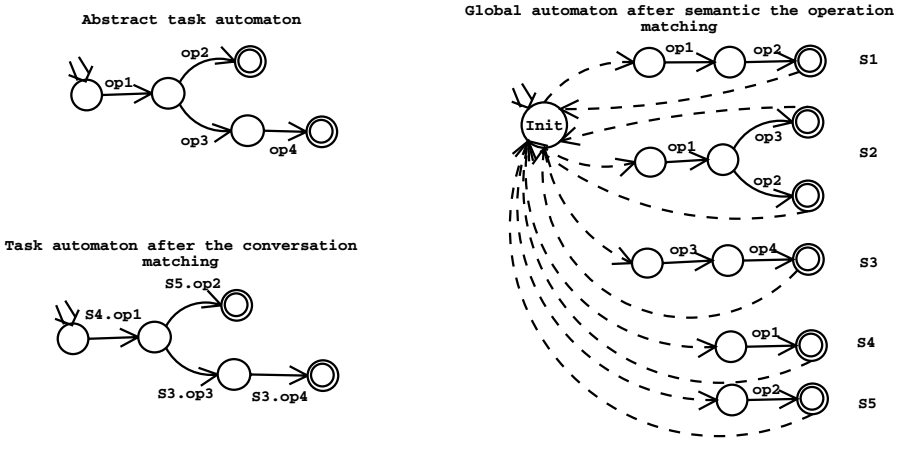

Fig. 3. An example of the conversation matching

Once the list of sub-automata that behave like the task automaton is produced, a last step consist in checking whether the atomic conversation constraints, have been respected in each sub-automaton. As the global automaton is modeled as a union of the selected services automata, it is easy to check whether an atomic conversation fragment, that is, a set of transitions, is provided by a single service. Indeed, it is sufficient to verify that for each transition set that corresponds to an atomic conversation there is no ϵ-transition going to the initial state before this conversation is finished. ϵ-transitions that connect final states to the initial state of the global automaton mark the end of a service conversation and the passing to a new one. After rejecting those sub-automata that don't verify the atomic conversation constraints, we select arbitrarily one of the remainders, as they all behave as the user task. Using the sub automaton that has been selected, an executable description of the user task that includes references to existing environment's services is generated, and sent to an execution engine that executes this description by invoking the appropriate service operations.

An example: a video application In this section we show a simple example of how our matching algorithm could be used to match conversations. This example is inspired from one of the Amigo's scenarios.

"...Robert, (Maria's and Jerry's son) is waiting for his best friend to play video games. Robert's friend arrives bringing his new portable DVD player. He proposes to watch a film rather than playing games, and asks Robert if he has any new films in his home databases. In order to use his friend's DVD player, Robert has asked the system to

consider this device as a guest device and to authorize it to use the home's services. This player is quite complex as it takes into consideration some user's contextual and access rights information. The former is used to display the video streams according to the user's physical environment and preferences (for example by adapting the luminosity and the sound volume), while the latter is used to check whether the user is authorized to visualize a specific stream (for example some violent films may be unsuitable for children)..."

This DVD player contains a video application that uses Web ontologies to describe its offered and required capabilities. The conversation that is published by this application is depicted in figure 4 (left higher corner). This conversation is described as an OWL-S process that contains concrete offered operations (uncolored) and abstract required operations (in gray) that have to be bound to environment's operations. On the other hand Robert's home environment contains a number of services among which a Digital Resource Database service and a Context manager service; both publish OWL-S conversations as shown in the same figure (on the right and left lower corner respectively).

At execution time this device will discover the missing abstract conversation fragment involved in its description. The semantic operation matching step will allow the selection of the two previous services as they contain operations that match the operations of the video application. For example, using the ontology fragment described in figure 5 the operation GetFilm of the video application will be matched against the operation GetDigitalResource of the Digital Resource Database service. More precisely, an exact match is recognized between the output of both operations as they are both instances of the concept Stream. On the other hand, a Plug In match is recognized between the inputs of both operations as the concept DigitalResource subsumes the concept VideoResource. The second step of the matching algorithm is the conversation matching. In this step our algorithm attempts to reconstruct the abstract conversation of the video application by using the conversations of the selected services. The selected fragments after matching are shown in figure 4.

4 Related Work on Matching Algorithms

We can classify the related work on service matching algorithms in two categories: interface-level matching algorithms and process-level matching algorithms. In the first category of matching algorithms, services are generally advertised as a set of provided outputs and required inputs. These inputs/outputs constitute the service's interface. On the other hand, the request is specified as a set of required outputs and provided inputs. A match between an advertisement and a request consists in matching all outputs of the request against all outputs of the advertisement and all inputs of the advertisement against all inputs of the request. An approach for matching semantic Web services at the interface level has been proposed by Paolucci et al. in [18,22]. We have employed this algorithm to semantically match operations as described in Section 3.3. This algorithm is one of the most used approaches in the literature. Because of its simplicity and efficiency, a number of research efforts such as [9,16,23,14,20], have elaborated matching algorithms that are mainly based on this algorithm. In the second category of matching

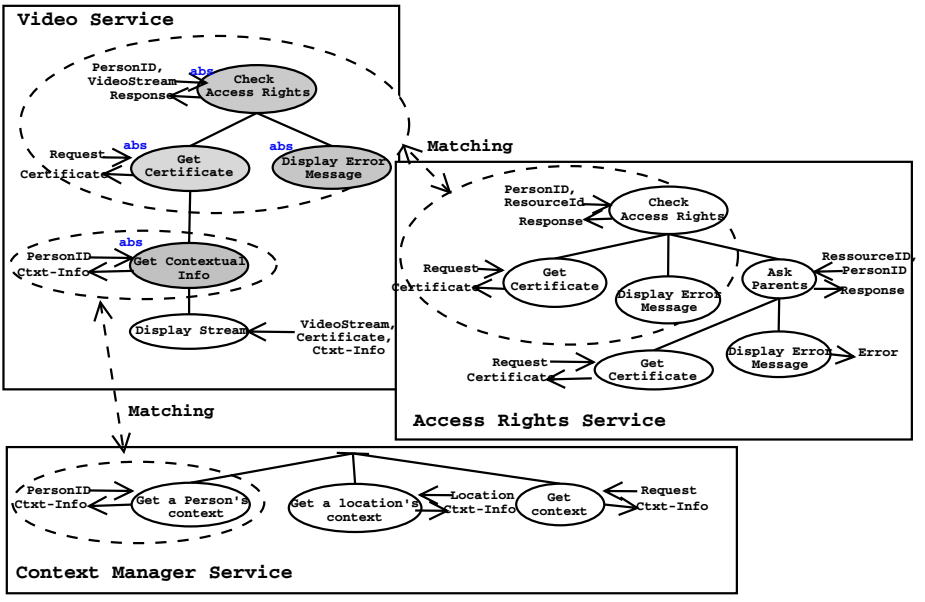

Fig. 4. An example : a video application

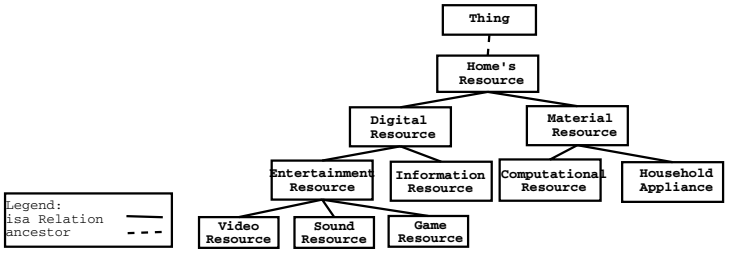

Fig. 5. A fragment of a Home Networking Ontology

algorithms, authors argue that the conversation description is richer than the interface description, as it provides more information about the service's behavior, thus, leading to a more precise matching [2]. Furthermore, we argue that to obtain valid results when dynamically executing a service that publishes a conversation, it is important to follow the sequence of operations to invoke, as specified in the conversation description, until the latter finishes. Consequently, when composing Web services that expose complex behaviors, it is important to have solutions for dynamically integrating conversations. A number of research efforts have been conducted in the area of process-level matching [4,25,16]. For example, Klein et al. in [4] propose to describe services as processes, and define a request language named PQL (Process Query Language). This language allows to find in a process database those processes that contain a fragment that responds to the request. While this approach proposes a new process query language to search for

a process, there is no process integration effort. Thus, the authors implicitly assume that the user's request is quite simple and can be performed by a single process. On the contrary, in our approach a composition effort is made to reconstruct a complex task process by integrating the services' processes.

Aggarwal et al. in [25] propose to describe a task as a BPEL4WS[5] process. This description may contain both references to known services (static links) and abstract descriptions of services to be integrated (service templates). At execution time, services that correspond to the service templates are discovered and the task is carried out by invoking the services following the process workflow. This approach proposes a composition scheme by integrating a set of services to reconstruct a task's process. However, the services being integrated are rather simple. Indeed, each service is described using a semantic model defined by the authors, which specifies the high-level functional and non-functional capabilities of the service, without describing its external behavior (conversation). On the contrary, we consider services as entities that can behave in a complex manner, and we try to compose these services to realize the user task.

Another process-level matching algorithm is proposed by Majithia et al. in [16]. In this approach, the user's request is specified in a high-level manner and automatically mapped to an abstract workflow. Then, service instances that matches the ones described in the abstract workflow, in terms on inputs outputs pre-conditions and effects, are discovered in the network, and a concrete workflow description is constituted. As we have noticed in the previous approach, the service composition scheme that is proposed in this approach does not involve any process integration, as the Web services are only described at the interface level.

5 Conclusion

In the pervasive computing vision the user has a central position. This vision involves that everywhere around us, the environment is populated with networked, both computing and input/output devices that provide the environment's components. Our objective has been to allow a user entering to a pervasive computing environment to perform a task, abstractly described on his/her device, by integrating on the fly the environment's components. A key feature of pervasive computing components is their heterogeneity. Most existing solutions poorly deal with heterogeneity since they assume that components being integrated have been developed to conform syntactically in terms of interfaces and conversations.

Our solution building on semantic Web services offers much more flexibility by enabling semantic matching of interfaces and ad hoc reconstruction of the user task's conversation from service's conversations. Our solution is achieved in two steps. In the first step, we perform a semantic matching of interfaces that leads to the selection of the set of services that may be useful during the integration. In the second step, we perform a conversation matching starting from the set of previously selected services, thus obtaining a conversation composition that behaves as the task's conversation. Our matching is based on a mapping of OWL-S conversations to finite state automata. This mapping facilitates the conversation integration process, as it transform this problem to an automaton equivalence issue.

The main feature of our solution is the ability to compose Web services that expose complex behaviors to realize a user's task that itself has a complex behavior. Existing approaches in process-level matching generally consider either that the services or the task have a simple behavior, thus leading to simple integration solutions. In our case, we assume complex behaviors for both services and task and we propose a matching algorithm that attempts to reconstruct the task's behavior using fragments of the services behaviors. In order to deal with such a high level of dynamicity our solution uses some costly mechanisms such as semantic reasoning. Thus, we intend to implement our approach and to evaluate it in terms of efficiency and performance. We have already implemented the conversation matching algorithm based on automata analysis; the next step is to augment this algorithm with the semantic matching.

References

1. Patil A., Oundhakar S., and Sheth A. Semantic annotation of web services. Technical report, LSDIS Lab, Department of Computer Science, University of Georgia, March 2003.
2. Sharad Bansal and Jose M. Vidal. Matchmaking of web services based on the daml-s service model. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 926–927. ACM Press, 2003.
3. Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.
4. Klein M. Bernstein A. Towards high-precision service retrieval. In Ian Horrocks and James Hendler, editors, *Proceedings of The First International Semantic Web Conference(ISWC), 2002*, number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2342 2002.
5. BPEL4WS. *Business Process Execution Language for Web Services (BPEL4WS) 1.1*. IBM , Microsoft , BEA, 1.1 edition, 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/.
6. Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo. Formalizing web services choreographies. In *Proceedings of the First International Workshop on Web Services and Formal Methods, Pisa, Italy*, February 2004.
7. Andreas Eberhart. Ad-hoc of invocation semantic web services. In *IEEE International Conference on Web Services (San Diego, California ICWS 2004)*, pages 116–124, June 2004.
8. Valrie Issarny Ferda Tartanoglu. Specifying web services recovery support with conversations. In *In Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS-38) , January 2005*.
9. J. G. Pereira Filho and M. van Sinderen. Web service architectures - semantics and context-awareness issues in web services platforms. Technical report, Telematica Instituut, 2003.
10. Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *IEEE International Conference on Automated Software Engineering*, 2003.
11. Andreas Heß and Nicholas Kushmerick. Machine learning for annotating semantic web services. In *AAAI Spring Symposium on Semantic Web Services*, Palo Alto, California, USA, 2004.
12. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation(2nd Edition)*. Addison-Wesley, 2000.
13. Valerie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Francoise Sailhan, Rafik Chibout, Nicole Levy, and Angel Talamona. Developing ambient intelligence systems: A solution based on web services. *Journal of Automated Software Engineering*, 2004.

14. Gonzalez-Castillo Javier, Trastour David, and Bartolini Claudio. Description logics for matchmaking of services. In *Proceedings of the of the KI-2001, Workshop on Applications of Description Logics Vienna, Austria*, September 2001.
15. M. Koshkina and F. van Breugel. Verification of business processes for web services. Technical report, York University, 2003.
16. Shalil Majithia, David W. Walker, and W. A. Gray. A framework for automated service composition in service-oriented architecture. In *1st European Semantic Web Symposium*, 2004.
17. Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the eleventh international conference on World Wide Web*, pages 77–88. ACM Press, 2002.
18. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of Web services capabilities. *Lecture Notes in Computer Science*, 2342:333–??, 2002.
19. Hamadi Rachid and Benatallah Boualem. A petri net-based model for web service composition. In Klaus-Dieter Schewe and Xiaofang Zhou, editors, *Fourteenth Australasian Database Conference (ADC2003)*, volume 17 of *Conferences in Research and Practice in Information Technology*, pages 191–200, Adelaide, Australia, 2003. ACS.
20. P.D.; Guizzardi G.; Ferreira Pires L.; Pereira Filho J.G. van Sinderen M Rios, D.; Costa. Using ontologies for modeling context-aware service platforms. In *OOPSLA 2003 Workshop on Ontologies to Complement Software Architectures*. Anaheim, CA, USA, 2003.
21. Kaarthik Sivashanmugam, Kunal Verma, Amit P. Sheth, and John A. Miller. Adding semantics to web services standards. In *Proceedings of the International Conference on Web Services, ICWS '03, 2003, Las Vegas, Nevada, USA*, pages 395–401, June 2003.
22. Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):27–46, 2003.
23. David Trastour, Claudio Bartolini, and Javier Gonzalez-Castillo. A semantic web approach to service description for matchmaking of services. In *The first Semantic Web Working Symposium, Stanford University, California, USA, July 30 - August 1, 2001SWWS*, pages 447–461, 2001.
24. W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: Yet another workflow language. *Accepted for publication in Information Systems*, 2004.
25. Rohit Aggarwaland Kunal Verma, John Miller, and Willie Milnor. Dynamic web service composition in meteor-s. Technical report, LSDIS Lab, Computer Science Dept., UGA, 2004.
26. WS-CDL. *Web Services Choreography Description Language Version 1.0*. W3C, 1.0 edition, 2004. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.
27. WSCI. *Web Service Choreography Interface (WSCI) 1.0*. BEA Systems, Intalio, SAP AG , Sun Microsystems, 1.0 edition, 2002. http://www.w3.org/TR/wsci/.
28. WSCL. *Web Services Conversation Language (WSCL) 1.0*. Hewlett-Packard Company, 1.0 edition, 2002. http://www.w3.org/TR/2002/NOTE-wscl10-20020314/.