Available online at www.sciencedirect.com

SCIENCE DIRECT

Electronic Notes in Theoretical Computer Science 146 (2006) 33–41

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

An Overview of Semantic Web Services Composition Approaches

Yasmine Charif and Nicolas Sabouret

LIP6 8, rue du Capitaine Scott. 75015 Paris.
{*Yasmine.Charif, Nicolas.Sabouret*}*@lip6.fr*

Abstract

In this paper, we are motivated by the problem of semantic web services composition. We first present a typical example requiring services composition, give a definition of an automated services composition approach and outline its main requirements. We then discuss existing techniques and their limitations with regards to those requirements. Finally, we present our proposal for web services composition based on autonomous services interactions.

Keywords: Semantic Web Services, Composition Approaches, Interaction between Web Services, Ontologies and Context for Web Services.

1 Introduction

The aim of research efforts around semantic web services is to facilitate automated handling of web services. Initial web service efforts failed to hold the promise of automatically interacting, dynamically composed web service. The reason is that the web service technology stack does not supply sufficient means for describing web services in a way that supports generic mechanisms for discovering, composing, and executing web services.

But semantic web and web services are synergistic: the semantic web transforms the web into a repository of computer readable data, while web services provide the tools for the automatic use of that data. Thus, the concept of *Semantic Web Service* (henceforth: SWS) has been established: based on concise and unambiguous semantic description frameworks for web services and related aspects, generic inference mechanims shall be developed for handling SWS.

doi:10.1016/j.entcs.2005.11.005

The above issues are being addressed by ongoing work in the area of SWS [2,3,6]. The overall approach is that by augmenting web services with rich formal descriptions of their competences, many aspects of their management will become automatic. Specifically, web service location, composition and mediation can become dynamic, with software agents able to reason about the functionnalities provided by different web services, to locate the best ones for solving a particular problem to automatically compose the relevant web services to build applications dynamically. [3]

Composition especially is a far from trivial problem: the composition approach must be automatic (the less manual as possible) and deal with the partial observability of the services' internal status, with complex goals expressing commands, temporal conditions and preferences requirements, and with heterogeneous results provided by several services.

In this paper, we will discuss some exiting techniques that have been applied to the SWS composition problem, their limitations with regards to some main requirements and present our approach.

The rest of the paper is organized as follows. Section 2 provides a motivating example that illustrates our requirements for SWS composition, and gives a definition of an automated services composition approach. Section 3 discusses existing techniques and their limitations with regards to those requirements. Finally, section 4 presents our dialogical approach for SWS composition, based on autonomous SWS interactions.

2 Motivating Example

Before summing up the existing approaches dealing with SWS composition, and in order to realize the value added by every approach, it is helpful to have realworld example in mind and to map each approach we present in the next sections to it.

Consider the task of comparing products on the web. In the absence of automated composition of services, the user invests considerable resources visiting numerous sites, determining appropriate service providers, entering his preferences repeatedly, integrating or aligning the different type of results coming from different sites.

We would prefer that the user enters information once and receives the expected results from the most appropriate services with minimal additional assistance. One possible interaction model is given in figure 1.

In this scenario, the user sends a single request, to a service (that we call *Mediator Service*), containing the type of the product concerned (for example: a camera, domotic apparatus) and the information required about the

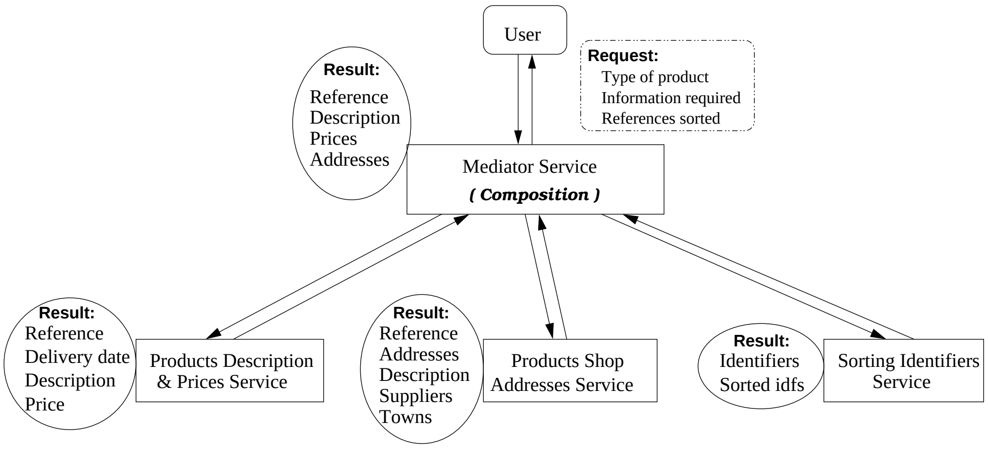

Fig. 1. An interaction model for a comparison service domain.

products (for example: the references, the prices, their description and the shop addresses where they are sold). As it is unlikely to find the *ad hoc* service solving the user's requirements, the service he/she interacts with tries to find several services providing as a result one or more of the needed information. Once these services discovered, the mediator would have to compose the functionalities leading to the needed results and solve their data types constraints.

We can then give a general definition of the SWS composition task. Composing services into one service, given target functionnalities, implies to discover the services that provide one or more of the functionnalities required; retrieve the definition of these specific functionalities (or *actions*); integrate them into the result service and solve data types constraints of the concepts included in the actions definitions. Note that this approach can be opposed to data integration and query mediation: SWS composition rather focuses on the services *functionnalities* and tries to build a chain of actions call that fulfills the user's requirements.

To address such a typical example, we require to solve the following main problems:

- the services discovery and selection, *i.e.* the user must only have to interact with a mediator service that is in turn responsible for the services discovery, selection (following the specified requirements) and composition;
- the composition of specific services competences, *i.e.* the composition task must only consider the needed functionalities, not merge the different services entirely;
- the heterogeneity of the concepts and the actions definitions handled by the

different services;

- to exempt the user of any machine-processable task, *i.e.* automate the composition approach.

We will show in the following sections how some existing approaches for web services composition deal with these problems.

3 Existing Approaches for Web Services Composition

A number of software systems are available to facilitate composition of web services. Such programs enable a user to manually specify a composition of programs to perform some task. The composition process itself, that is the resolution of data types constraints and the effective call of services' actions, should be as much automatic as possible. We will see in this section several approaches for SWS composition and we will discuss their limitations with regards to our example and its requirements.

3.1 Adapting BPEL4WS for the Semantic Web

BPEL4WS [5] (*Business Process Execution Language for Web Services*) provides a notation for describing interactions of web services as *business processes*. Services are integrated by treating them as *partners* that fill *roles* in a BPEL4WS *process model*.

D.J. Mandell and S. McIlraith propose in [7] a bottom-up approach for web services interoperation in BPEL4WS: they collect DAML-S [2] (or OWL-S [1]) service profiles into a repository and exploit their semantics to query for partners based on descriptions of the partners' desired properties. Then, they integrate semantic services descriptions querying into BPWS4J [2] (which is an engine that implements a subset of the features defined in the BPEL4WS specification). Since the current BPWS4J is not extensible, they construct a *Semantic Discovery Service* (SDS) to work within BPWS4J's perspective as an aggregator of web services.

The SDS sits between a BPWS4J process and its potential partners. Instead of routing requests to previously selected partners, BPWS4J directs them to the SDS through a locally bound web service interface. The SDS

[1] The OWL-S (*Ontology Web Language*) model (previsouly called DAML-S) attempts to provide a comprehensive approach to service description. The model has found considerable uptake by the SWS community and as such has set a certain bar against which any other proposals are typically compared.

[2] The BPWS4J engine consumes a BPEL4WS document and WSDL documents defining the bindings for the BPEL4WS process and its partners.

then locates approriate service partners and servers as a dynamic proxy between the BPWS4J engine and the discovered partners.

In our example of section 2, the SDS could be used to discover and invoke the several web services that provide the different results required by the user. However, it does not address the issue of the automated integration and composition of services functionalities. Indeed, DAML-S, which defines a clear semantics for services description, is not provided with a tool or a means for composing *dynamically* the specific functionalities or actions desired. The composite process description must be given *a priori* and cannot be built *at runtime*. As such, the user with the BPEL4WS and SDS tools could only have the sum of results given by all matching services, including nondesired results. In our example, the user would have the list of suppliers (service 2) while he/she did not ask for them. Moreover, the results are heterogeneous since they are directly obtained from the services partners.

3.2 Composition of WSMO-based SWS in IRS-III

Recently, a number of initiatives have started to integrate fully with web services, to support ecommerce and to take into account the notion of goal and mediation[3]. The WSMO ontology [6] (*Web Service Modeling Ontology*) is one of these initiaves' related technologies. IRS-III [3] (*Internet Reasoning Service*) is a framework and implemented infrastructure which supports the creation, publication, composition and execution of SWS according to the WSMO ontology.

D. Sell *et al.* introduce in [12] a graphical tool developed in Java that supports users on the definition of dynamic compositions in IRS-III by recommanding goals according to the context at each step of a composition. The generated composition is performed by their Java API for orchestration. Their approach holds the control of the definition of the composition, but laborious work such as discovery of services according to the users needs is assumed by the machine. Moreover, it introduces additional features such as dynamic invocation of web services in orchestration, control operator and mediation.

Even if such tools can solve our example, they have two main drawback. First, the user must be a computer specialist, whereas the services composition solutions are intended to help *ordinary users* in the web. Second, the manual steps performed by the user, and the services discovery could be done automatically.

[3] Mediation can be applied at several levels: mediation of data structures; mediation of business logics; mediation of message exchange protocols; and mediation of dynamic service invocation.

3.3 Integration of OWL-S into IRS-III

F. Hakimpour *et al.* discuss in [4] that supporting OWL-S will extend the potential of IRS-III in the sense that the separation of the goal and the web services can add to the flexibility in defining a composition of tasks.

Indeed, the IRS web service (component) is suitable for representing a service description as described by `Process` in OWL-S. However in IRS-III, the notion of goal refers to a general description of a problem and can be solved by different web services. A goal describes a problem to be solved and represents the knowledge required for matching the problem to a set of web service descriptions presented by providers.

The authors explain how ontologies describing a service in OWL-S specification (that uses the `ProcessModel` for modelling and describing web services) are mapped to the WSMO ontology (that uses the notion of goal) and translated to OCML which is in turn suitable to be used by IRS-III.

In our example, F. Hakimpour and al. could recommend to describe the web services in OWL-S and use IRS-III to compose them. However, the IRS-III does not perform the services discovery and selection and would constrain the user to do it manually. Moreover, the services would be composed completely, whereas we need to capture only some functionalities dynamically from each service. Finally, this approach as the previous ones is not adapted for ordinary users of the web.

3.4 Petri Nets and Planification

The main weakness of SWS models is their lack of real operational semantics. To this purpose, several attempts such as [9] try to use Petri nets to specify the execution semantics of SWS. However, we believe that the specification still fails in some important points, namely, the SWS composition must work at the implementation level, rather than being the implementation of a model.

In an other hand, several other approaches [13] see the problem of automated web services composition as a software/plan synthesis problem or a plan execution problem. To tackle the composition task as a plan synthesis problem, they perform planning using predefined available services as the building blocks of a plan.

However, as most of the previous approaches, these techniques does not address the problem of the services automatic discovery and selection and would actually compose the services entirely rather than integrating their relevant functionalities.

4 A Dialogical Approach for SWS Composition

We propose in ou approach to implement *Active* SWS (henceforth ASWS), able to interact with *ordinary* human users and other services, and capable of reasoning about their actions *at runtime* so as to compose autonomously and automatically.

In our approach, the user interacts with a mediator ASWS that behaves as an agent and asks him for a set of requirements $\mathcal{R} = \{r_0, ..., r_n\}$. These requirements are expressed using our specific request model [10] that is capable of representing high level user's questions such as "*I'd like you to display, given a product, the products references, descriptions, and prices*". This request model formalizes a request as a sextuple composed of the performative of the speech act of the request, its subject, objects and date, the type of the considered functioning, *etc.*

The mediator then recovers the elements of $\mathcal{R}$ (in our example $\mathcal{R} = \{product\,reference,\ product\,description,\ product\,price\}$) and searches for the appropriate service that provides as a result one or more of the needed requirements $r_i \in \mathcal{R}$. The available web services must be described using the ASWS model. This model is the VDL formalism [11], which stands for *View Design Language*. VDL is an AI-oriented programming language and an execution model that confers to the services it models their autonomous, reasoning and dialogical properties.

In an other hand, every service must be associated to a task ontology that gathers the description of the outputs r_k. A task ontology provides a vocabulary of the terms used to solve problems associated to tasks that can belong or not to the same field [8]. The mediator can then search for the appropriate service using a capability-based discovery (it searches for a service which has an output $r_k = r_i$).

Once the first service selected, corresponding to one of the $r_i \in \mathcal{R}$, the mediator asks the service, using a specific interaction model [1] for the action definition that returns the result r_i. Indeed, the VDL language defines within its interaction model three kinds of **basic interactions** (among which formal requests), which are XML based trees, that services can exchange.

When the mediator receives the action definition he asked for, we say that it "learns" this specific action, it continues with the next requirement. If an action found for a requirement r_j also returns the output r_i, its description is merged with the previously found action description using a specific algorithm [1].

As a consequence, the user interacts with a mediator service (an ASWS) that collects (or learns) the actions that enables it to satisfy the user's requirements. The mediator looks for services returning one or more of the elements of $\mathcal{R}$ and integrates their specific actions definitions removing redundancy and maintaining coherence.

A demonstration is available on our web page.[4]

5 Conclusion

We discussed in this paper existing techniques addressing the problem of semantic web services composition. We first described a typical example and outlined some main requirements to define an automated approach for services composition for ordinary users. We then summed up some techniques proposing to solve the composition problem and their gaps with regards to our requirements. We finally presented our proposal based on modeling autonomous services able to compose themselves using interactions.

Our ongoing work on semantic web services composition needs to be developped. In particular, we want to augment the modelled services with goal-driven behaviours. Thus, they could take initiatives, for example, to invoke a service they know appropriate to solve a certain task, like autonomous agents do. Moreover, we have made tests only on ASWS (VDL services). We want to solve heterogeneous services interoperability issues. Since our services exchange XML-based messages, they are interoprable with any WSDL-compatible agent or service. In particular, we want to interoperate with OWL-S and WSMO-compliant services.

References

[1] Charif, Y. and N. Sabouret, *A Model of Interactions about Actions for Active and Semantic Web Services*, in: *Proc. Semantic Web Service workshop at 3rd International Semantic Web Conference (ISWC'04)*, 2004, pp. 31–46.

[2] David Martin *et al.*, *OWL-S: Semantic Markup for Web Services*, Technical report, DAML Organization (2004).

[3] Domingue, J., L. Cabral, F. Hakimpour, D. Sell and E. Motta, *IRS-III: A Plateform and Infrastructure for Creating WSMO-based Semantic Web Services*, in: *Proc. of the Workshop on WSMO Implementations (WIW 2004)*, 2004.

[4] Hakimpour, F., J. Domingue, E. Motta, L. Cabral and Y. Lei, *Integration of OWL-S into IRS-III*, in: *Proc. of the 1st AKT workshop on Semantic Web Services (AKT-SWS04)*, 2004.

[5] IBM, Microsoft, SAP, Siebel Systems, *Business Process Execution Language for Web Services Version 1.1*, Technical report (2003).

[6] Lauren, H., D. Roman and U. Keller, *Web Services Modeling Ontology - Standard (WSMO-Standard)*, http://wsmo.org/2004/d2/v0.2/ (2004).

[4] http://www-poleia.lip6.fr/~sabouret/demos/index.html

[7] Mandell, D. and S. McIlraith, *Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation*, in: *Proc. of the 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida*, 2003.

[8] Mizoguchi, R., J. Vanwelkenhuysen and M. Ikeda, *Task ontology for reuse of problem solving knowledge*, in: *Proc. 2nd international conference on building and sharing of very large-scale knowledge bases*, 1995.

[9] Narayanan, S. and S. McIlraith, *Simulation, Verification and Automated Composition of Web Services*, in: *Proc. of the 11th WWW Conference*, 2002, pp. 77–88.

[10] Sabouret, N., *A model of requests about actions for active components in the semantic web*, in: *Proc. STAIRS 2002*, 2002, pp. 11–20.

[11] Sabouret, N., *Active Semantic Web Services: A programming model for agents in the semantic web*, in: *Proc. EUMAS*, 2003.

[12] Sell, D., F. Hakimpour, J. Domingue, E. Motta and R. Pacheco, *Interactive Composition of WSMO-based Semantic Web Services in IRS-III*, in: *Proc. of the AKT workshop on Semantic Web Services (AKT-SWS04)*, 2004.

[13] Traverso, P. and M. Pistore, *Automated Composition of Semantic Web Services into Executable Processes*, in: *Proc. of the 3rd International Semantic Web Conference, Hiroshima, Japan (ISWC'04)*, 2004, pp. 380–394.