Dynamic Web Service Selection and Composition: An Approach Based on Agent Dialogues

Yasmine Charif-Djebbar and Nicolas Sabouret

Laboratoire d'Informatique de Paris 6.
8, rue du Capitaine Scott. 75015 Paris
{yasmine.charif, nicolas.sabouret}@lip6.fr

Abstract. In this paper, we are motivated by the problem of automatically and dynamically selecting and composing services for the satisfaction of user requirements. We propose an approach dealing with requirements freely expressed by the user, and in which agents perform service composition through unplanned interactions. Our architecture is based on agents that offer semantic web services and that are capable of reasoning about their services' functionalities. We propose to provide such agents with an interaction protocol that allows them, through dialogues, to select and compose appropriate services' functionalities in order to fulfill a complex set of requirements specified by a user.

Keywords: Service Composition, Service Selection, Agent-based SOA, Interaction Protocol, Agents' Dialogues.

1 Introduction

Service-Oriented Architectures have been recognized as advantageous architectural styles for future enterprise and scientific applications. However, on top of already available middleware layers, many problems regarding service engineering and management, such as service composition [8], have been identified as open issues.

In recent years, web service composition in service-oriented architectures has been actively investigated in the database and semantic web communities. Some of the existing approaches propose to specify manually the composition process [3,9,10]; others need a designer's assistance in a semi-automated approach to composition [6,7]; and others tackle this problem as a planning task [1,15]. However, these approaches present recurring limitations:

- Most of them require the user to have low-level knowledge of the composition process; *e.g.* in the case of [9,10], the user is expected to set up a workflow at the XML level, and the tools proposed in [6,7] propose to the user low-level semantic suggestions. Consequently, these approaches cannot be handled by ordinary users of the web.
- In some of these approaches, *e.g.* [11,16], the user is not free to express his or her requirements and is rather constrained to choose a "generic goal" in a predefined list, *e.g. "Making the travel arrangements for the ICSOC conference trip"*.

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 515–521, 2006.

- Automated service composition requires handling the discovery and selection of services. Solutions that are proposed for these tasks are based on semantic annotations, which does not guarantee (on its own) the provision of the service that fulfills all the user's request. For instance, suppose that a service has been annotated as providing a sports coach and a user expresses the query *"I would like to engage a sports coach starting 4 July"*. A service discovery process based on semantic annotations may encounter scalability problems, as it will propose all the sports coach services, while the user needs a more specific service available from 4 July.
- Most of these approaches assume that the services to compose are elementary since they only propose solutions to compose the overall services and not the specific functionalities required [1,15].

These drawbacks lead us to propose a new approach for a **dynamic** service selection and composition. In such an approach, services are selected and composed on the fly according to requirements freely expressed by the user [12]. Therefore, this is the suitable approach for end-user applications in the web where available components are dynamic and expected users may vary.

The composition approach we are developing is based on multi-agent systems (MAS). Indeed, dynamic service selection and composition can benefit from solutions provided by research in MAS. For instance, service composition can be performed dynamically through agent collaboration, without predefining abstract plans. In addition, agent technology offers well-developed approaches to formally express and utilize richer semantic information, such as nonfunctional characteristics of web services or qualitative constraints on the results proposed by a service. On another hand, local and reactive processing in MAS can avoid centralized systems' bottlenecks and improve scalability.

Overview of Our Composition Approach

In our approach, we define semantic web services provided by interactive agents capable of reasoning about their services' functionalities. Our overall approach can be broadly decomposed into three steps (see figure 1):

1. Formalize and decompose the user's requirements into one or several interdependent requests, which are sent to a mediator agent responsible for the discovery of candidate services;
2. Discover and retrieve the candidate services from a registry using keywords extracted from the user's requests;
3. Select and compose the services' functionalities through the interactions of the mediator agent and the agents providing the candidate services, until the user's requirements are satisfied.

In this paper, we assume that the first two phases have been performed and we focus on the service composition phase by defining *an interaction protocol* allowing agents/services to dialogue so as to respond to a complex query requiring the coordination of several services. Three features distinguish our proposal

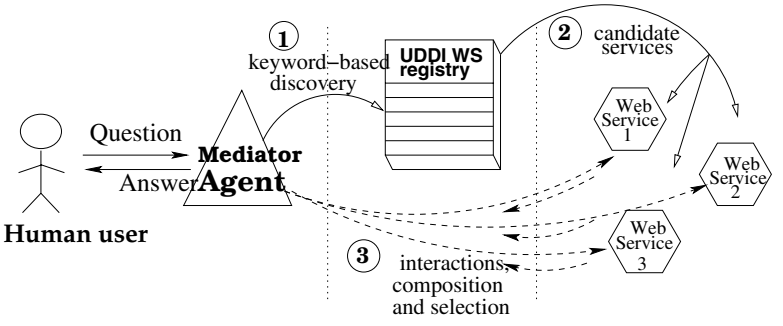

Fig. 1. Overview of our Agent-based Composition Approach

from other work in the area. First, in our approach web services are pro-active and can thus engage in complex conversations, instead of just simple invocation and response with a result or error. Second, since it is performed through a dialogue model, our composition process is dynamic and doesn't require predefined plans or a predefined order of invocation of the services' operations. Finally, the use of agent technology allows for the expression of global constraints, making it possible to select services according to specific user requirements.

In the rest of this paper, we present briefly our web service architecture based on MAS in section 2. We describe in section 3 the interaction protocol making it possible for agents/services to manage their dialogues so as to select and compose the appropriate services with respect to user's requests and constraints. Finally, in section 4 we discuss the limitations of our current work, and place it in perspective.

2 A Web Service Architecture Based on MAS

The aim of our research is to take draw on solutions provided in MAS to propose an approach for service selection and composition. To this end, we implemented a web service architecture following the definition given by the W3C [5] where a service is viewed as part of an agent. Our web service architecture is thus based on interactive agents that are capable of providing semantic web services and reasoning about them.

We designed these agents as the combination of three layers. *A concrete agent* reasoning about its offered service and processing the inter-agent communication issues; *an abstract service* representing an XML-based declaration of the functionalities offered by the service, associated to an ontology defining the concepts and actions it handles; and *a web service* which is the WSDL interface of the abstract service deployed on the web. To model such agents, we propose to use the VDL formalism. VDL (for View Design Language) [14] is a programming language for autonomous agents capable of interacting with the human user and other agents. VDL allows to implement agents able to offer semantic web

services, described in VDL-XML, to about reason them and answer requests for them. As illustrated in figure 2, using a generated HTML interface, a human user can interact with a VDL agent (a mediator agent) or invoke the service it provides. This mediator agent can in turn discover services from a registry and interact with other agents about their services so as to satisfy the user's requirements.

In our architecture, the human-agent interactions are made possible by the VDL request model [13]. This model ensures the formalization of the user and agents requests and makes it possible to represent a wide range of questions, commands, assertions, *etc.* The inter-agent interactions are ensured by the agent communication language (ACL) encompassing requests about services into structured messages. The content of each message is a set of requests among which dependencies can be raised. Services can then either be invoked by SOAP or, as we previously mentioned, accessed via the agent offering it through the ACL.

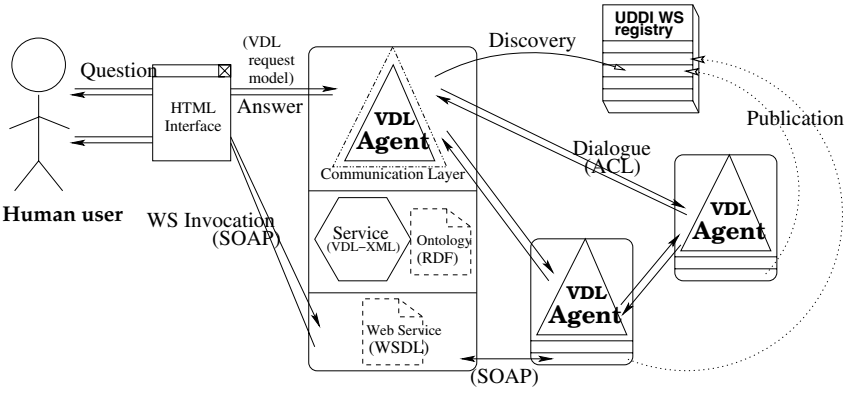

Fig. 2. Our Agent-based SOA

3 The Agent's Interaction Protocol

In our approach, the user's requirements are formalized as a set of (dependent) requests, *e.g. "I want to buy and have delivered a piano, and remove my old wardrobe"*, plus a set of constraints about the services discovered, *e.g.* QoS or trust parameters, or constraints about the results they provide, *e.g. "I want to pay less than $300 for both the wardrobe removal and the piano delivery"*. These requests and constraints are stored within the mediator agent in a record (gathering the information related to a composition goal), comprising also the set of messages exchanged with other services to satisfy the user's requests. Such a record is stored within a history table, provided for each agent, containing all the information related to an agent interactions.

To design our protocol, we consider that an agent has not only to answer to a message according to its services' functionalities, but also to take the initiative

whenever the message comprises requests requiring the coordination of several services. More precisely, the agent tries to discover services that may be composed so as to fulfill the user's needs. This composition and selection mechanism is performed following three main phases supported by three algorithms we defined. For each phase, agents' behaviors have been defined according to their role (mediator agent, or participant agent providing a candidate service).

3.1 Processing Triggering Messages

The mediator agent starts by broadcasting the triggering message, *i.e.* the message containing the user's requests, to the discovered services. When a participant agent, providing a service, receives such a message, it builds an answer to each request following its service's functionalities and knowledge using its request processing module (RPM) [13]. This module builds for each request range (assertion, command, *etc*) the corresponding answer according to the service's functionalities. The agent then returns the answering message to the sender.

Suppose in our example that the mediator agent discovered a music equipment service, a removal service, a delivery service, and another delivery service offering to remove equipments under certain conditions. In this step, each service answers according to its functionalities and knowledge. For instance, the music equipment service sends a proposal for a piano specifying its characteristics and price, and one of the delivery services informs the mediator agent that it can't answer to the delivery request as long as it hasn't been sent information related to the piano (the delivery cost could for instance depend on the value and weight of the equipment to deliver).

3.2 Message Composition

The processing of a non triggering message depends on the role of the agent receiving it. If this agent is the mediator agent, it analyzes it together with the stored messages in order to relaunch the interaction if needed. For instance, if one of the services sent an answer stating that it needs a variable value, and if the received message provides an assertion about this variable, the mediator agent builds a composed message containing this assertion and invokes the specific service requiring it. If a participant agent receives a non triggering message, in addition to it's service's functionalities and knowledge, it uses the assertions contained in the received messages as knowledge to solve the received requests.

In our example, the delivery service needs information related to the piano that the music equipment service sent to the mediator agent. Following the second step of our protocol, the mediator agent sends a composed message containing the assertion about the piano characteristics and invokes the delivery service needing it.

3.3 Service Selection

The mediator agent is provided with a timeout value over which, if no more message arrives, it proceeds to service selection. This step aims at selecting services that best fit to the user's constraints among those which could answer to the user's requests.

To perform this selection, the mediator agent builds for each triggering request (composing the triggering message) a candidate answers list storing the ids and answers of the services that could respond to this request. For each stored constraint, for instance `deliveryCost+removalCost<$300`, the mediator agent builds the corresponding expression using the services answers. In our example, the mediator agent build the expression `deliveryCost+removalCost` and assesses it first with the first delivery service and the removal service answers (where the expression is assessed to `$350`), then the second delivery service which offers equipments removal (where the expression is assessed to `$280`). Comparing the obtained results with respect to the constraint, the mediator agent removes from the candidate answers list the services that don't respect the constraint. In our example, the first delivery service and the removal service are removed from the candidate answers lists. The mediator agent then returns to the user, for each triggering request, the corresponding service and the answer it provided.

4 Conclusion and Future Work

We have proposed in this paper a dynamic approach to performing web service selection and composition using unplanned agent-based dialogues. Our approach rests on a web service architecture that supports pro-active and interactive web services provided by agents. We have specified an interaction protocol that allows agents interacting about their services functionalities so as to satisfy a composition goal with respect to requests and constraints freely expressed by the user. This protocol specifies the agents' behaviors according to their role (mediator or participant) in the composition process and allows them to coordinate so as to solve dependent requests. As a result, the mediator agent performs a selection over the proposed answers with respect to the user constraints and provides the user with the set of services, together with their answers, that could respond to both his requests and constraints.

However, even though our interaction protocol performs well in the presented example, in future work we need to consider several improvements in order to achieve greater scalability in the composition and selection techniques. For instance, if a user constraint involves the results of several services, as in the presented scenario, the service selection step should try different answer combinations. We propose to use MAS algorithms for negotiation [2] during this step to retrieve the best combinations of answers respecting complex constraints. Moreover, we also envision for our interaction protocol to consult with the human user whenever more information is needed by a service to answer a request, or to solve conflicts or multiple choice situations. On another hand, rule-based policies [4] might condition the evolution of the conversation, *e.g.* the requester might require a certificate to be sent by the service provider in order to disclose his credit card details. Finally, we plan to study (both theoretically and empirically) the performance of our system, so as to provide a quantitative demonstration of the benefit to our system's scalability from the use of MAS techniques.

References

1. R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint Driven Web Service Composition in METEOR-S. In *Special Issue of the International Journal of Electronic Commerce (IJEC)*, 2004.
2. S. Aknine. Improving Optimal Winner Determination Algorithms Using Graph Structures. In *Proc. Agent Mediated Electronic Commerce*, 2004.
3. B. Benatallah, Q.Z. Sheng, and M. Dumas. The Self-Serve Environment for Web Services Composition. *IEEE Internet Computing*, 7(1):40–48, 2003.
4. P. A. Bonatti, N. Shahmehri, C. Duma, D. Olmedilla, W. Nejdl, M. Baldoni, C. Baroglio, A. Martelli, P. Coraggio V. Patti and, G. Antoniou, J. Peer, and N. E. Fuchs. Rule-based Policy Specification: State of the Art and Future Work. Technical report, Project deliverable D1, Working Group I2, EU NoE REWERSE, Sep 2004.
5. D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. Technical report, W3C Working Group Note 11, 2004.
6. L. Chen, N.R. Shadbolt, C. Goble, F. Tao, S.J. Cox, C. Puleston, and P. Smart. Towards a Knowledge-based Approach to Semantic Service Composition. In *Proc. 2nd International Semantic Web Conference*, 2003.
7. J. Domingue and S. Galizia. Towards a Choreography for IRS-III. In *Proc. of the Workshop on WSMO Implementations (WIW 2004)*, 2004.
8. A. Gustavo, F. Casati, H. Kuno, and V. Machiraju. Web Services. Concepts, Architectures and Applications, 2004.
9. IBM Alphaworks. BPWS4J. `http://www.alphaWorks.ibm.com/tech/bpws4j`, 2002.
10. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Services Description Language (WS-CDL). Technical report, W3C, 2004.
11. S. McIlraith and T.C. Son. Adapting golog for Composition of Semantic Web Services. In *Proc. of the 8th International Conference on Knowledge Representation and Reasoning (KR'02)*, 2002.
12. T. Osman, D. Thakker, and D. Al-Dabass. Bridging the Gap between Workflow and Semantic-based Web services Composition. In *Proc. of the Web Service Composition Workshop WSCOMPS05*, 2005.
13. N. Sabouret. A model of requests about actions for active components in the semantic web. In *Proc. STAIRS 2002*, pages 11–20, 2002.
14. N. Sabouret. Representing, requesting and reasoning about actions for active components in human-computer interaction. Technical Report 2002-09, LIMSI-CNRS, 2002.
15. P. Traverso 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)*, pages 380–394, 2004.
16. M. Vallée, F. Ramparany, and L. Vercouter. Flexible Composition of Smart Device Services. In *Proc. of the 4th International Conference on Pervasive Computing*, pages 91–96, 2006.