Automatic Composition and Selection of Semantic Web Services

Tor Arne Kvaløy[1,2], Erik Rongen[1], Alfredo Tirado-Ramos[2], and Peter Sloot[2]

[1] IBM Center for Advanced Studies,
David Ricardostraat 2-4, 1066 JS Amsterdam, The Netherlands,
torarnek@pvv.org, erik@nl.ibm.com,
[2] Faculty of Sciences, Section Computational Science, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands,
{alfredo,sloot}@science.uva.nl

Abstract. Interactive applications like Problem Solving Environments require on demand access to Web Services, where the services are autonomously discovered, composed, selected and invocated based on a description of requested capabilities. Semantic Web Services aim at providing semantically interpretable capabilities through the use of shared ontologies. We demonstrate how Grid Services for an interactive biomedical application are annotated with a domain ontology, and propose algorithms for automated composition and selection of workflows, where workflows are created by semantically matching service capabilities, and where workflow selection is based on a trade-off between the types of semantic matches in the workflow and the number of services. The algorithms are demonstrated on semantically annotated Grid Services in the biomedical application.

1 Introduction

Web Services are self-describing, self-contained units of application logic that are either used as a single service or composed into a workflow that is executed sequentially as a whole. Creation of such workflows has traditionally been accomplished by manually locating suitable services via a registry like UDDI and then composing them into a workflow. With the increase of available services and the dynamic nature of the Web it is desirable to automate these processes by having software agents discover, compose, select and execute workflows autonomously [1].

XML Schema based standards for describing Web Services (WSDL and SOAP) guarantee only syntactical interoperability between services in a workflow, while what is needed for automating the mentioned processes is interoperability on a semantic level. The Semantic Web provides standards for representing and reasoning with computer interpretable information [2], and this has lead to the development of OWL-S, which is an attempt to standardize how Web Services are described semantically, thus creating Semantic Web Services [3].

OWL-S provides a semantic layer on top of WSDL that maps operation parameters to semantics defined in ontologies. Grid Services are with OGSI [4] and WS-Resources [5] based on Web Services in that the interfaces are defined with WSDL. OWL-S does neither reflect nor interfere with the standardized interfaces for statefulness, so we maintain that OWL-S can be used to describe the application specific interfaces of Grid Services. Semantic Grid Services are therefore in our view Grid Services with semantically annotated capabilities, where OWL-S is an example of a standard that specifies the semantic layer. In this work we will focus on the interfaces of the services and not their statefulness.

It is anticipated that in the future, Grid Services will have associated economical costs with various degrees of quality and performance that can be used for composition and preferred selection of services. Before this higher level information is taken into account, it is important to achieve composition and workflow selection on a fundamental level, where the focus is on the information and state transformation of the services. In this work we represent services by the information transformation they undertake, in the form of input and output parameters which are described by semantics defined in ontologies.

To relieve the user from the unnecessary complexity of specifying exactly which services to use, a semantic description of the information transformation that is needed could be specified in the form of requested capabilities. Based on these requested capabilities a Matchmaker could on-the-fly find the best services and compose them into a workflow. And since semantic interoperability is guaranteed by the semantics, the workflow could be automatically integrated into the client application, which is in our case a Problem Solving Environment (PSE) [7].

We demonstrate in this paper the significance of Semantic Web Services in a PSE and show how a domain ontology is developed to annotate services. Furthermore, we demonstrate composition of services with a simple algorithm and we propose a novel algorithm for automated workflow selection.

The rest of the paper is organized as follows. In Section 2 we give a detailed description of the scenario of the interactive biomedical application. In Section 3 we introduce the architecture of the Matchmaker by describing its interaction with services, clients and ontologies. In Section 4 we discuss semantics for annotating Semantic Web Services, and in Section 5 we develop a domain ontology to annotate the Semantic Grid Services for the biomedical application. Algorithms for automated composition and workflow selection are presented in Sections 4 and 5, and in Section 8 we demonstrate the algorithms by composing and selecting the biomedical services, and finally in Section 9 we conclude.

2 An Interactive Biomedical Application

Problems where blood arteries are weakened or cluttered are called vascular disorders, and lead to reduced or blocked blood circulation, causing in many cases stroke, and eventually death. Medical data acquired by e.g. Magnetic Resonance Angiography (MRA) may be used by specialists to detect these disorders, and treatment consists of reconstructing defected arteries or adding bypasses so that the blood flow can normalize. The best treatment is however not obvious. Problem Solving Environments (PSE) are therefore developed to allow surgeons conduct pretreatment planning in a virtual 3D environment [7]. The PSE components and Grid Services for this case study are shown in figure 1. The surgeon

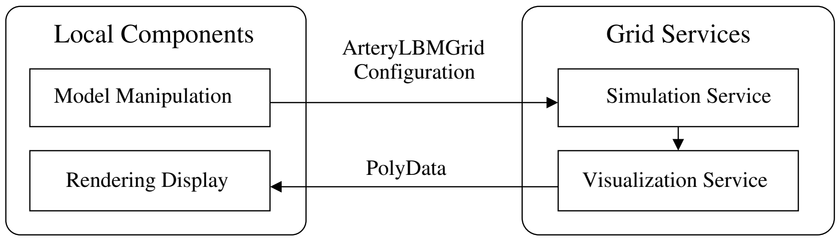

Fig. 1. Dataflow between local components and Grid Services in the PSE.

works with the Model Manipulation component by inserting or modifying a bypass around the defected part of the blood artery, and when finished submits the data of the artery's geometry (ArteryLBMGrid) to the Blood Flow Simulation Service according to the also provided Configuration.

The Simulation Service produces a dataset (RawVisualizationData) that is sent to a Visualization Service where it is converted into 3D polygons (PolyData), which is then sent to the local rendering component where a visualization of blood flow is displayed to the surgeon. This procedure is repeated until the desired effects are obtained.

The user specifies in the PSE that an information transformation that takes ArteryLBMGrid and Configuration as inputs, and that returns PolyData is needed. This is communicated to the Matchmaker, which tries to find a sequence of services that provides this transformation.

3 Matchmaker Architecture

Figure 2 shows the architecture for the Matchmaker, adapted from [1], depicting the involved parties and their events of interaction. Ontologies are used to annotate capabilities of services that are advertised to the Matchmaker, and then to annotate requested capabilities specified by the client, and based on these requested capabilities the Matchmaker composes and selects the workflows that satisfy the requirements. This involves retrieving the necessary ontologies from the Web, and semantically matching the capabilities using an inference engine. The Matchmaker returns then a proposed workflow of services to the client, that is used to invoke the services.

The Matchmaker maintains the list of advertised services in a local database, and by using the OWL-S API library from The University of Maryland, the capabilities, described by Profile and Process, are parsed for input and output

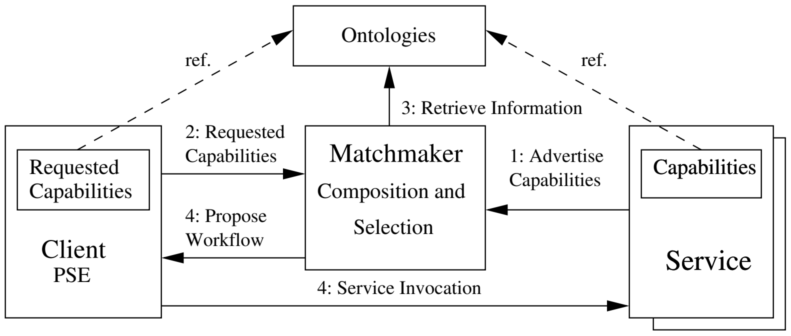

Fig. 2. Architecture with sequence of interaction(solid arrows) and references to ontologies(stippled arrows).

parameters consisting of URIs to ontologies on the Web. The reasoning engine (Racer [12]) loads these ontologies and compares them for equivalence and subsumption.

4 Semantic Data Structures

Input and output parameters of services are represented by semantics defined in domain ontologies, where the ontologies describe the parameters by specifying what information that has to be provided and communicated as part of the parameter. Each parameter points to one concept (core concept) in the ontology, that specifies the necessary information for the parameter, where the information consists of concepts, properties and data values. This is achieved by defining concepts and properties, and then relating the core concept to the concepts and properties through axioms like equivalence and subsumption, specified in OWL as equivalentClass and subclassOf [8], respectively.

Properties, either relating a two concepts or a concept to a data value, are defined with restrictions like allValuesFrom and cardinality, respectively specifying existential qualification of the range-concept and qualified exactly cardinality for the property [8].

The ontologies define the structure that the information has to conform to, and this differs significantly from structure verifications provided by XML Schemas, because the interpretation of the information is based on an ontology that allows complex relations and constraints. Thus, communication with such structures becomes more flexible in that there might be more than one way of satisfying the constraints.

To the best of our knowledge, there exists no adequate name for such semantic structures, thus we coin the name Semantic Data Structure (SDS) to define a group of concepts and relations that are required to satisfy the constraints of a core concept.

A domain ontology may consists of several SDSs with equally many sub concepts and data values. They can therefore grow large and become very complex. Requested and advertised capabilities are annotated with core concepts, and manually finding these concepts become thus tedious and difficult, and the reusability of ontologies is lowered. Core concepts should therefore be distinguished from other concepts on a semantic level, so that e.g. computer based design tools can be used to select and present core concepts to the user. We propose therefore that core concepts should be subsumed by a certain concept (say CoreConcept) defined in a standardized ontology for Semantic Web Services like OWL-S, thus in this way improving the reusability of domain ontologies. It is furthermore tempting to specify core concepts as inputs and outputs in the domain ontology [9], but outputs for one service might be inputs for another service so that would restrict reusability.

Usage of core concepts can be enforced by only allowing parameters refer to concepts that are subsumed by CoreConcept. For OWL-S, this could be achieved by specifying a range restriction for the property parameterType [3].

5 Domain Ontology

Figure 3 shows fragments of a domain ontology defined to annotate the Grid Services in the biomedical application. The ontology is described with Description Logics notation [11] because of its more compact form than OWL. ConfigLight is a SDS defined as CoreConcept with a property Viscosity, where the data property is restricted to Integer as range, and cardinality of one. ConfigFull is then defined as equal to ConfigLight with ReynoldsNumber as data property. ConfigFull is a specialization of ConfigLight and thus derives property Viscosity and concept CoreConcept. ConfigFull can thus be used in replacement of ConfigLight because the former specifies stricter restrictions and includes more information than the latter. PolyData and PolyDataExtra are defined with similar relation.

$$
\begin{aligned}
&\texttt{ConfigLight} && \equiv \texttt{CoreConcept} \sqcap \forall\ \texttt{Viscosity.Integer} \sqcap \equiv 1\texttt{Viscosity} \\
&\texttt{ConfigFull} && \equiv \texttt{ConfigLight} \sqcap \forall\ \texttt{ReynoldsNumber.Integer} \sqcap \equiv 1\texttt{ReynoldsNumber} \\
&\texttt{ArteryLBMGrid} && \equiv \texttt{CoreConcept} \sqcap \forall\ \texttt{LocatedAt.URI} \sqcap \equiv 1\texttt{LocatedAt} \\
&\texttt{RawVisualizationData} && \equiv \texttt{CoreConcept} \sqcap \forall\ \texttt{LocatedAt.URI} \sqcap \equiv 1\texttt{LocatedAt} \\
&\texttt{RawVisData} && \equiv \texttt{RawVisualizationData} \\
&\texttt{PolyData} && \equiv \texttt{CoreConcept} \sqcap \forall\ \texttt{LocatedAt.URI} \sqcap \equiv 1\texttt{LocatedAt} \\
&\texttt{PolyDataExtra} && \equiv \texttt{PolyData} \sqcap \forall\ \texttt{NumberOfFrames.Integer} \sqcap \equiv 1\texttt{NumberOfFrames}
\end{aligned}
$$

Fig. 3. Fragments of a domain ontology.

To demonstrate mapping between concepts, possibly defined in different ontologies, we define RawVisData as equivalent to RawVisualizationData. These core concepts can then be used interchangeable, because RawVisData inherits the property LocatedAt from RawVisualizationData.

SDSs are defined with equivalence axioms and not subsumption. In this way is it sufficient, and not only necessary, to satisfy the concepts and relations on the right-hand side to be a valid member of the SDS. This makes semantic matching more flexible in that different SDSs can refer to the same right-hand side concepts and relations, and thus match as Exact.

6 Automated Composition

In our work, services are composed into workflows based on the information transformation they undertake in the form of input and output parameters. Outputs can either be condition or unconditional, but we are currently for simplicity only considering the latter, thus in this way creating simple workflows that are directly executable without the need for human intervention or machine reasoning during execution.

Service capabilities, requested or advertised, consist of input and output parameters that each refer to a core concept in an ontology. These capabilities are semantically matched as described by Paolucci et al. [10], where valid matching types of SDSs are Exact and PlugIn. Exact match means that the SDSs are interchangeable and equal, while PlugIn match means that an inverse subsumption relationship between the SDSs exists, and which depend whether it applies to input or output parameters. A PlugIn match for an advertised input parameter means that the service requires less information as input then what is asked for or is provided to the service, and for advertised output parameters it means that the service provides more output than what is asked for. PlugIn matches are therefore defined as weaker than Exact matches.

The overall match between two capabilities, either between requested and advertised capabilities, or between capabilities of two succeeding services in a workflow, is determined by the weakest match of the parameters.

Compositions are made by first finding all services with inputs that match with requested inputs, and then in forward chaining fashion, adding services with inputs that match with the outputs of the last service in the workflow. This leads to graphs of workflows where each branch represents a workflow, and composition continues until no more matches among the unused services can be found (each branch holds its own list of unused services), or the outputs of the last service in the workflow match with the requested outputs.

Forward chaining is equally good as backward chaining when conditional outputs are not taken into account, but if they are to be considered then a backward chaining, or goal driven approach, would have been preferred because of the asymmetries introduced by the conditional outputs.

7 Automated Workflow Selection

Several workflows might satisfy the requested capabilities specified in the PSE and the problem is then to autonomously select the most optimal one. We present here an algorithm that evaluates and assigns a cost to each workflow so that they can be compared and thus one workflow selected. Evaluations are based on the types of semantic matches the workflows consist of and the following observations:

- Workflows with as few services as possible are preferred, because services might be located far from each other geographically and communication latency can lower overall performance.
- Exact matches are in general preferred over PlugIn matches, because with services that match as PlugIn there might be information produced that is not used and this might affect the performance negatively.
- Inputs of the first service in the workflow that match as PlugIn with the requested inputs might affect the quality of the outcome differently than PlugIn matches elsewhere in the workflow and should therefore be considered separately. A PlugIn match of the first inputs is considered worse than a PlugIn match in the middle or at the end. This is based on the assumption that what the client requests as input is important and will influence the execution of the workflow. As an example consider a request for a data conversion from one format to another including a certain parameter specifying the conversion. A workflow taking this parameter into account would most likely bring about a conversion closer to what is requested than a workflow not taking this parameter into account.

Equation 1 shows the function that calculates the cost for each workflow, where the inputs and outputs of services are evaluated and given values based on their type of semantic matches. The overall match of the inputs of the first service in the workflow is evaluated and then added to the summation of the matches of the outputs of all N services, where n denotes the service number. Evaluating the input match of the first service separately enables us to specify a matching value uniquely, and for succeeding services in the workflow only the outputs need to be accounted, because an output match is equal to the input match of the next service.

The workflow with the lowest cost is chosen, thus the selection is based on the types of semantic matches and the number of services.

$$\begin{aligned} Cost_{wf} &= Inputs_1(match) + \textstyle\sum_{n=1}^{N} Outputs_n(match), \\ Inputs(match) &= \begin{cases} 1 \text{ if match} = \text{Exact;} \\ 2 \text{ if match} = \text{PlugIn.} \end{cases} \\ Outputs(match) &= \begin{cases} 1 \quad \text{if match} = \text{Exact;} \\ 1.5 \text{ if match} = \text{PlugIn.} \end{cases} \end{aligned} \tag{1}$$

8 Demonstration

In the PSE it is specified that a composition is requested that takes ArteryLBMGrid and ConfigFull as inputs, and that produces PolyData as output. Five different compositions that satisfy these requested capabilities are displayed in figure 4. The workflows consist of two simulation services (FlowSim1 and FlowSim2), two visualization services (Vis1 and Vis2) and one service that simulate and visualize (FlowSimVis).

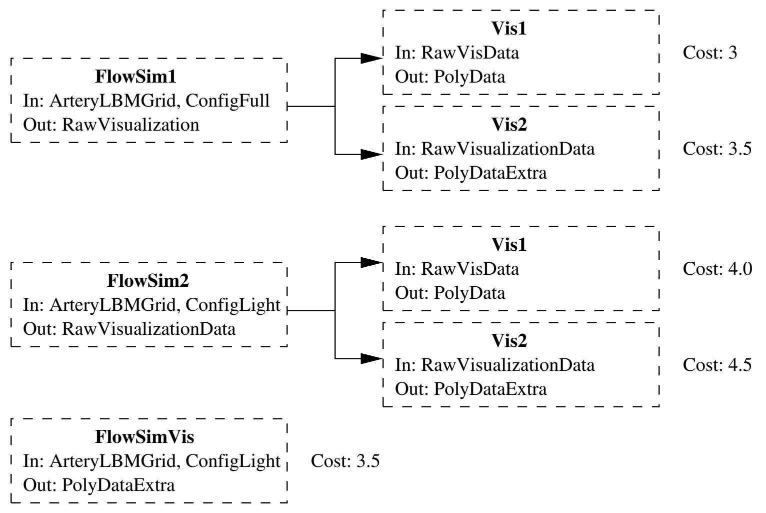

Fig. 4. Workflows with assigned costs.

The inputs of FlowSim1 match as Exact for the requested inputs, while the inputs of FlowSim2 and FlowSimVis match as PlugIn because ConfigLight subsumes ConfigFull (inverse subsumption match). All the matches between simulation and visualization services are Exact, because RawVisualizationData is equal to RawVisData in the domain ontology. And the outputs of Vis1 match as exact with the requested outputs, while outputs of Vis2 and FlowSimVis match as PlugIn because PolyData subsumes PolyDataExtra.

The cost of each workflow is calculated with the algorithm in section 3, and the workflow with cost 3 is selected. This workflow is preferred even over the workflow consisting of only FlowSimVis because it brings about a more exact semantic transformation of information.

9 Conclusions and Future Work

This paper identifies a class of applications where Semantic Web Services will be indispensable, in that they enable automated discovery, composition and selection of services for direct integration, without human involvement, into the Problem Solving Environment.

We emphasize that each service input and output parameter refers to a concept in a domain ontology that specifies the semantic description of the parameter. Communicating the information of the parameter require that the concepts and properties defined by this concept is satisfied. We describe this information therefore as a Semantic Data Structure, because it differs from XML Schemas in that interpretation is based on an ontology.

We argue that concepts representing service parameters must be distinguished on a semantic level by sub-classing a certain predefined concept. This enables design tools, and possible agents, to better select the right concepts to annotate requested capabilities, thus improving the reusability of domain ontologies. We propose an extension to OWL-S to enforce such a restriction.

We present an algorithm for automated workflow composition where services are composed into workflows in forward-chaining fashion. The parameters are semantically matched as either Exact or PlugIn, and the overall match between services is determined by the weakest parameter match.

We propose an approach for selecting the best workflow from a set of alternatives, where the quality and cost of each workflow is estimated, based on the types of semantic matches, and number of services, involved.

The algorithms are demonstrated within the context of a biomedical application by composing and selecting Semantic Grid Services that are annotated with a domain ontology.

Future works involves applying the framework to statefull Web Services such as described in the Web Service Resource Framework [5].

References

1. Massimo Paolucci, Katia Sycara, and Takahiro Kawamura. Delivering Semantic Web Services. In Proceedings of the Twelve's World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003, pp 111- 118
2. Grigoris Antoniou and Frank van Harmelen. A Semantic Web Primer. The MIT Press, 2004.
3. Dean, M. (ed). OWL-S: Semantic Markup for Web Services. Version 1.0, 2004.
4. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C, Maguire T., Sandholm, T., Snelling, D., and Vanderbilt, P. Open Grid Services Infrastructure (OGSI) Version 1.0.
5. I. Foster (ed.). Modeling Stateful Resources with Web Services v. 1.1. March 5, 2004.
6. Z. Zhao; G.D. van Albada; A. Tirado-Ramos; K.Z. Zajac and P.M.A. Sloot: ISS-Studio: a prototype for a user-friendly tool for designing interactive experiments in Problem Solving Environments, in P.M.A. Sloot; D. Abrahamson; A.V. Bogdanov; J.J. Dongarra; A.Y. Zomaya and Y.E. Gorbachev, editors, Computational Science - ICCS 2003, Melbourne, Australia and St. Petersburg, Russia, Proceedings Part I, in series Lecture Notes in Computer Science, vol. 2657, pp. 679-688. Springer Verlag, June 2003. ISBN 3-540-40194-6
7. P.M.A. Sloot; A. Tirado-Ramos; A.G. Hoekstra and M. Bubak. An Interactive Grid Environment for Non-Invasive Vascular Reconstruction. 2nd Interna-tional Workshop on Biomedical Computations on the Grid (BioGrid'04), in con-junction with Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)
8. OWL Web Ontology Language. W3C Recommendation 10 February 2004.

9. Li, Lei and Horrocks, Ian. A Software Framework for Matchmaking Based on Semantic Web Technology. In Proceedings International WWW Conference, Budapest, Hungary. (2003)
10. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic Matching of Web Services Capabilities. The First International Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.
11. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. The Description Logic Handbook. Cambridge University Press, 2002.
12. Volker Haarsley and Ralf Moller. RACER User's Guide and Reference Manual Version 1.7.7