Type based service composition

Ion Constantinescu Boi Faltings Walter Binder
Artificial Intelligence Laboratory
Swiss Federal Institute of Technology
IN (Ecublens), CH–1015 Lausanne (Switzerland)
{ion.constantinescu,boi.faltings,walter.binder}@epfl.ch

ABSTRACT

Service matchmaking and composition has recently drawn increasing attention in the research community. Most existing algorithms construct chains of services based on exact matches of input/output types. However, this does not work when the available services only cover a part of the range of the input type. We present an algorithm that also allows partial matches and composes them using *switches* that decide on the required service at runtime based on the actual data type. We report experiments on randomly generated composition problems that show that using partial matches can decrease the failure rate of the integration algorithm using only complete matches by up to **7 times** with no increase in the number of directory accesses required. This shows that composition with partial matches is an essential and useful element of web service composition.[1]

Categories and Subject Descriptors: H.3.5 Online Information Services : Web-based services, D.2.12 Interoperability : Distributed objects, D.2.m Miscellaneous : Reusable software.

General Terms: Design, Management.

Keywords: web services, large scale discovery, type based composition, partial matches, runtime non-determinism.

1. AUTOMATIC SERVICE COMPOSITION

Service composition is an exciting area which has received a significant amount of interest in the last period.

Initial approaches to web service composition [**?**] used a simple forward chaining technique which can result in the discovery of large numbers of services.

There is a good body of work which tries to address the service composition problem by use planning techniques based either on theorem proving (e.g., Golog and [**?**, **?**] or on hierarchical task planning (e.g., SHOP-2 [**?**]). The advantage of this kind of approach is that complex constructs like loops (Golog) or processes (SHOP-2) can be handled. All these approaches assume that the relevant service descriptions are initially loaded into the reasoning reasoning engine and that no discovery is performed during composition.

Recently Lassila [**?**] has addressed in more detail the problem of interleaving discovery and integration but has considered only simple workflows where services have one input and one output.

In this paper we are concerned by a particular combination of issues that is specific and unique to the web services context:

1. **discovery in large scale directories** - we assume that a large number of available web services will be stored in (possibly distributed) directories. How should we discover exactly the services that are relevant at each step of our composition process?

2. **runtime non-determinism** - when discovered services do not match completely[2], the reasoning engine will have to aggregate several services in order to fulfill the required functionality. The actual flow of messages will be routed based on runtime parameters on the appropriate paths. How can we discover and create those switches and how can we make sure that they handle correctly all possible combinations of parameter values?

Other challenging issues are not addressed in these this paper but are considered for future work: behavior based integration, dealing with side-effects and changes of the world that are not under the control of the composition engine, knowledge engineering issues, to enumerate only a few.

This paper contributes with solutions to these two issues. As a first contribution we present an algorithm that interleaves the discovery and composition process by using a partial order planning approach that focuses the directory searches. Our second contribution is a technique for discovering and composing services with partial type compatibility. In our approach we discretize the space of possible parameter values and we use a greedy method to incrementally reduce the space of values that cannot be handled. Then partially matching components are assembled into switches that can route the flow of messages on the appropriate paths based on runtime values.

2. COMPUTING SERVICE INTEGRATION PLANS

We consider a service integration query specified in terms of a set of available input parameters and a set of required output parameters. An integration solution will consist of a given ordering of services that can be invoked such that finally all parameters required by the query are known.

[1]The work presented in this paper was partly carried out in the framework of the EPFL Center for Global Computing and was supported by the Swiss National Science Foundation as part of the project MAGIC (FNRS-68155), as well as by the Swiss National Funding Agency OFES as part of the European projects KnowledgeWeb (FP6-507482) and DIP (FP6-507483).

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

[2]The **subsume** match type identified by Paolluci [**?**] and the **intersection** or **overlap** match type identified by Li [**?**] and Constantinescu [**?**].

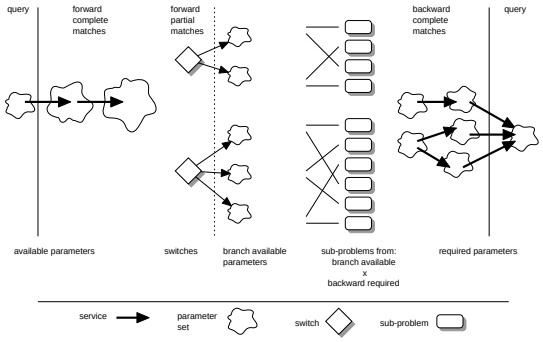

Figure 1: A service query matching type perspective on service integration.

For solving the problem we first discover and apply forward and backward complete matches. Applying backward completely matching services creates a directed graph of sets of required parameters as the order in which different parameters can be applied affects the set of parameters that still need to be provided.

Several forward partially matching services can be aggregated together into a composite service as a software switch that maps each possible combination of parameter values from the space of available inputs to one or more *partially matching* services. In order to be able to fulfill the same functionality as the *completely matching* service, we have to have for each possible range combination of input parameters one or more services that can accept those values. For determining which of the discovered switches can be used for the final solution one has to make sure that all switch branches care correctly provide the required parameters. For that sub-problems can be generated as a cross product between the output parameters provided by each switch branch and the parameters required by any of the backward required inputs. For each branch at least one of such problems will have to lead to a solution in order for the switch to function correctly.

3. EVALUATION AND ASSESSMENT

For evaluation purposes we have defined an abstract domain (see Fig. **??**) where we consider a number of layers that define sets of parameter names. Services are defined as transformations between parameters in adjacent layers and problems are defined between parameters of the first and last layer. For example, a possible service between layers1-2 with the parameters A, B could have as input the types A=a1, B=b1,b2 and for the output parameters C and D could have as types C=c2, c3 and D=d1, d2. For the input parameters A, B the query could have the types A=a1,a2, B=b2 and for the output parameters E,F the types E=e1, F=f2.

We have randomly generated services and problems which we have solved using first an algorithm that handles only *complete* type matches and then an algorithm that handles *partial* type matches (and obviously includes *complete* matches). We have measured the number of directory accesses (Fig. **??** (a)) and the failure ratio of

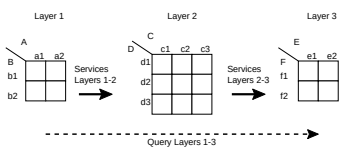

Figure 2: The layered domain - simplified example.

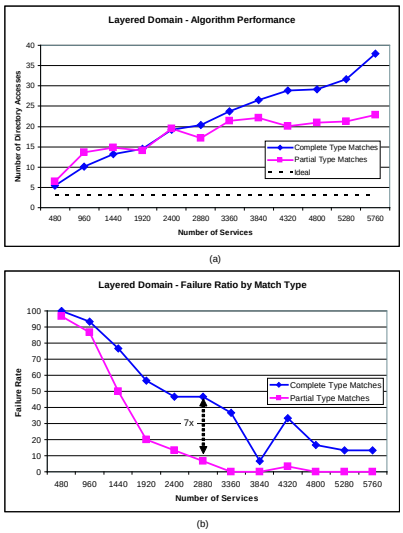

Figure 3: The layered domain.

the integration algorithms (Fig. **??** (a)).

Experiments with randomly generated problems show that such partial matches bring significant gains in the range of problems that can be solved by automated composition with a given set of services (reducing the failure rate by factor **7**). Furthermore, it appears that this comes at no increase in the complexity as measured by the number of accesses to service directories. Thus, we consider partial matches to be an essential element of any future service composition algorithm.

4. REFERENCES

[1] I. Constantinescu and B. Faltings. Efficient matchmaking and directory services. In *The 2003 IEEE/WIC International Conference on Web Intelligence*, 2003.

[2] O. Lassila and S. Dixit. Interleaving discovery and composition for simpleworkflows. In *Semantic Web Services, 2004 AAAI Spring Symposium Series*, 2004.

[3] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web*.

[4] S. McIlraith, T. Son, and H. Zeng. Mobilizing the semantic web with daml-enabled web services. In *Proc. Second International Workshop on the Semantic Web (SemWeb-2001)*, Hongkong, China, May 2001.

[5] S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 482–496, San Francisco, CA, Apr. 22–25 2002. Morgan Kaufmann Publishers.

[6] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, 2002.

[7] S. Thakkar, C. A. Knoblock, J. L. Ambite, and C. Shahabi. Dynamically composing web services from on-line sources. In *Proceeding of the AAAI-2002 Workshop on Intelligent Service Integration*, pages 1–7, Edmonton, Alberta, Canada, July 2002.

[8] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, October 2003.