Definition and Execution of Composite Web Services: The SELF-SERV Project

Boualem Benatallah
School of Computer Science & Engineering
The University of New South Wales
Sydney NSW 2052, Australia
boualem@cse.unsw.edu.au

Marlon Dumas
Centre for IT Innovation
Queensland University of Technology
GPO Box 2434, Brisbane Q 4001, Australia
m.dumas@qut.edu.au

Zakaria Maamar
College of Information Systems
Zayed University
PO Box 19282, Dubai, U.A.E
zakaria.maamar@zu.ac.ae

Abstract

Web services composition is emerging as a promising technology for the effective automation of business-to-business collaborations. It allows organizations to form alliances by connecting their applications, databases, and systems, in order to offer "one-stops shops" for their customers. The SELF-SERV project aims at providing tool support and middleware infrastructure for the definition and execution of composite Web services. A major outcome of the project has been a prototype system in which Web services are declaratively composed, and the resulting composite services can be orchestrated either in a peer-to-peer or in a centralized way within a dynamic environment. Work is underway to extend this system in order to enable user-driven composition of Web services, and their execution in a mobile environment.

1 Introduction

The growth of Internet technologies has unleashed a wave of innovations that are having tremendous impact on the way organizations interact with their partners and customers. In particular, Web services composition is emerging as a new model for automated interactions among distributed and heterogeneous applications. This approach is an alternative to hand-coding an integrated application. A Web Service is defined as a modular and self-described application that uses standard Web technologies to interact with other services [8, 1]. An example of a Web service is bidding in an auction through a SOAP interface.

SELF-SERV is an ongoing research project that aims at providing tool support and middleware infrastructure for facilitating the composition of Web services in large, autonomous, heterogeneous, and dynamic environments. More precisely, the SELF-SERV system considers the following key aspects related to service

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

composition. Firstly, *declarative service composition* approaches are required to enable rapid development and flexible adaption of composite services. Secondly, the number of services to be composed may be large and continuously evolving. Consequently, approaches where the development of a composite service requires understanding and establishing interactions among component services at service-definition time are inappropriate. Instead, a divide-and-conquer approach should be adopted, whereby services providing similar capabilities (also called *alternative services*) are grouped together, and these groups take over some of the responsibilities of service composition. Thirdly, given the highly distributed nature of services, and the large number of network nodes that are capable of service execution, we believe that novel mechanisms involving scalable and completely decentralized execution of services will become increasingly important. Specifically, it should be possible to execute composite services under different communication topologies: peer-to-peer, centralized, and hybrid.

At present, the technological infrastructure for Web Services is structured around three major standards: SOAP, WSDL, and UDDI [4]. These standards provide building blocks for the description, discovery, and invocation of Web services. Other proposed standards such as BPEL4WS, WSCI, WS-Coordination, and WS-Transaction (`dev2dev.bea.com/techtrack/standards.jsp`), layer up functionality related to composition and transactions on top of the three basic standards.

The SELF-SERV system leverages these standards (especially SOAP, WSDL, and UDDI) and builds upon lessons learned in previous related work (e.g. eFlow [3], CMI [6]), in order to provide a scalable service middleware through which Web services can be declaratively composed, and the resulting composite services can be enacted according to different communication topologies within a dynamic environment. The rest of the paper provides an overview of the concepts (Section 2), architecture (Section 3), and planned extensions (Sections 4 and 5) of the SELF-SERV system.

2 Web Service Composition in SELF-SERV

A composite Web service is an umbrella structure that aggregates multiple other elementary and composite Web services, which interact according to a given process model. For example, a composite Web service "Travel Planner" may aggregate multiple Web services for flight booking, travel insurance, accommodation booking, car rental, itinerary planning, etc., which are executed sequentially or concurrently.

In SELF-SERV, the process model underlying a composite service is specified as a statechart [5] whose states are labeled with invocations to Web services, and whose transitions are labeled with events, conditions, and variable assignment operations. Statecharts possess a formal semantics and offer most of the constructs found in contemporary process modeling languages (sequence, branching, structured loops, concurrent threads, inter-thread synchronization, etc.). This ensures that the service composition mechanisms and orchestration techniques developed within the SELF-SERV project can be adapted to other process modeling languages for Web Services such as BPEL4WS and WSCI.

SELF-SERV exploits the concept of *service community* in order to address the issue of composing a potentially large and changing collection of Web services. Service communities are essentially containers of services that provide a common capability. They provide descriptions of desired services (e.g., flight booking) without referring to any actual provider (e.g., UA flight-booking Web service). When a community receives a request for executing an operation, it delegates it to one of its current members. The choice of the delegatee is based on a selection policy involving parameters of the request, the characteristics of the members, the history of past executions, and the status of ongoing executions. The set of members of a community can be fixed when the community is created, or it can be determined through a registration mechanism, thereby allowing service providers to join, quit, and reinstate the community at any time.

Communities are services themselves. In particular, their operations can be invoked by a composite Web service. In this way, dynamic provider selection and even some forms of dynamic planning can be achieved. Specifically, by labeling a state of a composite service with an invocation to an operation provided by a community, the selection of the actual service which is to execute this invocation, is delayed until the last possible moment, thereby allowing the decision to be taken according to the current execution status. Similarly, when there are several ways (or plans) for executing a "portion of a composite service", it is possible to select at run-time which of these plans is the most appropriate given the current context (e.g., choose the fastest way if time is important, or the cheapest otherwise). To achieve this, each of the alternative plans is modeled as a composite service, and all these composite services are grouped in a single community which can be invoked from within an encompassing composite service. The decision as to which of the available plans to adopt (i.e., which of the composite services in the communities should be invoked), is taken at the last possible moment.

The execution of a composite service in SELF-SERV is orchestrated by a collection of software components called *state coordinators*. One coordinator is attached to each state of a composite service. The coordinator of a state is in charge of initiating, controlling, and monitoring this state, as well as collaborating with its peers to orchestrate the composite service execution. In particular, the coordinator of a state S is responsible for determining: (i) when should S be entered for a given execution of the composite service; (ii) which other states may potentially need to be entered after S is exited and which data items should be sent to the coordinators of these states; and (iii) how to handle events such as cancellation or timeouts, while the service invocation labeling the state S is underway. The knowledge required at runtime by each of the coordinators of a composite service (e.g. location, peers, and control flow routing policies) is statically extracted from the composite service's statechart and represented in a simple tabular form (as *routing tables*). In this way, the coordinators do not need to implement any complex scheduling algorithm.

By distributing the responsibility of orchestrating a composite service across several state coordinators, SELF-SERV enables both peer-to-peer and centralized orchestration. Peer-to-peer orchestration is achieved by placing each state coordinator in the site of the Web service that is invoked when the corresponding state is entered. The state coordinators are therefore placed in different physical nodes (the sites of the components of the composite service), and they interact in a peer-to-peer way. Centralized orchestration is achieved by placing all the state coordinators in the host of the composite service. The collection of state coordinators placed in a single physical node can then be seen as a central scheduler, which remotely invokes the components of the composite service. Hybrid approaches can be achieved by placing some state coordinators in the host of the composite service provider, and others in the hosts of the Web services participating in the composition.

3 SELF-SERV Architecture and Implementation

The SELF-SERV architecture (Figure 1) consists of three modules, namely the *service builder*, the *service deployer*, and the *service discovery engine* [7]. These modules have been implemented using Java and the IBM Web Services Toolkit (`alphaworks.ibm.com/tech/webservicestoolkit`).

The service builder facilitates the creation and configuration of composite services and service communities. It provides an editor for describing registration modes and selection policies of service communities, as well as an editor for drawing and annotating the statechart diagrams of composite service operations. Annotated statecharts are translated into XML documents for subsequent processing by the service deployer.

The service deployer generates the routing tables of every state of a composite service statechart using the algorithms presented in [2]. The input of this generation process are statecharts represented in XML, while the outputs are routing tables formatted in XML as well. Once generated, routing tables are uploaded into the sites of the corresponding component services (for peer-to-peer orchestration), or into the site of the provider of the composite service (for centralized orchestration). Peer-to-peer orchestration requires in addition that the participating service providers download and configure a class called `Coordinator`, which implements the concept of state coordinator. This class contains the code required to load and interpret a routing tables, and to communicate with other coordinators in order to orchestration composite service executions. Centralised orchestration on the other hand does not require the service providers participating in the composition to install

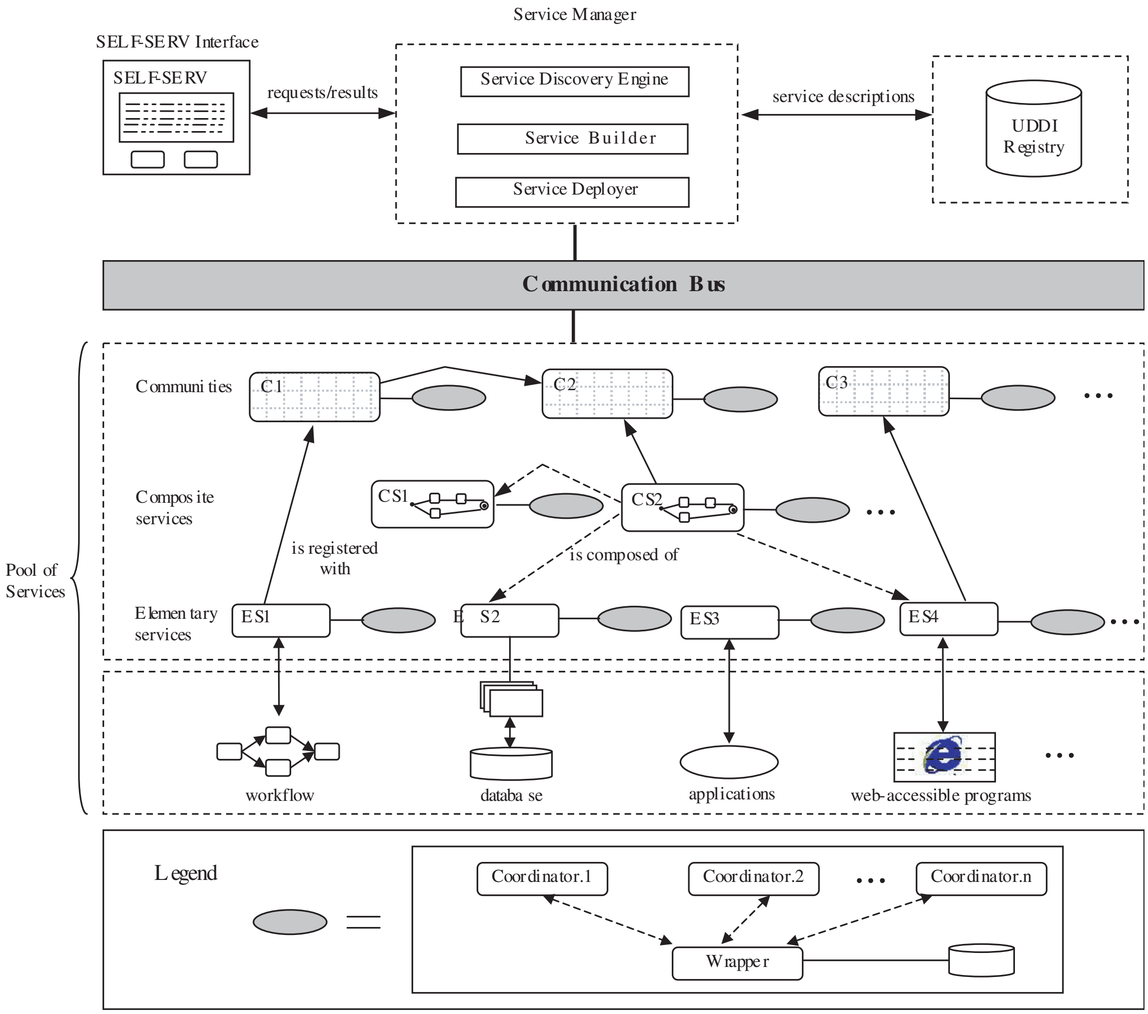

Figure 1: Architecture of SELF-SERV.

any SELF-SERV-specific runtime environment: they only need to provide their Web services through SOAP.

The SELF-SERV prototype has been used to experimentally compare the performance of the peer-to-peer orchestration model with respect to the centralized one. The experimental results have shown that peer-to-peer orchestration leads to less message exchanges in average, better load distribution among the participants, and as a result, lower execution times.

4 User-Driven Web Services Composition

Many Web sites that presently deliver their services through HTML Web pages, are likely to provide programmatic access to these services in the near future by exploiting emerging Web Services standards. Google, eBay, and Amazon are examples of popular Web sites that have already made some of their functionalities available through programmatic interfaces based on SOAP and WSDL[1]. As this trend continues to unfold, Web services with dual (interactive and programmatic) interfaces will become a currency. This will in turn open the possibility for dual navigation, whereby a set of interactions between a user and one or several Web sites can be assimilated to a composite Web service, and vice-versa.

[1]See `http://www.xmethods.com` for other examples of Web services with HTML-based interfaces that have been made available through SOAP/WSDL.

This idea is better explained through a simple scenario. A user opens the Web page of a stock quote site using his/her browser. (S)he enters a stock symbol in an input field, clicks a submit button, and obtains the latest quote for this stock in US Dollars. (S)he then opens a currency converter Web page, enters the quote obtained from the previous operation into an input field, selects the currency code "USD" in a menu, the currency code "AUD" in another menu, clicks the submit button, and obtains the stock quote in Australian Dollars. If the currency converter or the stock quote service are unavailable, (s)he will try using alternative ones. This sequence of navigation operations are expected to be repeated often, so the user would like to build his own Web page, that will allow him/her to enter a stock symbol in an input field, and obtain the quote in Australian Dollars in an output field, by reproducing the above process. An approach to achieve this outcome would be to map the sequence of navigation operations performed by the user to an equivalent composite Web service with two sequential states: one that invokes an operation over a community of stock quote services, and another one that invokes an operation over a community of currency converter services. The input of this composite Web service will be the stock symbol, and the output would be the quote in Australian Dollars.

More generally, in the context of Web services with dual interfaces, it is conceivable to specify composite Web services by dragging and dropping elements from several Web pages into a *composition panel*, and relating these elements through control and data flow relationships (e.g. the output of one element is given as input to another). With some simplifying restrictions, these manipulations can be performed by non-expert users, opening the door for the widespread use of composite Web service technology as a means to enable user-driven integration of Web services. An application of this type of integration can be found in account aggregation, where users perform repetitive navigation tasks in order to obtain a complete view of their accounts and investments.

Within SELF-SERV, we are developing a front-end tool to facilitate this kind of user-driven composition of Web services. Specifically, the tool will enable users to drag-and-drop elements from multiple Web pages (especially input and output fields within Web forms), in order to specify invocations to elementary services, which can then be grouped and linked together to form communities and composite Web services. In order to connect elements of Web pages to operations of Web services, SELF-SERV requires that the provider of services with dual interfaces defines links between the programmatic and the interactive interfaces. For example, a submit button in a Web form can be mapped to a Web service operation described in WSDL, while a form input element can be mapped to an input parameter of an operation. Such links can be expressed using RDF triples, where the "subjects" of the triples are XPath expressions designating an XHTML element such as a button or an input field, while the "objects" are Web service operations or input/output parameters.

Having specified a set of invocations to elementary services, the user can then group these services into communities. In the previous example, the currency converters could be grouped in a community, and a selection policy could be specified by providing an order over this set: the community would then invoke the first service in this order, unless it is unavailable, in which case the next service in the order would be selected, and so on.

The front-end composition tool will also allow the user to specify control and data-flow relationships between the elementary services and communities created in the previous steps by means of a visual language based on a subset of statecharts. Specifically, the control-flow operators provided by the composition tool will include sequence, choice, repeat, and parallel branching, while the data-flow will be specified through simple variable assignments and arithmetic expressions. The composition tool will then translate the resulting composite service specification into an annotated statechart, that can be enacted using a subset of the SELF-SERV environment. Since the composite service is intended to be used mainly by its creator, a client-side centralized orchestration model is appropriate, whereby the composite service is orchestrated from the user's machine.

5 Composite Web Services in Mobile Environments

The explosive growth of interconnected computing devices (e.g., PDAs, wireless technologies) enable new environments where ubiquitous information access will be a reality. More importanly, Web services will be accessible from mobile devices [9]. However, existing service provisioning techniques are inappropriate to cope with the requirements of these new environments. Several obstacles still hinder the seamless provisioning of Web services in mobile environments. Examples of such obstacles are: throughput and connectivity of wireless networks, limited computing resources of mobile devices, and risks of communication channel disconnections. We are extending the SELF-SERV architecture to support service provisioning in mobile environments. More precisely, we are investigating the following issues:

- *Context-sensitive service selection*: In addition to criteria such as monetary cost and execution time, service selection should consider the location of requesters and services, and the capabilities of computing resources on which services are executed (e.g., CPU, bandwith). This calls for context-aware service selection policies that enable the system to adapt itself to different computing and user requirements.

- *Handling disconnections during composite service execution*: In a mobile environment, disconnections are frequent (e.g., disconnection due to discharged battery, change of location, or to minimize communication cost). To cope with issues related to client or provider disconnection during a composite service execution, we are investigating an agent-based service composition middleware architecture. In this architecture, users and services are represented by delegate agents, which contain disconnection control policies related to requesters and providers. For example, a user delegate may be used to collect execution results during disconnection of the user's device, and returns these results to the user upon re-connection. Delegate agents reason and act upon evolution regarding user and service devise disconnections.

References

[1] B. Benatallah and F. Casati, editors. Special Issue on Web Services. *Distributed and Parallel Databases, An International Journal*, Kluwer Academic Publishers, 12(2-3), September 2002.

[2] B. Benatallah, M. Dumas, Q.Z. Sheng, and A. H.H. Ngu. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 297–308, San Jose CA, USA, February 2002. IEEE Press.

[3] F. Casati and M.-C. Shan. Dynamic and adaptive composition of e-services. *Information Systems*, 26(3):143–162, May 2001.

[4] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, March 2002.

[5] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[6] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Proc. of the Int. Conference on Advanced Information Systems Engineering (CAiSE)*, Stockholm, Sweden, June 2000. Springer Verlag.

[7] Q. Z. Sheng, B. Benatallah, M. Dumas, and E. Mak. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In *Proc. of the 28th VLDB Conference (Demonstration Session)*, Hong Kong, China, August 2002. Morgan Kaufmann.

[8] G. Weikum, editor. Infrastructure for Advanced E-Services. *IEEE Data Engineering Bulletin*, 24(1), March 2001.

[9] S. J. Vaughas-Nicholas. Web services: Beyond the Hype. *IEEE Computer*, 35(2), February 2002