SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment

Quan Z. Sheng[1] Boualem Benatallah[1] Marlon Dumas[2] Eileen Oi-Yan Mak[1]

[1] School of Computer Science & Engineering
The University of New South Wales
Sydney NSW 2052, Australia
{qsheng, boualem, eileenm}@cse.unsw.edu.au

[2] Centre for Information Technology Innovation
Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
m.dumas@qut.edu.au

1 Introduction

The automation of *Web services* interoperation is gaining a considerable momentum as a paradigm for effective Business-to-Business collaboration [2]. Established enterprises are continuously discovering new opportunities to form alliances with other enterprises, by offering value-added integrated services.

However, the technology to compose Web services in appropriate time-frames has not kept pace with the rapid growth and volatility of available opportunities. Indeed, the development of integrated Web services is often ad-hoc and requires a considerable effort of low-level programming. This approach is inadequate given the size and the volatility of the Web. Furthermore, the number of services to be integrated may be large, so that approaches where the development of an integrated service requires the understanding of each of the underlying services are inappropriate. In addition, Web services may need to be composed as part of a short term partnership, and disbanded when the partnership is no longer profitable. Hence, the integration of a large number of Web services requires scalable and flexible techniques, such as those based on declarative languages. Also, the execution of an integrated service in existing approaches is usually centralised, whereas the underlying services are distributed and *autonomous*. This calls for the investigation of distributed execution paradigms (e.g., peer-to-peer models), that do not suffer of the scalability and availability problems of centralised coordination [3].

Motivated by these concerns, we have developed the *SELF-SERV* platform for rapid composition of Web services [1]. In SELF-SERV, Web services are *declaratively* composed, and the resulting composite services are executed in a *peer-to-peer* and *dynamic* environment. In the following sections we overview the design and implementation of the SELF-SERV system, and sketch the proposed demo.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

2 SELF-SERV Overview

SELF-SERV distinguishes three types of services: *elementary services*, *composite services*, and *service communities*. An elementary service is an individual Web-accessible application (e.g., a Java program) that does not explicitly rely on another Web service.

A composite service aggregates multiple Web services which are referred to as its *components*. SELF-SERV relies on a *declarative* language for composing services based on statecharts: a widely used formalism in the area of reactive systems that is emerging as a standard for process modeling following its integration into the Unified Modeling Language (UML). An operation of a composite service can be seen as having input parameters, output parameters, consumed and produced events, and a statechart glueing these elements together.

SELF-SERV exploits the concept of *service community* in order to address the issue of composing a potentially large number of dynamic Web services. Service communities are essentially containers of alternative services. They provide descriptions of desired services (e.g., providing flight booking interfaces) without referring to any actual provider (e.g., UA flight-booking Web service). At runtime, 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 the parameters of the request, the characteristics of the members, the history of past executions and the status of ongoing executions.

The execution of a composite service in SELF-SERV is coordinated by several peer software components called *coordinators*. Coordinators are attached

to each state of a composite service. They are in charge of initiating, controlling, monitoring the associated state, and collaborating with their peers to manage the service execution. The knowledge required at runtime by each of the coordinators involved in a composite service (e.g., location, peers, and control flow routing policies) is statically extracted from the service's statechart and represented in a simple tabular form called *routing tables*. Routing tables contains *preconditions* and *postprocessings*. Preconditions are used to determine when a service should be executed. Postprocessings are used to determine what should be done after service execution. In this way, the coordinators do not need to implement any complex scheduling algorithm.

3 Implementation

The SELF-SERV architecture (Figure 1) features a *service manager* and a *pool of services*. All of them have been implemented in Java. Services communicate through XML documents. These documents are exchanged through Java sockets. Oracle's XML Parser 2.0 is used for parsing XML documents.

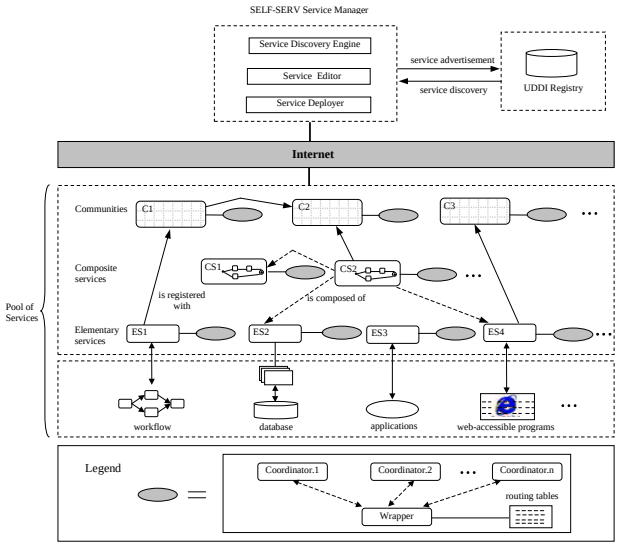

Figure 1: Architecture of SELF-SERV.

The *service manager* consists of three modules, namely the *service discovery engine*, the *service editor*, and the *service deployer*. The *service discovery engine* facilitates the advertisement and location of services. It is implemented using the Universal Description, Discovery and Integration (UDDI), the Web Service Description Language (WSDL), and the Simple Object Access Protocol (SOAP). Service registration, discovery and invocation are implemented as SOAP calls. When a service registers with a discovery engine, a UDDI/SOAP request containing the service description in WSDL is sent to the UDDI registry. After a service is registered in the UDDI registry, it can be located by sending a UDDI/SOAP request (e.g., business name, service type, etc.) to the UDDI registry. In the implementation, we make extensive use of the IBM Web Services Toolkit 2.4 (WSTK2.4) [4], which is a showcase package for Web services emerging technologies. Details about the implementation of the discovery engine are presented in [5].

The administrator of the registered service has to download and install a pre-existing class, namely *Coordinator*, implementing the concept of coordinator. The administrator is also required to build a wrapper for the service by downloading and configuring a class *Wrapper* provided by the SELF-SERV platform. The only infrastructure required to install and configure these classes are standard Java libraries and a JAXP-compliant XML parser. By default, the XML documents containing the routing tables are stored in plain files, so that there is no need to have a DBMS in the site where the installation is performed. Still, the platform can be configured to rely on a DBMS if required.

The *service editor* provides facilities for defining new services and editing existing ones. A service is edited through a visual interface, and translated into an XML document for subsequent analysis and processing by the *service deployer*. The service deployer is responsible for generating the routing tables of every state of a composite service statechart, using the algorithms presented in [1]. The input of the programs implementing these algorithms are statecharts represented as XML documents, while the outputs are routing tables formatted in XML as well. Once the tables are generated, the service deployer assists the service composer in the process of uploading these tables into the hosts of the corresponding component services. It also assists the composer in the deployment of the wrapper of the composite service.

4 Demo Scenario

A travel scenario has been developed using the SELF-SERV platform. The scenario involves several Web services including: *domestic flight booking*, *international flight booking*, *travel insurance*, *accommodation booking*, *attractions search*, and *car rental*. The travel scenario works as follows:

- A traveller books a domestic flight or an international flight, as well as an accommodation.
- A search for attractions is performed in parallel with the flight and accommodation bookings.
- When the search and the bookings are done, a car rental is performed if the major attraction is far from the booked accommodation.

The SELF-SERV platform provides an integrated environment where (i) service providers can register

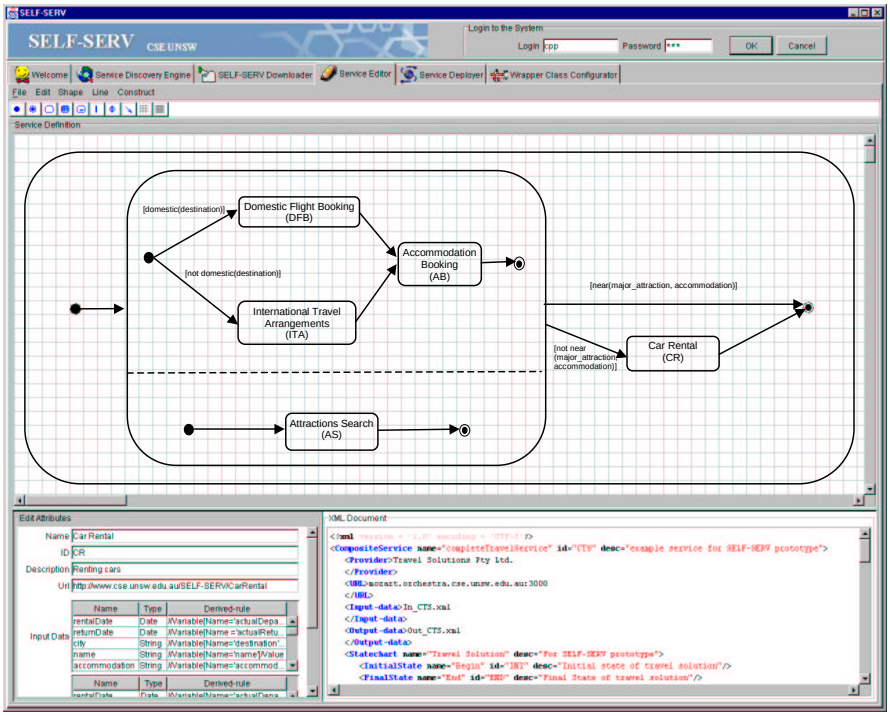

Figure 2: Defining services in SELF-SERV.

their Web services, download, and install the classes `Coordinator` and `Wrapper`, (ii) service composers can edit and deploy composite services, and (iii) end users can locate Web services and execute their operations. We will demonstrate: (i) how to define a composite service for the travel scenario, (ii) how to deploy and register the service, and (iii) how to locate and execute the service using the SELF-SERV platform.

Defining a composite service. The Service Editor offers a *graphical user interface* (GUI) (Figure 2) allowing composers to define composite services. Prior to defining a composite service, the service composer has to search the UDDI registry and find the Web services that will be used as the component services. This is done through the Service Discovery Engine.

A composite service is defined by drawing a statechart diagram (top panel of Figure 2). The information associated with each state or transition (e.g., state ID, state name, input/output parameters of the Web service associated with the state, ECA rule of transition) can be defined in the bottom left panel of Figure 2. After the definition is completed, the service is translated into an XML document (bottom right panel of Figure 2). The composite service of the travel scenario is defined as shown in Figure 2. The component services referenced in the composite service are assumed to have been previously registered with the Discovery Engine. During this registration process, the `Coordinator` and `Wrapper` classes have been downloaded via the Service Deployer, and installed in the hosts of the component services. Among the component services, `Accommodation Booking` is a service community, while others are elementary services.

Deploying and registering a composite service. Once a composite service has been defined, the Service Deployer assists the composer during the deployment process. This process takes as input the XML description of the composite service and involves two steps: (i) generating the control-flow routing tables of each state of the composite service statechart, and (ii) uploading these tables into the hosts of the component services.

The composite service also needs to be registered with the Service Discovery Engine so that it can be located and executed. A service can be published with the UDDI registry using the *Publish* panel offered by the Service Discovery Engine. Before a service can be published, its WSDL descriptions should be created and deployed. This essentially means placing the WSDL descriptions so that they can be retrieved using public URLs. The information of the service (e.g., service name, locations of WSDL descriptions) and of the provider of the service (e.g., provider name, contact data) is entered via the `Publish` panel. When the `Publish` button is clicked, the Service Discovery Engine publishes the service details in the UDDI registry.

Locating and executing a composite service. An end user can locate Web services from the UDDI registry using the *Search* panel offered by the Service Dis-

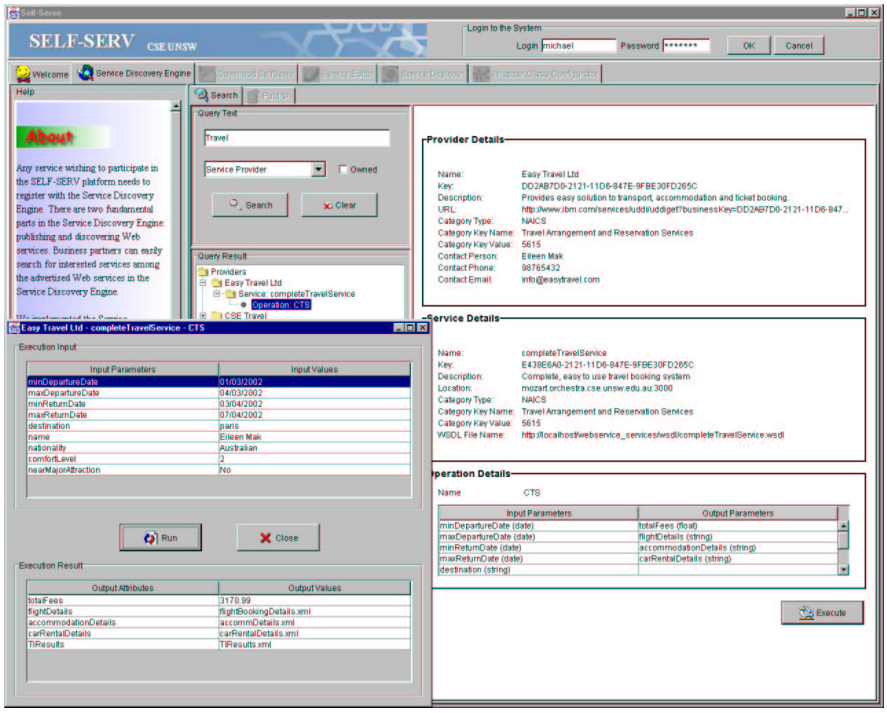

Figure 3: Locating and executing services.

covery Engine (left panel of Figure 3). The user can search Web services by providers, service names or operations (top left of the `Search` panel in Figure 3). The query yields a list of service providers displayed in the bottom left part of the `Search` panel. Each provider is listed with all its services and each service is listed with all its operations. The user can browse these lists and view the detailed information of a given service or operation (right part of the `Search` panel of Figure 3).

An end user can also execute a specific operation of a service by clicking on the `Execute` button in the bottom right of the `Search` panel. An execution window is popped up (bottom left part of the Figure 3), which enables the end user supplying values of the parameters that are needed to execute the service (e.g., customer name, departure date, return date of the travel scenario). After that, the user can click on the `Run` button. An XML document storing the input values is created and sent to the service using the binding details of the WSDL service descriptions. When the wrapper of the composite service receives the document, it sends a message to the coordinator of the state(s) in the statechart which need(s) to be entered in the first place. This/these coordinator(s) invoke their underlying service(s) through the wrapper(s). From there on, the orchestration of the composite service execution is carried out through peer-to-peer message exchanges between the coordinators. Eventually, the coordinators of the states which are exited in the last place send their notification of termination back to the composite service wrapper. At this point, the results of the execution are displayed in the `Execution Result` panel. Details about the peer-to-peer coordination algorithm of SELF-SERV can be found in [1].

References

[1] B. Benatallah, M. Dumas, Q. Z. Sheng, and A. H.H. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proc. of 18th Int. Conference on Data Engineering (ICDE)*, pages 297–308, San Jose, USA, February 2002. IEEE Computer Society.

[2] F. Casati, D. Georgakopoulos, and M. Shan editors. Special Issue on E-Services. *VLDB Journal*, 24(1), January 2001.

[3] Q. Chen and M. Hsu. Inter-Enterprise Collaborative Business Process Management. In *Proc. of 17th Int. Conference on Data Engineering (ICDE)*, pages 253–260, Heidelberg, Germany, April 2001. IEEE Computer Society.

[4] IBM WSTK Toolkit. `http://alphaworks.ibm.com/tech/webservicestoolkit`.

[5] Q. Z. Sheng, B. Benatallah, R. Stephan, E. Oi-Yan Mak, and Y. Q. Zhu. Discovering E-Services Using UDDI in SELF-SERV. In *Int. Conference on E-Business*, Beijing, China, May 2002.