2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement

Automated Web Service Composition System on Enterprise Service Bus*

Xinhuai Tang Sizhe Sun
School of Software
Shanghai Jiao Tong University
Shanghai, China
tang-xh@cs.sjtu.edu.cn; sunsizhe@gmail.com

Xiaozhou Yuan
School of Mechanical and Dynamics Engineering
Shanghai Jiao Tong University
Shanghai, China
Colin_yuan@hotmail.com

Delai Chen
Shanghai Key Lab of Advanced Manufacturing Environment
China Telecom Shanghai Branch
Shanghai, China
dlchen.2005@hotmail.com

***Abstract*—The fast and constant development of economic and computer network technology urges the building of enterprise integration platform, and quick delivery of new web services. These problems can be solved by applying technologies like Enterprise Service Bus (ESB) and automated web services composition. In this paper, we present an ESB based automated web service composition prototype. Our prototype D-Composer consists of two subsystems: AI planning subsystem and workflow binding subsystem. The former subsystem takes users' requirement as input, uses an AI planning engine to construct an abstract workflow, and transforms it to the workflow binding subsystem through ESB. The latter one binds appropriate service instances to the abstract workflow via a matchmaking engine, then generates a BPEL workflow, and executes it.**

Keywords-Enterprise Service Bus; web service conposition; AI Planning; Matchmaking

I. INTRODUCTION

The fast and constant development of economic and computer network technology speeds up the process of economic globalization. Under such circumstances, enterprises can't survive unless they share information and collaborate with each other, which make the building of integration platform of enterprise applications possible. However, the technologies, software and hardware platforms for enterprises are different, even for the developers' preferences. Thus, it gives rise to the extreme situation that every application system has its nature barriers. To break these barriers, we wish to construct a seamless integration platform of applications, for improving the business efficiency. The traditional way of enterprise application integration is to use component-oriented techniques, like CORBA and COM. However, these techniques complicate the system topology because of the limited access to private components and nonstandard transform protocols. As a service-oriented architecture (SOA) infrastructure, ESB can connect IT resource, whatever its technology is or wherever it is deployed. It can also easily combine and re-assemble services to meet ever-changing requirements without disruption.

On the other hand, the demand of quickly delivering new applications gradually becomes a business imperative today. Much of this application development is currently done in an ad hoc manner without standard frameworks or libraries, thus resulting in poor reuse of software assets. To solve this problem, we apply a web-service-composition manner to automatically meet these needs. Currently, due to their potential in facilitating seamless-to-business or enterprise application integration, web services have received much interest in both industry world and academia. There are basically two different approaches to standardize and compose web services. The industry world has adopted the workflow approach, while the academia has propounded the AI planning approach. Both of these approaches have been developed rather independently from each other, and have some issues respectively. To address this problem, we construct an automated composition system of web services, by combining these two approaches.

The rest of this paper is organized as follows. In the next section, we give the background knowledge about ESB and web service composition. In Section 3 we describe a sample scenario for our research. In Section 4, we describe an ESB based automated web service composition system named D-Composer. Finally, Section 5 concludes our work and presents some future research directions.

II. BACKGROUND

A. Enterprise Service Bus

In the past several years, SOA [18] has been proposed, and its related technologies have been developed enormously. SOA boasts an open standard based on loosely coupled architecture, for better integrating applications in a distributed heterogeneous environment. To enable

*This paper is supported by the Chinese Defense Advance Research Program of Science and Technology (Grant No. 513150302).

978-0-7695-3758-0/09 $25.00 © 2009 IEEE
DOI 10.1109/SSIRI.2009.24

transparent service integration, ESB is introduced as an infrastructure for SOA service connection and message exchange [1]. ESB simplifies the integration and flexible reuse of business components within a SOA. ESB provides a dependable and scalable infrastructure that connects disparate applications and IT resources, mediates their incompatibilities, orchestrates their interactions, and makes them broadly available as services for additional uses [2].

In most organizations, technological heterogeneity is more the rule than the exception. To integrate applications within an ESB, it simplifies connection of new applications, web services, and hundreds of other technologies, including batch files, application servers, legacy middleware products and packaged applications. Any kind of systems could be deployed onto an ESB. For simplicity, in this paper, we assume each software system is wrapped as a web service, which offers substantial benefits in our case.

B. *Web Service Composition*

Web services are considered as self-contained, self-describing, and modular applications that can be published, located, and invoked across the Web [3]. They would offer standardized interface description, discovery and messaging mechanisms. Currently, the programming tools and runtime environments for web services is also matured. Web service composition is to compose autonomous web services to achieve new functionalities. Through web services more quickly to provide more powerful services [4].

Nowadays this issue has received much interest in industry world and academia due to their potentials in facilitating seamless business-to-business or enterprise application integration. On one side, the business world has developed a number of XML-based standards to formalize the specification of web services (such as WSDL [6]). Their interfaces look like remote procedure call and the interaction protocols are manually written, and use the workflow-based composition methods to composite web services. On the other side, the Semantic Web community focuses on reasoning about web resources by explicitly declaring their preconditions and effects with terms precisely defined in ontologies (such as OWL-S [7]). For the composition of Web services, they draw on the goal-oriented inference from AI planning [3] [5].

So far, both approaches have been developed rather independently from each other. By comparing these approaches and analyzing their solutions to the problems of modeling, composing, executing and verifying web services, it turns out that AI planning technique has better performance in automaticity than workflow technique. However, it is rarely used in practice. Although workflow based composition methods have less automaticity, it is widely used in industry world. In order to apply AI planning technique to practical business projects, we combine these two different techniques, and present an automated Web service composition system: D-Composer.

III. A MOTIVATING EXAMPLE

Before showing how D-Composer works, we first describe an example scenario of an automobile accessory purchasing problem.

Here we assume there is an online automobile accessory business platform, which accessory providers can wrap their applications and deploy on. A buyer needs to buy several kinds of accessories to construct a car. He/she wants to take an order request by describing the car and get everything needed, leaving the rest to the platform, as in Fig.1.

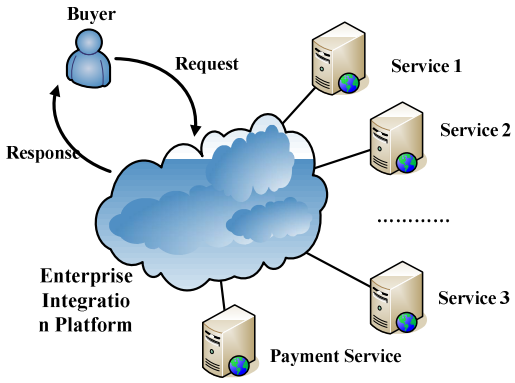

Figure 1. The online business platform

In this scenario, all services have no idea about the location of other services on the platform, and what kind of technology they use to build their application system. The buyer doesn't know this information neither, he/she just have to focus on the purchase request, and wait for the platform to give an appropriate result.

Therefore, we use ESB to build the business platform, and achieve the enterprise integration which deals with the reliable message transferring and routing, and shields the technical information from the user and web services, leaving them focus on their business logics. At the same time, we build an automated Web Service composition system (D-Composer), which is deployed as a single web service on the platform. It takes request from the buyer, generates an composite plan, and return it to the buyer.

IV. D-COMPOSER: AN AUTOMATED WEB SERVICE COMPOSITION SYSTEM

A. *System Overview*

In our approach, D-Composer is a two-step web service composition system, which consists of two parts: AI Planning subsystem and workflow binding subsystem. In AI planning subsystem, we take users' request as input, and use an AI planning engine to construct a composition plan, which is known as an abstract workflow (we will describe in detail in the next section). Then the workflow binding subsystem processes the abstract workflow, binds the compatible service instance to abstract workflow and forms

an executable workflow. Finally it applies a workflow engine to execute, monitor and manage the compound service.

Both these two subsystems are wrapped as web services and deployed independently on ESB (see Fig.2), which are used for reliable message transferring and routing, etc. In our design, they are not only subsystems of D-Composer, but also service components of the entire enterprise integration platform. They can communicate with other service components which are available on the platform, respectively. We believe this will enhance the reusability of service components. We will discuss these two subsystems in the following sections. In this paper, we use Mule [13] as the basic Service Bus.

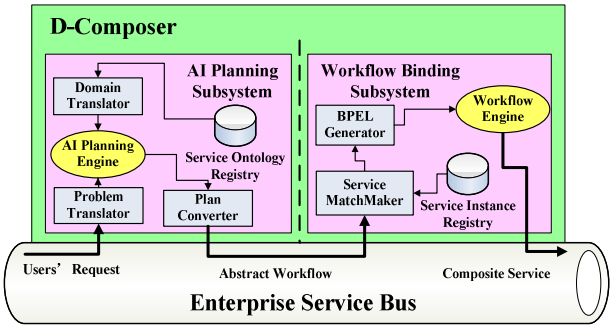

Figure 2. D-Composer system architecture

B. *AI Planning Subsystem*

The AI planning approach focus on the process-centric description of services as actions that are applicable in states. It is responsible for finding a course of actions that can take an agent from the initial state to a goal state, and given a set of actions (legal state transformation functions) in the domain. Formally, a planning problem P is a 3-tuple < I, G, A> where I is the complete description of the initial state, G is the partial description of the goal state, and A is the set of executable (primitive) actions. An action sequence S (a plan) is a solution to P if S can be executed from I and the resulting state of the world contains G [8].

Given a representation of services as actions, we can exploit AI planning techniques for automatic service composition by treating service composition as a planning problem. Ideally, given a user's objective and a set of Web Services, a planner would find out a collection of Web Services requests that satisfies the objective.

By comparison to workflow technique, AI planning offers better automaticity. Therefore in our AI planning subsystem, we complete the planning process in the following three phases.

1) Representation of web services

First of all, in the AI planning subsystem, the composition process typically involves reasoning procedures. Therefore we should find a way to describe the available web services, to enable automatic discovery and composition of desired functionality. At this point, we use OWL-S markup language to create the domain model using services that are described. The OWL-S markup language (previously known as DAML-S) is also being defined for facilitating the creation of web service ontologies. It defines a service that presents a ServiceProfile (i.e. what the service does), is described by a ServiceModel (i.e. how it works) and supports a ServiceGrounding (i.e. how to access it).

Currently, OWL-S is designed to describe a single web service instance. However there is one thing to be conceded. In our example scenario, it is possible, that there exists a large collection of web services which offers the same function on the integration platform. This will make the total number of web services enormous, and affects the performance of the planning procedure. To avoid this problem, we take the representation of web service at two levels – web service types and web service instances [9]. In this phase, services are described in a high-level manner, which is web service type level. It suffices to describe the capabilities of the types of web services, using semantic annotations. In our solution, we use the ServiceProfile model of OWL-S to represent web service type definitions.

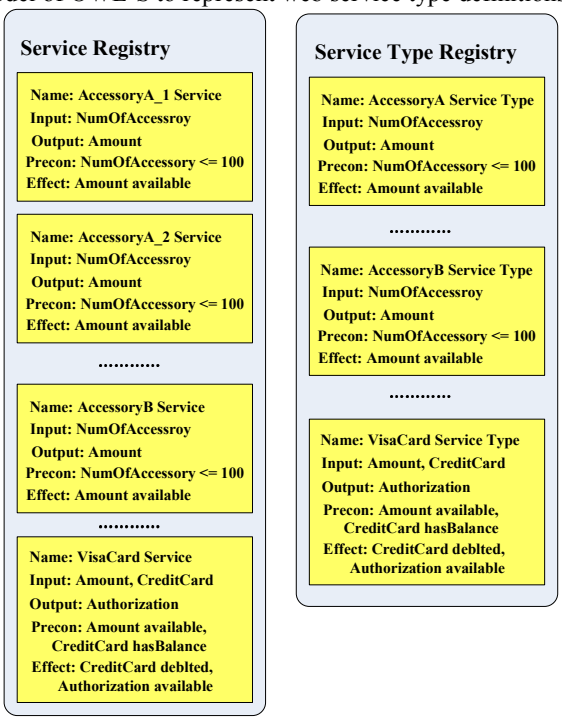

Figure3. Service type registry

Fig.3 depicts the difference between a service registry and a service type registry. In Fig.3, there are more than one service offering accessory A in service registry, while in service type registry, we combine these services together as the only Aceesory A Service Type.

2) Transformation of Planning Models

In this phase, transformation of planning models

comprises two parts: web service model transformation and planning problem transformation. In the OWL-S process ontology, operations are modeled as processes. There are three kinds of processes: atomic process, composite process and simple process. In service model transformation, the goal is to translate these process models into planning models. This process is considered as engine-specific, in our paper, we use SHOP2 [10] AI planning system. It is a domain-independent HTN planning system. HTN planning [11] is an AI planning methodology that creates plans by task decomposition. It recursively decomposes a large task into small tasks, until they can not be decomposed any more. We believe that HTN planning is especially promising for our work, because the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in OWL-S process ontology. The core arithmetic is to transform atomic processes which can be execute directly into SHOP2 operators, and can transform composite and simple processes into SHOP2 methods [12].

Through the planning problem transformation, we should translate the users' requirement into a planning problem. We again use the OWL-S for representing the requirements in IOPE (input, output, precondition, effect) terms, since the result of the composition is also a service. Note that this transformation part is also engine-specific, which means in our paper, the translated planning problem is related to SHOP2 planner.

3) Construction of a Plan

After transforming the planning model, the last phase is to execute the planning engine to construct a composite process, which ultimately consists in the execution of a collection of specific atomic processes. After the plan is generated, we should translate it into OWL-S format, and named as abstract workflow since web service instance information is not known at this stage. Fig.4 depicts the complete process of AI planning subsystem in automobile accessory purchasing problem.

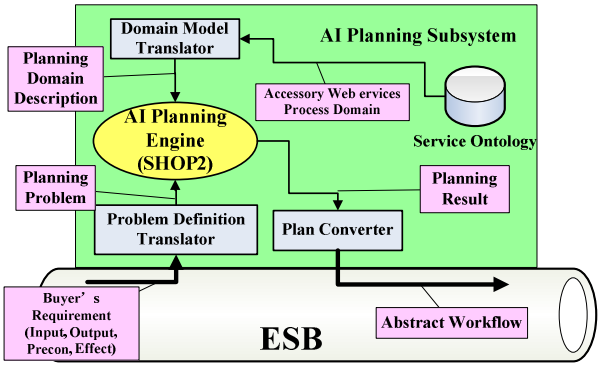

Figure 4. AI planning subsystem

In Figure 4, the first step is to take the buyer's requirement and translate it into a planning problem through a *Problem Definition Translator*. Secondly, the subsystem translates the available accessory web service processes into planning domain models using a *Domain Model Translator*. Next, we apply the *AI Planning Engine* and *Plan Converter* to construct an abstract purchasing plan. Finally, we pass the plan through ESB.

C. Workflow Binding Subsystem

Now the abstract workflow for the composite service is fed to the Workflow Binding subsystem through ESB. At this stage, we need to bind services instance to it, then generate an executable workflow, and finally take it into execution. In Workflow Binding subsystem, there are two main phases, which we will describe below respectively.

1) Matchmaking of Service Instance

In Section 4.2 we only consider the type of web services, however at this stage the process of matchmaking each service type to a corresponding instance should be taken into consideration. The matchmaking problem has been addressed extensively for web services and involves a number of issues related to data flow orchestration, data type, invocation protocol matching, QoS matching and SLA composition. While some of these issues can be resolved in an automated manner, others might require manual intervention from a developer supervising the composition process [8]. At this point, we need matchmaking facilities namely symmetry of information exchange between services and their consumers, the ability of each side to describe requirements from the other side, consumer demands, and a methodology to choose efficiently among competing service instances. For simplicity, in our paper, we simply build a matchmaking engine which merely takes input/output and binding information into consideration.

2) Generating of Executable Workflow

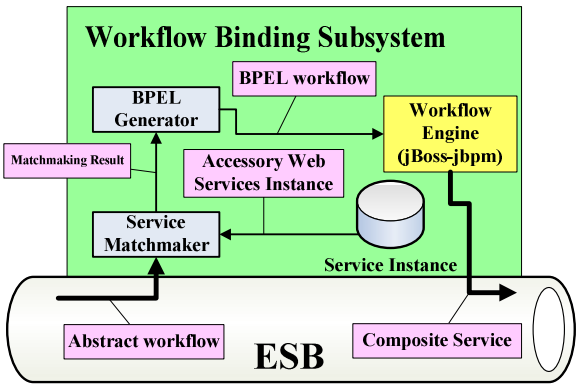

Figure 5. Workflow binding subsystem

Until now, we have applied OWL-S for most of the representation. However, in this phase, we should translate it into a BPEL [16] workflow [14] [15]. In our paper, we build a BPEL generator to produce a BPEL workflow from the matchmaking result. The generated workflow can be deployed onto a workflow engine (we use jBoss-jbpm

workflow engine [17]) to execute it.

In Figure 5, the *Service Matchmaker* takes the abstract workflow (which is known as the abstract purchasing plan) and the accessory web service instance into matchmaking, and output the matchmaking result. We use a *BPEL Generator* to construct a BPEL workflow and finally apply a *Workflow Engine* to execute it.

V. CONCLUSION

ESB is a good alternative for constructing an enterprise integration platform, since it shields the detail information of transport protocols from application systems, also it can integrate all kinds of application systems. Automated web service composition is an important way for integrating information flexibly and quickly. It can quickly deliver composited Web Service, and make web service more reusable.

In this paper, we present an ESB based service integration platform. According to the fast deployment and plug-in properties of ESB, we describe an automated web service composition prototype named D-Composer. It consists of two subsystems: AI planning subsystem and workflow binding subsystem. The former subsystem takes the users' requirements and constrains as the description of the demanded service by using an AI planner to construct an abstract workflow; the latter binds service instances to the abstract workflow, translates it into a BPEL workflow, and uses a workflow engine to execute it, which completes the Web Service composition ultimately. The two subsystems are both deployed on ESB as independent service components, use ESB for message transform.

However, some limitations may exist in our current system. The first one is that we consider the world as closed, while in reality it is quite the opposite, which means web services are changing all the while. This requires us to gather dynamic information while planning and binding. The second problem is that the simple matchmaker we use is insufficient in the future. We would like to refine our system further, and consider these two problems as our future work.

REFERENCES

[1] X Bai, J Xie, B Chen, and S Xiao, "DRESR: dynamic routing in enterprise service bus", IEEE International Conference on e-Business Engineering, 2007

[2] D.Chappell, Enterprise Service Bus. O'Reilly. 2004.

[3] [3] J Rao, and X Su, "A survey of automated web service composition methods", In Proceedings of First International Workshop on Semantic Web Services and Web Process Composition, 2004

[4] H Sun, X Wang, B Zhou, and P Zou, "Research and implementation of dynamic web services composition", Springer-Verlag Berlin Heidelberg, 2003

[5] B.Srivastava, and J.Koehler, "Web service composition – current solutions and open problems", ICAP 2003 Workshop on Planning for Web Services, 2003

[6] R. Chinnici, J.Moreau, A.Ryman, et al, "Web Services Description Language (WSDL) 2.0", Online: http://www.w3.org/TR/wsdl20/, 2007

[7] D.Martin, M.Burstein, J.Hobbs, et al, OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/overview/

[8] V.Agarwal, K.Dasgupta, N.Karnik, A.Kumar, A.Kundu, S.Mittal and B.Srivastava, "A service creation environment based on end to end composition of web service", In Proceedings of WWW 2005, pages 128–137, ACM, 2005

[9] A.Kumar, S.Mittal, and B.Srivastava. "Information modeling for end to end composition of semantic web services", In IBM Research Report RI05001, http://domino.watson.ibm/library/CyberDig.nsf/Home. 2005

[10] D.Nau, T.Au, O.Ilghami, U.Kuter, J. William Murdock, D.Wu, and F.Yaman. "SHOP2: an HTN planning system", Journal of Artificial Intelligence Research, pages 379-404, 2003

[11] E.Sirin, B.Parsia, D.Wu, J.Hendler, and D.Nau, "HTN planning for web service composition using SHOP2", Web Semantics: Science, Services and Agents on the World Wide Web 1, pages 377-396, 2004

[12] O.Ilghami, Documentation for JSHOP2, http://sourceforge.net/projects/shop. 2006

[13] Mule: Open Source ESB, http://mule.mulesource.org/display/MULE/Home. 2008

[14] D.J.Mandell and S.A.Mclraith, "Adapting BPEL4WS for the semantic web: the bottom-Up approach to web service interoperation", Springer-Verlag Berlin Heidelberg, 2003

[15] P.Traverso and M.Pistore. "Automated composition of semantic web services into executable process", Springer-Verlag Berlin Heidelberg, 2003

[16] T.Andrews, F.Curbera, H.Dholakia, et al, "Business Process Execution Language for Web Services." http://www.ibm.com/developerworks/library/specification/ws-bpel/. 2007

[17] JBoss jBPM WS-BPEL Runtime User Guide. Introduction to jBPM BPEL. http://docs.jboss.com/jbpm/bpel/v1.1/userguide/introduction.html. 2008

[18] E.Newcomer and G.Lomow. Understanding SOA with Web Services. Addison Wesley Professional. 2004.