A Modeling Approach for Service-Oriented Architecture

Tao Zhang[1], Shing Ying[1], Sheng Cao[2], Jiankeng Zhang[3], and Jun Wei[4]
1 State Key Laboratory of Software Engineering, Wuhan University, Wuhan, Hubei Province 430072, P. R. China cwhzhangtao@163.com yingshi@whu.edu.cn
2 Computer Center, Hubei Telecom, Wuhan, Hubei Province 430023, P. R. China caosheng@ttsc.hbtelecom.com.cn
3 Trillium Business Unit, Continuous Computing□China)Co. Ltd, Shenzhen, Guangdong Province 518057, P. R. China jacken.zhang@ccpu.com
4 Fakultät Informatik, Technische Univäsität Dresden, Emil-Ueberall-Strasse 31, 01159 Dresden, Germany weijun1980@googlemail.com

Abstract. Specifying service-oriented system, which is a new type of Enterprise Information System, is critical with the rapid development of Web Services. For the purpose, this paper presents an approach to model Service-Oriented Architecture (SOA). We do not exclude the traditional opinions of software architecture description as the current specification approaches; on the contrary, some similar conceptions, such as service components and connectors are defined in this paper. For architectural reuse, the notion of composite service components is specified to model services composition and that of composite connectors is defined to abstract the complex communication protocols. Within the context of the Travel Reservation System (TRS), we demonstrate the usage and practicability of our approach based-on Web Services.

1 Introduction

With the rapid development of information science and technology, different kinds of enterprise information systems are proposed to handle the diverse changes of requirements anywhere and anytime. In order to deal with the situation better, service-oriented systems are hot spots to decrease dependencies between different information artifacts. At the same time, Web Services has made considerable progress recent years, thus Service-Oriented Architecture (SOA) is paid extensive attention. SOA provides a new blueprint to solve software reuse and enterprise

Please use the following format when citing this chapter:

Zhang, T., Ying, S., Cao, S., Zhang, J., Wei, J., 2006, in International Federation for Information Processing, Volume 205, Research and Practical Issues of Enterprise Information Systems, eds. Tjoa, A.M., Xu, L., Chaudhry, S., (Boston:Springer), pp.63-72.

information system integration which publish business functionality in the form of programming and accessible software services, other application programs can use these services by published and discoverable interfaces [1-4]. Owing to its loosely coupled, open, high dynamic and high flexible nature, software systems developed based on SOA can be better suitable for requirement and application environment changes.

For the sake of the service-oriented system design, it is imperative to model SOA at an abstract level, taking no account into concrete technologies involved in the application employed. However it is impossible to describe SOA sufficiently by the current popular UML-based approach [5-7] and BPEL [8, 9] since UML is short of the support to describe some elements involved in software architecture and BPEL focuses on the specification of abstract or executive business processes but not on the description of functional units of system architecture. On the other hand, the existing ADLs (architecture definition languages) [10-14] cannot specify SOA very well in spite that we can benefit from the approaches based-on them.

Therefore, on the basis of traditional software architectural describing techniques, this paper puts forward a modeling method for SOA by SO-ADL. Unlike other specification approaches, we do not exclude the traditional architectural description opinions since they are helpful to specify SOA to some extend, so some similar conceptions are defined as the first-class entities in our approach. Meanwhile, to the end of architectural reuse, composite service components and composite connectors are specified, too. Then, by using Web Services [15, 16], the usage of our method is manifest within the context of the Travel Reservation System (TRS).

The remainder of this paper is organized as follow. Section 2 highlights the requirements for modeling SOA. Then, SO-ADL is illuminated in Section 3, and Section 4 elucidates the specification of Travel Reservation System (TRS). Furthermore, related work is discussed in Section 6 and Section 7 concludes this paper.

2 Requirements

SOA is a particular kind of software architectures that is designed to create a dynamically organized environment of networked services that are composable and interoperable [17]. In order to provide an effective and practical approach to specify SOA, several requirements should be taken into account at least.

Firstly, the description should capture the structure of SOA in an abstract layer and as far as SOA is concerned, services are its building blocks that communicate by transmitting synchronous or asynchronous messages. Secondly, it should give strong consideration of SOA's highly dynamic structure, highly flexible nature and its composable ability in descriptions. Thirdly, the notation has to accord with the vocabulary used to describe SOA in natural language for improving its usability and architectural reuse. Besides, the approach should keep the balance between concision and understandability, simplicity and functionality. And as the changeful nature of SOA itself, it should be open and extensible not only at describing syntax but also at specifying framework.

3 SO-ADL

This section gives the definition of SO-ADL at length and their corresponding notations are specified, too. Unlike existing modeling approaches referred before, traditional software architectural specification techniques are not excluded in our method; on the contrary, we believe that they are helpful in SOA description issues and several similar conceptions are defined. Then, aiming at architectural reuse, our approach allows the definition of composite service components and composite connectors to abstract the complex constructs. Meanwhile, we adopt the traditional types-and-instances model and both of service components and connectors have types which are defined at design time and instances which are specified at run-time.

3.1 Service Components

The concept of *Service Components* is introduced to abstract and specify the computing units related to services since they either provide or request services or both provide and request services simultaneously. As shown in figure 1, a service component type is defined from four aspects as follows,

- *Identification* refers to the name of the service offered and it is a unique identifier, which identifies the service component.
- *Interface* specifies the service a service component provides or requests and it is a set of interaction points between a service component and its external world. The notion of interface separates the services from their implementations, and then service provider is able to change the implementation without any influence on service consumer, which only needs to know interface specifications of provider service component. There are two types of interfaces: *provision interface* and *request interface*.
- *Specification* is a brief description of services provided by service components, including *service specification*, which declares what the service offers and *service contract* that gives information about how to use the service and indicates the roles a service component acts. SO-ADL prescribes specification should be described in a machine-and-human readable format and platform-independently, while it has not any further restrictions on the definition mechanism in detail.
- *Condition* defines *pre-conditions*, *post-conditions* and *invariants* of service components that are useful during the service components' composition process. Similarly, SO-ADL does not limit the describing techniques for condition declaration and various logical methods are applicable.

At first sight, our definition of service components looks similar to the components defined in traditional ADLs given in [11-14], and they have some similar features indeed. However, compared with common components, service components are larger granularity that provide relatively integrated functionality and rely on a bigger data set. Meanwhile, service components are much closer to business requirements than ordinary components just like the advantage of SOA relative to traditional software architecture. On the other hand, service component

notion is not the same as that of service since a service component is a functional unit rather than a business-oriented service in spite of their many similar properties.

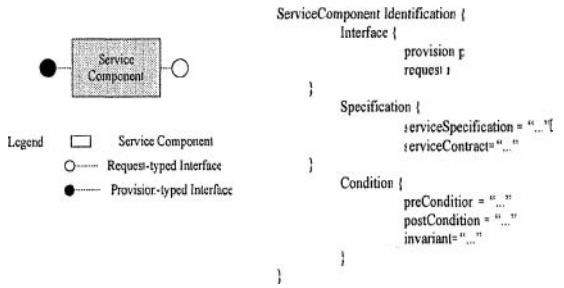

Fig. 1. The Notation of Service Component Type

3.2 Connectors

The notion of connectors is used to model the communication mechanisms between service components explicitly. Note that some ADLs are short of the definitions of connectors, therefore the interaction pattern specified by them is asymmetric. Thus, the interaction patterns cannot be described independent of the computation unit, which violates the design principle of SOA. Accordingly, we define connectors in an explicitly manner and direct connections between different service components are prohibited without connectors while connectors can link each other directly.

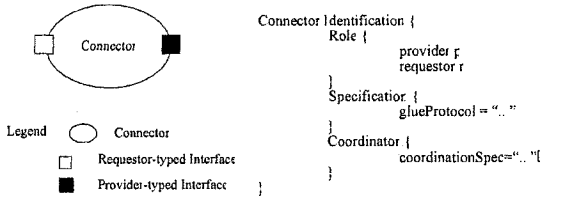

Fig. 2. The Notation of Connector Type

As shown in figure 2, SO-ADL specifies connector type in four aspects as following,

- *Identification* is the name of a connector that is a unique identifier to identify it.
- *Role* specifies the interface of a connector. A role prescribes the responsibility of service component that is attached to the connector and there are two types of roles: *provider* and *requestor*.
- *Specification* declares the *glue protocol* between the different roles of the connectors, which are the transport protocol between different service

components in fact. SO-ADL supports various kinds of transport protocols based-on different implementation techniques.

- *Coordinator* is defined to enable proper connectivity of service components that is responsible for the compatibility of type between connector and service component. The coordinator are defined by declaring the *coordination specification*, including the coordination information and policy which give constraints to restrict the attachments of service components and connectors.

3.3 Configuration

A service-oriented system is configured by service component instances and connector instances and the symbol-"as" in textual notation while "—"link in graphic notation are used to identify configuration relationships. For example, attachment of service component instance s and connector instance c is noted: "s.p as c.r" (p is a provision interface of s while r is a requestor role of c). That is, s provides its services to other service component instances through c. Thereby declaration of service-oriented system configuration is composed by the declaration of service component instances, connector instances as well as their attachment relationships.

As far as SOA, its configurations are open, highly dynamic and highly flexible which allow unbounded creation, deletion and modification of constructs. Therefore, we introduce a role-driven way of configuration, that is, configurations are triggered on-demand directly. Note that, roles and services are relatively determinate by business requirements in spite of the changeable structures. Further discussion is beyond of the scope of this paper. Then, based-on services and roles elicited from requirements analyzing, we can draw various configurations as blueprints of service-oriented systems.

3.4 Composite Constructs

As SOA is defined in terms of its ability to compose and recompose services, it is imperative to define the composition of service components. Therefore, the notion of *composite service components* is defined to support composed service component specifications and to achieve architectural reuse simultaneously.

Note that service components are not allowed to link one another directly, thus, composite service components have inter-structures that are configured from other service components' instances and corresponding connectors' instances. To gain service composition diagrams automatically, a composition process is designed, which can be implemented in programming language. According to the composition process, we define composite service components by declaration of their constituents' instances and their configuration. For conciseness, we do not introduce any new notation for composite service component specification while describe it based-on basic constructs defined before. In this way, composite service components are specified from four aspects in the same form as primitive service components while there are some differences in contents. As a matter of fact, SO-ADL does not

give any restrictions of composing patterns and it is useful to specify different composite service component composed in different ways.

For example, figure 3 shows the ordinal composition pattern and "dummy provision" and "dummy request" are specified to define the composite service components' interfaces that are attached to the first service component's interfaces in the composing sequence and the last one. What's more, composite service component specification declares its inter-structure (or inter-configuration). As far as condition, its pre-condition and post-condition will be the pre-condition of the first service component and the last one's post-condition respectively, while its invariant will be the logical "and" of the constitutes' invariants.

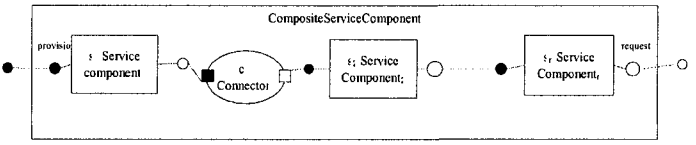

Fig. 3. Ordinal Composite Service Component Type-Graphic Notation

On the other hand, considering the complex communication mechanisms between service components, SO-ADL supports the definition of composite connectors for architectural reuse. As connectors are allowed to connect each other directly, composite connectors have interior structures that are composed by different connectors. SO-ADL does not limit the composition patterns, either. And different composite connectors can be depicted by connector specifications and their inter-structure declarations. Furthermore, the approaches presented in [18, 19] are helpful to provide some operators with which new connectors can be built up from old connectors using well-founded operators for composition.

3.5 Dynamic Structures

The dynamic changes of SOAs encounter the key issues in three aspects: dynamic modification, dynamic configuration and reconfiguration, and dynamic composition and recomposition. Dynamic modification is an intrusive change which may modify the structures or behaviors of the service components, whereas, the other ways of dynamic changes are non-intrusive changes.

Our approach supports all of the above three kinds of dynamic changes in SOA. It is obvious that the dynamic modification has no influence on the definition of SOA since SO-ADL is specified as platform-neutral and implementation-independent. At the same time, in view of the loosely-coupled nature defined in SO-ADL, the dynamic modifications almost have no effects on users.

As configurations are triggered on-demand directly in virtue of a role-driven way in our method, dynamic configuration and reconfiguration are also supported. We can identify services, roles as well as their relationship, which are elicited during the requirement analyzing by use case techniques (which is out of the scope of this

paper), then, which can be mapping into the descriptions of service components and connectors. In so far as implementation, specification of constructs and their configuration relationships will be mapped to lower level. Similarly, dynamic composition and recomposition can be described in SO-ADL as an automated role-driven process, too.

4 Case Study

An application scenario of Travel Reservation System is given to manifest the effect of our approach based-on Web Services. We simplify the concrete scenario in order to illustrate the practicability of the method this paper presented and in the scenario; the requirements of the travelers trigger the dynamic configurations of service-oriented systems and composition of different service components.

To satisfy the needs of travelers, a simple reservation system may include different services, and then, we define service components in the TRS, including airline reservation service component, hotel reservation service component as well as the corresponding connectors are defined, too. Suppose a simple TRS depicted as figure 4 that is constituted from a primitive hotel reservation service component, a composite AirlineReservation service component, several Traveler service components and connectors between them. The instances of the above service components and those of the connectors compose the simple TRS.

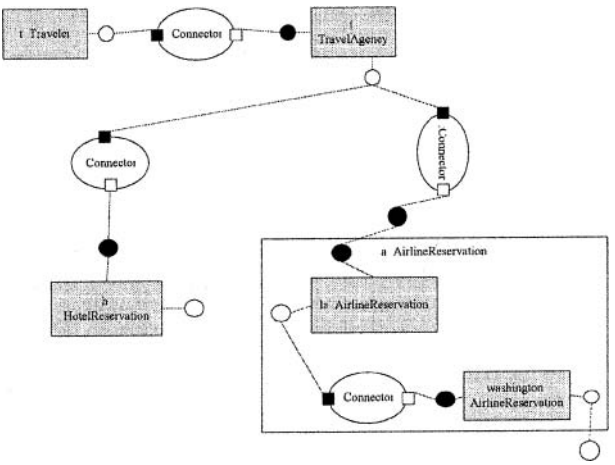

Fig. 4. The Specification of a Simple Travel Reservation System

As referred before, Web Services are used to implement the application system; therefore, WSDL [20], SOAP [21] and other techniques may be involved. Except

declaration of interfaces, WSDL is adopted to specify service components' specifications that only need to indicate URIs of the WSDL documents. Furthermore, a composition process is designed to gain a service composition diagram automatically, which can be implemented in programming language. For instance, the AirlineReservation service component, which is able to provide the airline reservations from one place to Los Angeles, then to Washington, is composed from two service components that provide different airline reservation services. As far as the condition declaration of service component, description logic is used to express the pre-conditions, post-conditions and invariants, containing *and*, *or* and *not* logic symbol. Simultaneously, we can decompose a complex condition to atomic conditions, which can be matched by single service component respectively. So, several service components can be orderly composed to satisfy the request as the same as the definition of the AirlineReservation service component.

When it comes to definition of connectors, SOAP is helpful to be the glue protocol that specifies the communication pattern between different roles of connectors and it abstracts the interaction mechanisms between various service components in fact. SOAP is used as the main communication protocol of TRS that allows different composite connectors to express complex protocols yet. And the coordinator will prescribe the configuration relationships between service components and connectors, and between different connectors that may be specified by other techniques provided by Web Services (not included in this paper version).

Then, based-on the identification of roles and their relationships elicited in upper phases, we can configure a simple TRS as figure 4. Meanwhile, with the dynamic changes of roles' specification, the TRS will be dynamic reconfigured and the service components involved will be dynamic recomposed too.

5 Related Work

The work presented in this paper has been influenced by several different proposals. First of all, we are inspired by the research work on software architectures, especially the specification of different ADLs, such as Darwin [11], Wright [12, 22], xADL [13], and ABC [14]. The elements defined within them are the foundation of our describing, and the framework of our approach is benefit from them, too. Although we cannot use the tools they provided, our approach can benefit from the methodologies used by them.

On the other hand, the modeling of SOA based-on UML is a hot-spot nowadays. In most of these UML-based approaches, we noticed the methods presented by Baresi *et al.* [5, 6] and AMir *et al.* [7]. Baresi *et al.* describe SOA by the static model and dynamic model in UML, then, encode the model into a transition system. AMir *et al.* use the UML profile to specify SOA, and their SOA UML profiles consist of five profiles: the resource profile, the service profile, the message profile, the service policy profile, and the agent profile. Then the notation of UML and the stereotype is used to describe them. However, both of them fail to model SOA appropriately since UML itself lack the support to describe the peculiar elements involved in software architecture that is overcome by our approach. In addition, we can take advantage of

the use case technologies defined in UML to elicit the roles and services, which are the key elements of our approach, from business requirement.

In the context, we mush mention the work done by Krüger *et al.* [10] who propose an ADL for describing service, which views roles and service components as fundamental elements in SOA. Furthermore, they substantiate their view of services as cross-cutting architectural aspects by providing a mapping from services to aspects in AspectJ. But, the ADL does not emphasize the whole structure description of SOA and the corresponding graph supports. Besides, Stojanovic *et al.* [23] present a CBD-based method to model SOA. However, it limits to the traditional component modeling techniques and does not concern the properties of SOA eligibly. The approach in this paper gives relatively sufficient consideration of the above problems.

6 Conclusions

This paper addresses the problem of modeling SOA that gives the blueprints to a kind of enterprise information system. For the purpose, SO-ADL is put forward to specify SOA and its effect is shown within the application scenario of travel reservation system. In future, we will devote ourselves to the research of SO-ADL-based service-oriented development process and its corresponding tools. At the same time, how to bridge the gap between different service-oriented development phases better is our concern, too.

Acknowledgements

This research work is supported by the National Nature Science Foundation of P. R. China under No. 60473066 and Young Outstanding Talent Foundation of Hubei Province, P. R. China under No. 2003ABB004.

I appreciate the help from my colleagues; they are Huiming Xiong, Xiangyang Jia, Zaoqing Liang, Peng Ye, Honghua Cao, Jie Zhang, Hua Qin, and Yi Zhang.

References

1. M. Endrei, J. Ang, and A. Arsanjani, Patterns: Service-Oriented Architecture and Web Services (April 2004); *http://www.ibm.com/redbooks/.*

2. M. P. Papazoglou, D. Georgakooulos, Service-Oriented Computing: Introduction, *Communications of the ACM* **46**(10), 24-28 (2003).

3. T. Erl, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services* (Prentice Hall, New Jersey, USA, 2004).

4. M. P. Papazoglou, Service-Oriented Computing: Concepts, Characteristic and Directions, in: Proceedings of the 4th International Conference on Web Information System Engineering, Roma, Italy (2003), pp. 3-10.

5. L. Baresi, R. Heckel, S. Thöne, and D. Varró, Modeling and Validation of Service-Oriented Architecture: Application vs. Style, in: the Proceedings of ESEC/FSE 2003, Helsinki, Finland, September (2003), pp. 68-77.
6. L. Baresi, R. Heckel, and S. Thöne,, Modeling and Analysis of Architectural Styles Based on Graph Transformation, in: the Proceedings of 6th ICSE Workshop on Component-Based Software Engineering (CBSE6): Automated Reasoning and Prediction (2003), pp. 67-72.
7. R. Amir and A. Zeid, A UML Profile for Service Oriented Architectures, in: the Proceedings of OOPSLA '04, Vancouver, British Columbia, Canada, Oct. 24-28 (2004), pp.192-193.
8. T. Andrews, F. Curbera, and H. Dholakia, Business Process Execution Language for Web Services, Version 1.1. BPEL4WS Specification, 2003; http://www.ibm.com/developerworks/library/ws-bpel/.
9. OASIS. Web Services Business Process Execution Language Working Draft (February 27, 2005); http://www.oasis-open.org/apps/org/workgroup/wsbpel/.
10. I.H. Krüger and R. Mathew, Systematic Development and Exploration of Service-Oriented Software Architectures, in: the Proc. of the fourth Working IEEE/IFIP Conference on Software Architecture (WISCA'04).
11. J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, Specifying Distributed Software Architectures, in: the Proceedings of the Fifth European Software Engineering Conference (Springer-Verlag, New York, 1995), pp.137—154.
12. R. Allen, A Formal Approach to Software Architecture, PhD thesis, School of Computer Science, Carnegie Mellon University, 1997.
13. E. M. Dashofy, A. van der Hoek, and R. N. Taylor, An Infrastructure for the Rapid Development of XML-based Architecture Description Languages, in: the Proceedings of the 24th International Conference on Software Engineering (ACM Press, New York, 2002), pp.266-276.
14. H. Mei, F. Chen, and Q.X. Wang, ABC/ADL: an ADL Supporting Component Composition, in: the Proceedings of the 4th International Conference on Formal Engineering Methods (Springer-Verlag, New York, 2002), pp.38-47.
15. F. Curbera, R. Khalaf, and N. Mukhi, The Next Step in Web Services, *Communications of the ACM* **46**(10), 29-34 (2003).
16. M. Champion, C. Rerris, and E. Newcomer, Web Service Architecture, W3C Working Draft (November 2002); *http://www.w3.org/TR/2002/WD-ws-arch-20021114/.*
17. J. Bloomberg, The Role of the Service-Oriented (May 2003) Architect; *http://www.therationaledge.com/may_03/f_bloomberg/.*
18. B. Spitznagel and D. Garlan, A Compositional Approach for Constructing Connectors, in: the Proceedings of the 2nd Working IEEEE/IFIP Conference on Software Architecture (IEEE Computer Society, 2001), pp. 148-157.
19. A. Lopes, M. Wermelinger, and J. L. Fiadeiro, Higher-order Connectors, *ACM Transactions on Software Engineering and Methodology* **12**(1), 64-104 (2003).
20. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, W3C. Web Services Description Language (WSDL) Version 2.0; http://www.w3.org/TR/2005/WD-wsdl20-primer-20050510.
21. M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, and H. Nielsen, W3C SOAP Version 1.2 (June 24, 2003);http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.
22. R.J. Allen and D. Garlan, A Formal Basis for Architectural Connection, *ACM Transactions on Software Engineering and Methodology* **6**(3), 213-249 (1997).
23. Z. Stojanovic, A. Dahanayake, and H. Sol, Agile Modeling and Design of Service-Oriented Component Architecture, in: 1st European Workshop on Object-Orientation and Web Services, in conjunction with ECOOP 2003, July 21-25, 2003, Darmstadt, Germany, IBM Technical Report.