Journal of Southeast University (English Edition) Vol. 22, No. 3, pp. 311 – 314 Sept. 2006 ISSN 1003—7985

Supporting web services reuse by semantic service component and composition pattern

Chu Wang Qian Depei

(Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: Due to the fact that the existing web service description methods cannot address the issue of service reuse of various levels of granularity, the concept of service component is introduced, which packages together web services and choreography, and their operations and properties are presented in a consistent and uniform manner. Service components are published externally as normal web services and can thus be employed by web-based applications. In order to improve reusability and testability of service components, the concept of composition pattern is also proposed, which presents the relationships among service components. The relationships and relationship compositions have a rigorous semantic, so that composite components can be validated at the configuration stage. The composition patterns support to integrate service components of various levels of granularity. Experience indicates that the application assembly can effectively be conducted by understanding, selecting, and reusing components easily.

Key words: semantic web; semantic service component; service reuse; composition pattern

One of the challenges to reusing services in an effective way is the specification of web services. Current standard technologies for web services (e. g. WSDL) provide only syntactic-level descriptions of their functionalities, without any formal definition of what the syntactic definitions mean. The lack of machine-readable semantics necessitates human intervention for service discovery and composition. Semantic web services relax this restriction by augmenting web services with rich formal descriptions of their capabilities, thus facilitating automated composition, discovery, dynamic binding, and invocation of services[1].

Another challenge to reusing web services is the granularity. The service reuse should support the integration of services at various levels of granularity. The services should be understood at a coarse granularity. That is, instead of modeling actions and interactions at a detailed level, it would be better to capture the high-level services. Coarse granularity reduces dependencies among the participants and reduces communications to a few messages of greater significance[2–3].

This paper introduces the concept of service component and composition pattern to support services reuse at various levels of granularity. Service component encapsulates operation, behavior, constituent services, and choreography. Because the choreography among the services is a critical part of the application and may vary over time, it is important that the choreography is explicitly represented and available to both the provider and the consumer of the published services. Pattern is a way of reusing abstract knowledge about a recurrent problem in a particular context and its solution. When the problem is complex, it can be divided into a few simpler problems, each one being resolved in an independent pattern. The simplicity of a pattern and its "small size" make it easy to understand, to integrate, and to reuse. A domain specific pattern is a natural way for the formulation of accumulated development expertise in system development[4–5]. In this paper, the services composition pattern is represented as service components and relationships among them. The composition pattern supports the carrying out of "what-if" analyses in an efficient manner.

1 Service Reuse and Problems

Karhunen presented the service-oriented software engineering (SOSE) component framework in Ref. [6], which offers service and business components as design models for the system development. The SOSE component model has three levels of granularities: system, business service, and component level. The system level component is a set of services, which all publish services for the customers. The business service component has a four-layer architecture where components depend on another component forming components de-

Received 2006-04-12.

Foundation item: The National Basic Research Program of China (973 Program) (No. 1999032710).

Biographies: Chu Wang (1966—), male, graduate; Qian Depei (corresponding author), male, professor, depeiq@263. net.

pendency. The simplicity of the model enhances its reusability.

Yang et al. [7] presented an approach to the development of composite web services based on the model driven architecture (MDA). The UML class diagram and the activity diagram are used to model structure and behavior of the platform independent model (PIM), respectively, and then the PIMs are converted to specific web services specification[7].

From a software engineering perspective, the real challenge is how to support the service composition, consistency checking, reuse, and maintainability. The service provider imposes a particular decomposition of the business process, which makes it inappropriate for the discovery tool to perform automated service composition. Recomposing a process without knowing the intended side effects of the original composition runs the risk of composing services with unintended side effects[8].

Services composition requires proper abstractions that capture the semantic of the composition pattern and the constraints on how the services may participate in compositions and help reject unsuitable compositions so that only acceptable systems are built. A key aspect in the design of a composed service is to maintain global coherence. This calls for a means with which to pool knowledge and evidence, determine shared goals, determine common tasks across services, and avoid unnecessary conflicts[2]. Most service composition approaches attempt to address service composition by composing single web services from scratch, ignoring the reuse of existing compositions. From a developer's perspective, it is interesting to explore the reuse of composite services. The higher level of service reusability will lead to a more efficient and more structured composition process which will accelerate application development [9].

2 Semantic Service Component and Composition Pattern

The service component is a packaging mechanism combining published web services, which includes seven parts: category, global-goals, operations, constituent services, behavior, choreography, and constraints.

Service component: : =
{
[Category: ID, name, domain, {synonyms}, {Abbreviations}]
[Global-goal: Service component] //Global-goal is represented as a service component structure
[Operations: {(Syntactic properties, semantic properties, operational properties, registered service)}]
[Behavior: Finite state transition system]
[Constituent services: {Service components}]
[Choreography: {Operations and mode (parallel | sequential …)}]
[Constraints: // Knowledge to enforce integrity { constraints }]
}.

The category contains five attributes: ID, name, domain, synonyms, and abbreviations. ID is unique service component identifier that takes the form of a universally unique ID (UUID). Domain denotes the area of interest of the component. The synonyms attribute contains a set of alternative characteristics of name. The abbreviations attribute is a set of short forms of name. Service components are invoked through their operations. The global-goals give the reasons for the existence of the service component.

The operations are described in four parts: syntactic, semantic, operational, and registered service.

Syntactic properties Operations are syntactically described by the following attributes: Op-ID, name, mode, input, and output. Op-ID is unique operation identifier that takes the form of a universally unique ID (UUID). The second attribute gives the name assigned to the current operation. The operation's mode has one of the following values in, out, in/out, and out/in.

Semantic properties The semantics of operations is crucial to enabling services on the semantic web. Semantic properties defined for operations include pre-condition and post-condition.

Operational properties We propose to provide a service component with a test-suite as operational properties that can be used to ensure that the service behavior is preserved through evolution. Every test case of a test-suite is described as t = {initial state, input, output, final state}. If it is shown or required that a service component satisfies business goals (Φ) by testing based on test-suite (S), we write the case as $S \models \Phi$, otherwise $S \not\models \Phi$. These test suites are published as service component facets, so that users can download and periodically run them against the service.

Registered service Registered service is an implemented web service. Let us distinguish between abstract operation and concrete operation. An abstract operation only specifies the business goals without referring to any specific service implementation. A concrete operation specifies the goals and web locations of the service. An abstract operation allows users to share web component without reference to any specific service implementation. For a concrete web component, the invocation of operation is translated into the invocation of registered service. Behavior and choreography repre-

sent the semantic at the service component level. The choreography determines the interoperability of the constituent services. Behavior describes another kind of semantic information of operations by using formalisms such as finite state transition system. Behavior is used to determine valid invocation orders of operations.

When services are developed for reuse, services provider decomposes business goals into sub-goals, and ends up with goals that are satisfied by services to be published or ones that can be constructed easily. This "divide-and-conquer" procedure results in three kinds of relationships among service components: "Goals-Assignment" relationships (named γ-relationships), "Play-Role-In" relationships (named β-relationships), and "Satisfied-By" relationships (named λ-relationship)[10]. Service components are intended to be reusable, if there exist errors or defects in the components, they will be "reused" to applications. Hence, service components and composition should be testable. In order to enhance the testability of service components, the mentioned relationships and relationship compositions should have rigorous semantics.

Definition 1 (γ-relationship) Let C_1 and C_2 be service components, Φ_1 and Φ_2 be goals of C_1 and C_2, respectively; S_1 and S_2 be test suite of C_1 and C_2, respectively. $\gamma: C_1 \rightarrow C_2$ means that Φ_1 is the global-goal of Φ_2, and $S_1 \models \Phi_1 \Rightarrow S_2 \models \Phi_2$, $S_2 \not\models \Phi_2 \Rightarrow S_1 \not\models \Phi_1$.

Definition 2 (β-relationship) Let C_1 and C_2 be service components, Φ_1 and Φ_2 be goals of C_1 and C_2, respectively; S_1 and S_2 be test suites of C_1 and C_2, respectively. $\beta: C_1 \rightarrow C_2$ means that the constituent services of C_2 contains C_1, Φ_2 is the global-goal of Φ_1, and $S_2 \models \Phi_2 \Rightarrow S_1 \models \Phi_1$, $S_1 \not\models \Phi_1 \Rightarrow S_2 \not\models \Phi_2$.

Definition 3 (λ-relationship) Let C_1 and C_2 be service components, Φ_1 and Φ_2 be goals of C_1 and C_2, respectively; S_1 and S_2 be test suites of C_1 and C_2, respectively. $\lambda: C_1 \rightarrow C_2$ means that Φ_1 is the same as Φ_2, and $S_2 \models \Phi_2 \Rightarrow S_1 \models \Phi_1$.

Definition 4 (Service composition pattern) Let C_0, $\{C_i \mid i \in \mathbf{N}\}$, and C be service components, C_0 be an abstract component, and there exist $\{\gamma_i: C_0 \rightarrow C_i \mid i \in \mathbf{N}\}$ and $\{\beta_i: C_i \rightarrow C \mid i \in \mathbf{N}\}$. If there exists $\lambda: C_0 \rightarrow C$, then C_0, C, $\{C_i\}$, $\{\gamma_i\}$, $\{\beta_i\}$, and λ constitute a service composition pattern, written as $C_0 \rightarrow_{\lambda} C\{C_i \mid i \in \mathbf{N}\}$.

By composition pattern, business goals can be mapped into the sub-goals, constituent service components, and choreography. Particularly, service composition patterns describe valid services composition by which developers can retrieve web services and assembly applications efficiently. The service composition pattern also offers an adequate means to deal with the granularity variation problem.

Proposition 1 Let C_1, C_2, and C_3 be service components. ① If there exist $\gamma_1: C_1 \rightarrow C_2$, $\gamma_2: C_2 \rightarrow C_3$, then $\gamma = \gamma_2 \circ \gamma_1: C_1 \rightarrow C_3$ is well-defined. ② If there exist $\lambda_1: C_1 \rightarrow C_2$, $\lambda_2: C_2 \rightarrow C_3$; then $\lambda = \lambda_2 \circ \lambda_1: C_1 \rightarrow C_3$ is well-defined. ③ If there exist $\gamma: C_1 \rightarrow C_2$, $\beta: C_2 \rightarrow C_3$, and for every functional goal of C_1, there exists a corresponding choreography in C_2, then $\lambda = \beta \circ \gamma: C_1 \rightarrow C_3$ is well-defined; otherwise, $\gamma = \beta \circ \gamma: C_1 \rightarrow C_3$ is well-defined.

According to definition 1, definition 2, and definition 3, proposition 1 obviously holds. Proposition 1 means that if the relationships between service components are well-defined when the design decisions are made and the relationships composition has a rigorous semantic, they can be used to trace and to understand service components. The service composition pattern supports coarse granularity composition, which is complementary to the existing service composition approach.

3 Conclusion

Organizing services as components is especially attractive within service reuse contexts. First, components are the sole ingredients of the composition process, which simplifies the composition model as a useful and practical process. Secondly, it can improve service reuse. Components incorporate no implementation details. They are only metadata describing all service aspects. Thirdly, the concept of composition pattern that presents the relationships among components supports the reuse of service composition and testability of service components. The relationships have a rigorous semantic so that composite components can be validated at the configuration stage.

References

[1] Cabral L, Domingue J, Motta E, et al. Approaches to semantic web services: an overview and comparisons [EB/OL]. (2004-08) [2006-03-21]. http://kmi. open. ac. uk/projects/irs/cabralESWS04. pdf.

[2] Huhns M N. Software development with objects, agents, and services [EB/OL]. (2004-10) [2006-03-21]. http://www. open. org. au/ Conferences/oopsla2004 /PapersAO/Keynote-Huhns. pdf.

[3] Majithia S, Walker D W, Gray W A. Automated composition of semantic grid services [EB/OL]. (2004-05) [2006-03-21]. http:// www. wesc. ac. uk/resources/publications/pdf/AHM04/148. pdf.

[4] Rothenberger M A, Dooley K J, Kulkarni U R, et al. Strategies for software reuse: a principal component analysis of reuse practices [J]. *IEEE Transactions on Software Engineering*, 2003, **29**(9): 825 - 837.

[5] Rajasree M S, Reddy P J K, Janakiram D. Pattern oriented software development: moving seamlessly from requirements to architecture [EB/OL]. (2003-05) [2006-03-21]. http://lotus. iitm. ac. in/LabPapers/STRAW03. pdf.

[6] Karhunen H. Dynamic method for service-oriented software design [EB/OL]. (2005-12) [2006-03-21]. http://www. hia. no/iris28/Docs/IRIS2028-1034. pdf.

[7] Yang Yanping, Tan Qingping, Yu Jinshan, et al. A new approach to development of composite web services [J]. *Wuhan University Journal of Natural Sciences*, 2006, **11**(1): 211 - 216.

[8] Mandell D J, McIlraith S A. A bottom-up approach to automating web service discovery, customization, and semantic translation [EB/OL]. (2003-11) [2006-03-21]. http://www. daml. org/services/pubs/www2003sam-djm-workshop. pdf.

[9] Granell C, Gould M, Grønmo R, et al. Improving reuse of web service compositions [EB/OL]. (2005-05) [2006-03-21]. http:// www. geoinfo. uji. es/pubs/ecweb05. pdf.

[10] Chu Wang, Qian Depei, Liu Chuda. Architecture-centric software process for software reuse [A]. In: *Proceedings of the 8th International Conference for Young Computer Scientists*[C]. Beijing: International Academic Publishers/ Beijing World Publishing Corporation, 2005. 217 - 222.

利用语义服务构件和组合模式支持 web 服务重用

楚 旺 钱德沛

(西安交通大学计算机科学与技术系,西安 710049)

摘要:由于目前的 web 服务描述方法不能有效地解决不同粒度的服务重用问题,引入“语义服务构件”的概念封装多个服务以及它们的协调规则(choreography),为不同抽象层次的服务提供统一的描述框架. 为了提高服务的可重用性,提出了“组合模式”的思想描述构件之间的关系,并定义了构件关系的语义. 组合模式可以有效地支持不同粒度的服务重用和设计阶段的测试. 开发经验表明:由于比较容易地进行构件的理解、选择和重用,组合模式能够有效地支持应用组装.

关键词:语义 web;语义服务构件;服务重用;组合模式

中图分类号:TP393