Planning and Specifying the Composition of Web Services

Xiang Gao
INFOLAB
Tilburg University
The Netherlands
gao@uvt.nl

Abstract

As the Web services paradigm becomes popular and more and more applications are created and deployed as Web services, the need for developing new solutions tackling the composition of Web services becomes manifest. However, emerging web service standards and existing methods are not sufficient for realizing the goal of flexible and dynamic composition of Web services, although some preliminary work has been conducted in the area of services composition. This situation has raised the interesting research points for creating and developing new approaches for the Web services composition. In this paper we separate the design and implementation phases of the composite services, and concentrate on the research of the way a composite service is constructed in terms of its constituent services. We call this way the composition logic of a composite service. We firstly propose well-defined notions "composition structures" to represent the joints among the constituent services and generate the reliable structure of a composite service, and then we provide a specification mechanism to clearly specify the internal dependencies of a composite service in terms of the notification and dataflow dependencies. Our solutions for planning and specifying the composition of Web services can effectively support the properties of modularity, interoperability, dynamic reconfigure-ability and fault-tolerance for a composite software system in the dynamic Web environments of business applications, which are a organic part of our framework for tackling the challenges of Web service discovery and composition on which we are working.

1. Introduction

Web services are becoming the prominent paradigm for distributed computing and electronic business, because they represent a novel approach and framework for creating and deploying application-to-application communication on the Web. By Web services, we refer to self-contained, Internet-enabled applications capable not only of performing business activities on their own, but also possessing the ability to engage other Web services in order to complete high-order business transactions. Examples of such Web services include bill payment, customized on-line newspapers, or stock trading services and so on. The platform neutral nature of Web services creates the opportunity for developing composite services by using existing atomic or composite services possibly offered by different organizations. For example, a travel-plan-service-package can be developed by combining several atomic services such as booking-air-ticket-service, booking-hotel-service, making-reservation-for-restaurant and renting-car-service, etc., based on their WSDL descriptions [4].

As the Web services paradigm becomes popular and more and more applications are created and deployed as Web services, the need for developing new solutions tackling the composition of Web services becomes manifest. However, emerging web service standards (e.g., WSDL, UDDI, WSFL and BPFL4WS) and existing methods are not sufficient for realizing the goal of flexible and dynamic composition of Web services. Although some preliminary work has been conducted in the area of services composition, mostly in aspects of workflow-like service integration, service conversation, and B2B protocol definition [1, 2, 3]. However, these approaches are either not flexible or too limited, there is still a lot of research that needs to be done in this direction. This situation has raised the interesting research points for creating and developing new approaches for the Web services composition.

The real challenge in services composition lies in how to provide a complete solution that supports the entire life cycle of services composition, i.e., planning, definition and implementation. By planning, we mean that according to user's requests, a composition plan that how to complete the user's tasks which maps his requests needs to be proposed firstly, afterwards, then the candidate atomic or composite services that possibly complete these tasks need to be discovered, correspondingly. During this phase, every task from the user side needs to be mapped to each service. The outcome of this phase is the synthesis of a composite service out of desirable, or potentially available, atomic services, and the structure of a composite service is generated and formed. At the definition phase, the internal dependencies of the composite service need to be clearly defined and specified. The outcome of this phase is the interdependencies specifications of services composition. Finally, the implementation phase implements the composite service bindings based on the services composition specifications. The first two phases serve as the blueprint and reference of the third phase within the entire life cycle of services composition. Therefore, the first two phases are crucial for the implementation of Web services composition. Without the appropriate planning and definition of Web services composition, it is impossible to implement Web services composition and

satisfy the user's requests. Thus the first two phases within the entire life cycle of services composition are very important, and they correspondingly raise two interesting research points, one is how to systematically plan and model the structure of a composite service? Another is how to clearly specify the inter-relationships of a composite service? If these two issues can be properly solved, the development and implementation of Web services composition would be greatly facilitated. However, the existing standards and approaches haven't systematically addressed these issues, or are not enough to effectively tackle these issues.

In this paper we concentrate on first two interesting issues and propose the well-defined notions "***Composition Structures***" to plan and model the structure of a composite service during the planning stage of services composition, and then we come up with a specification mechanism that effectively tackles the definition stage of the entire life cycle of services composition, as it can clearly specify the internal dependencies of the composite service which effectively facilitated the implementation of services composition. In the planning stage of services composition, our objective is to easily generate the composition process and to support the composer in selecting most suitable services. Thus we modified and enhanced the task structures notions [7] and developed our new composing notions for the services composition: composition structures. These composition notions capture the most critical aspect of services composition: ***joints*** among the several services; and address the use of libraries of well-defined building blocks to represent the joints and connect the services, which provides a kind of reliable and lightweight mechanism for services composition. By using these notions and notations, the basic structure of services composition can be generated, afterwards, and the complete and nested structure of services composition also can be generated based on the basic structure of services composition if it is necessary. Naturally, our assumption is that the composer will prefer dealing with appropriate and fewer services where possible, which increases the security and the trustworthiness of the resulting services. During the definition stage of services composition, we would like to provide a specification approach which specifies the internal dependencies of a composite service. We view every service as an independent unit of computation. Our work is motivated by the observation that a composite Web service is constructed by composing several constituent Web services, which are executed in a heterogeneous environment. The resulting composite Web service might be very complex in structure and relations, containing many ***notification*** and ***dataflow*** dependencies among their constituent services. Furthermore, the execution of such a composite service may take a long time to complete, and may contain long periods of inactivity, often due to the constituent services requiring user interactions. In a distributed environment, it is inevitable that long running composite service will require support for fault-tolerance and dynamic reconfiguration: machines may fail, services may be moved or withdrawn and application requirements may change. In such an environment it is essential that the clear specification of the structure and internal dependencies of the composite service need to be clearly specified, and this specification need support the properties of modularity, interoperability, dynamic reconfigure-ability and fault-tolerance for a composite software system in the dynamic Web environments of business applications.

Thus the focus of this paper is on the following:

- We propose well-defined notions and notations "composition structures" to capture the most key aspect of services composition: joints among the several constituent services; and represent these joints and connect the constituent services. By using "composition structures", the structure of a composite service can be reliably generated and formed.
- Afterwards, we come up with a specification mechanism that can clearly specify the internal dependencies of a composite service which include the notification and dataflow dependencies, and it can effectively support the properties of modularity, interoperability, dynamic reconfigure-ability and fault-tolerance for a composite software system in the dynamic Web environments of business applications.

The outline of this paper is as follows. Section 2 firstly addresses our novel notions and notations: composition structure, which has four basic building blocks such as and-split, or-split, and-join and or-join. Section 3 comes up with our specification mechanism for the services composition, and last two sections present the related work, summarize our contributions and conclude the paper.

2. The Composition Structures

As we know, the dynamism and unpredictability of the business applications and Web environment require that the Web services composition have the ability to adapt to unknown situations. We believe that some form of enhanced features from reflection and process modeling will provide the right ingredients to enable adaptability.

Reflection [5] is the act of revealing a system's implementation, and allowing changes to that implementation in a controlled manner. By inspecting internal aspects of a system, its competence may be improved, either through better performance or greater adaptability. The concepts of reflection have been applied in many areas of computing and information technology, where it has been employed as a means of building up flexible and extensible computer systems – ones that can evolve and adapt to changing circumstances and expectations [6]. In this paper we apply to the concept of reflection and inspect the internal aspects of a composite service, which includes two main parts: the structure of a

composite service and the internal dependencies of a composite service. Correspondingly, we come up with our solutions in order to tackling these two parts. We believe our solutions tackle the most critical aspects of a composite service, and satisfy the requirements of the flexibility, adaptability and reliability for a composite service.

During the process modeling of a composite service there is a vital aspect and a hard point that needs to be firstly tackled: that is how to deal with the joints among the constituent services. Here we would like to propose our notions: composition structures based on the concept of task structures. As we know, a composite Web service involves the completion of several Web services. These several services need coordinate to complete a specific task. This kind of coordination is actually the coordination of several tasks, which has the process style. For example, Flight-booking service need to be finished before Accommodation-booking service commences. After finishing Accommodation-booking service and Attraction-searching service, the composite service needs to choose one service between Bicycle-booking service and Car-rental-booking service. After that, Events-planning service takes place. Processes focus on the coordination of tasks. Any process specification language should at least be capable of capturing sequential composition, moments of choice, parallel composition, and various forms of synchronization. Task Structures [7] is notions and notations for describing the various tasks within a process, and their interdependencies. In this paper we enhance the notions and notations of task structures specifically for the Web services composition. Here we only apply to these enhanced notions and notations to generate and model the structure of a composite Web service, and use another solution to specify the inter-dependencies among the constituent services of a composite Web service. We believe this kind of treatment is more reasonable, because the composing structure and the dynamic inter-dependencies of a composite Web service are two different facets of a composite service and have their own characteristics. It deserves to obtain the different treatments and apply to the different approaches and solutions.

Thus the composing structure of a composite Web service may be composed from one or more of five particular building blocks, here we call them "***Composition Structures***". (Naturally, we might introduce more composition structures similar to process patterns in workflow research community [8,9,10]. However, here our purpose is to generate and model the structure of the coordination of several Web services, which is different from modeling the control structure of workflow, although they have some similarities. After all, the composing structure of a composite service is different from the control structure of workflow. Moreover, over-introducing the building blocks might lose flexibility and manageability for a composite service. This aspect of research is beyond the scope of this paper).

Sequential-composition: After Service A has been performed, only Service B will be performed. This is called **Sequential-composition** between two services.

AND-split: After Service A has been performed, both Service B and Service C will be performed. Service A is said to ***trigger*** the other services.

OR-split: After Service A has been performed, either B or C, but not both, will be performed. This is called a ***decision point***.

AND-join: Service C cannot be performed until both A and B have been performed. This form of join is called a ***synchronizer***.

OR-join: Service C will be performed whenever either A or B, but not both, have been performed. This form of join is called a ***discriminator***.

The above notations are not limited to relations among three services, and they might be extended more than three services. For example, Service A can **trigger** more than two parallel services at the same time.

Please note the above notations are not only notations, but also they are notions, since they express a kind of semantics and satisfy the requirements of process specification language which captures sequential composition, moments of choice, parallel composition, and various forms of synchronization.

3. Specifying the Internal Dependencies of a Composite Service

3.1 Specification Approach

In terms of the composition structures notions, the reliable structure of a composite service can be generated and formed. In order to describe the dynamic dependencies of a composite service, and there is the second part of work which needs to be done. We need to clearly specify two vital inter-dependencies of a composite service: the ***notification dependency*** and the ***dataflow dependency.*** The notification dependency means that there exists the sequential relation between two services or two services structures. For example, Service 2 can't be started if Service 1 has not completed; the dataflow dependency means that there exist the data-flowing relations between two services or two services structures. For example, Service 2 needs the data of Service 1; Accommodation-booking service needs the data of the Flight-booking service such as the date, location and the number of clients, etc. We assume that every dataflow dependency also has the notification dependency.

Notification Dependencies. Each notification dependency takes the form:

Notification from {…}

For example, Service 2 can't be started until Service 1 is successfully completed.

Service 2

{
Notification from
{Service 1 if Service 1 success}
}

Dataflow dependencies. If we use the textual representation in the dataflow dependencies, each dataflow dependency takes the following form:

Service 2
{
Inputs {input-X from
{output-x of Service 1};
input-Y from
{….};
…
}

For example, the input-A of Service 2 needs the output-a of Service 1; and the input-B of Service 2 needs the output-b of Service 1.

Service 2
{
Inputs{input-A from
{output-a of Service 1};
input-B from
{output-b of Service 1}
}
}

3.2 Example

Now that we have the specification mechanism to specify the composing structure and dynamic inter-dependencies of a composite service, a complete specification of a composite service can be formed. If we take the "Travel Solutions" example, its complete textual specification can be written in the following:

Composite-service "Travel Solutions"
{
AND-split from beginning splitting to Flight-booking service and Attraction-searching service
{
Service Flight-booking
Inputs {date; location; the quantity of the tickets; …},
Service Accommodation-booking
Notification from {Service Flight-booking if it is successful}
Inputs
{outputs (date; location; the quantity of the tickets; …) of Flight-booking service}
};
Service Attraction-searching
Inputs {location, …},
AND-join from Accommodation-booking service and Attraction-searching service to a decision point (OR-split),
OR-split from Accommodation-booking service and Attraction-searching service to Bicycle-booking service or Car-rental-booking service,
Service Bicycle-booking
Notification from
{Service Accommodation-booking and Service Attraction-searching if they are successful}
Inputs{…};
Service Car-rental-booking
Notification from
{Service Accommodation-booking and Service Attraction-searching if they are successful}
Inputs{…},
OR-join from Bicycle-booking service or Car-rental-booking service to Events-planning service,
Service Events-planning
Notification from
{Service Bicycle-booking or Service car-rental-booking if one of them is successful}
Inputs{outputs(date;location;…) of Attraction-searching service}
}

Similarly, the above textual specification also can be drawn in a graphical manner with the textual annotations. Because of the limitation of space, here we will not draw the complete graphical specification blueprint.

4. Related Work

Most of the work in service composition has focused on using workflows either as a engine for distributed activity co-ordination or as a tool to model and define service composition. Representative work is described in [14] where the authors discuss the development of a platform specifying and enacting composite services in the context of a workflow engine. The eFlow system provides a number of features that support service specification and management, including a simple composition language, events and exception handling.

The work related to Web services and co-ordination or composability can be found in [25]. In this paper the authors examine the potential of using coordination technology to model electronic business activities and illustrate the benefits of this approach.

Our work in this paper is different from the above-mentioned work. Our main idea is to separate the design and implementation of a composite service, and concentrate on the research of composition logic for the composite services. Correspondingly, we come up with our sound solutions for the service composition. We believe the advantages of this way are to greatly facilitate discovery of Web services, since the generated structure by using our approaches maps the services we need to discover. Moreover it possesses more flexibility and adaptability, and the structure of a composite service may be easily changed and adapted according to the user's realistic needs. Our work satisfies and supports the main requirements of a flexible composite software system such

as modularity, interoperability, flexibility, adaptability and fault-tolerance and dynamic reconfiguration in the dynamic Web environments of business applications. Our concrete approaches are related to the following research fields:

Generic process models: In [27], each workflow schema is associated with a family of variants. A particular task in the schema may be viewed as the root of an extensible class hierarchy, with the hierarchy expressing allowable instantiations of that particular task.

Workflow pattern: In the work of [21], a systematic overview of process control constructors is provided. The patterns address business requirements in an imperative workflow style, but are independent of any particular workflow language. They encapsulate commonly used forms of complex workflow functionality.

Architecture Description Languages: Software architecture specification is intended to describe the structure of the components of a software system, their interrelationships, and principles and guidelines governing their design and evolution [15, 16, 17]. It is common to model an application as a set of components communicating through connectors. Typically, an application is composed from components, where a component provides services to other components. A component within an application can be either a simple component, or composed out of a group of other components. The components provide and obtain service through ports. The interaction between ports can take many forms, for example, buffered message passing, one-to-many event dissemination, or synchronous request-reply communication. Currently available ADLs however do not capture the computation unit structure of a composite application. This requires describing the structure and inter-dependencies of an application. Our approaches capture this structure in terms of composition structures notations and their dependencies by specifying input and output requirements. Another advantage of describing composite structure in terms of services is that it directly enables application level fault tolerance requirements to be specified and controlled.

5. Concluding Remarks and Future Research Directions

In this paper we come up with our sound solutions for the composition logic of the composite services, which is referred to the way a composite service is constructed in terms of its constituent services. Our ideas are to separate the design and implementation phases of a composite service, and concentrate on the research of composition logic of the composite services. The advantages of this treatment are to make the composite services have more flexibility and adaptability, which greatly facilitates the discovery of the constituent services, since the structure of a composite service we generate maps the discovery of its constituent services. Our concrete approaches are that we firstly propose well-defined notions "composition structures" to represent the joints among the constituent services and generate the reliable structure of a composite service, and then we provide a specification mechanism to clearly specify the internal dependencies of a composite service in terms of the notification and dataflow dependencies. Our work is motivated by the main requirements of a flexible composite software system such as modularity, interoperability, flexibility, adaptability and fault-tolerance and dynamic reconfiguration in the dynamic Web environments of business applications. Our solutions and specification mechanisms satisfy these requirements and support these properties. The concrete reasons are summarized in the following:

- We view every service as an independent computing unit. Our notions, notations and specification mechanism are applied to easily generate the structure of the composite computing unit (or composition structure or service) and clearly specify the inter-relations of these constituent computing units. So our solutions support the modularity and interoperability of the composite system.
- Our "composition structures" notions and notations provide a kind of flexible, adaptable mechanism to address the composition logic of the composite system, which means the structure of a composite service can be easily and reliably adapted if the users' needs are changed.
- Our solutions provide a reliable specification mechanism for the implementation of a composite service. We believe only we have the clear specification of a composite service in an abstract manner, then it can support the properties of fault-tolerance and dynamic reconfiguration for a long running composite service on the Web.

Based on our above work, future work might include two research directions: one is to continue to research the composition logic of Web services, specifically address the composition logic in a more formal way; another is to develop the corresponding application tools or languages for our specifications of the composite services.

References

[1] F. Casati, M. Sayal, and M-C Shan. Developing E-Services for Composing E-Services. Procs. 13th CaiSE conference, Switzerland, 2001.

[2] C. Bussler. The Role of B2B Protocols in Inter-Enterprise Process Execution. Procs. 2nd VLDB-TES Workshop, Rome, 2001.

[3] H. Kuno, M. Lemon, A. Karp, and D. Beringer. Conversations + Interface = Business Logic. Procs. 2nd VLDB-TSE Workshop, Rome, 2001.

[4] Web Service Definition Language. http://www.w3.org/TR/wsdl

[5] Kiczales, G., des Rivieres, J. & Bobrow, D.G., The Art of Metaobject Protocol, The MIT Press, Cambridge, Mass., USA. 1991.

[6] Maes, P., "Computational Reflection", The Knowledge Engineering Review 3(1), 1-19. 1988.

[7] Hofstede, A. t. & Nieuwland, E. "Task Structure Semantics through Process Algebra", Software Engineering Journal 8(1),1993.

[8] W.M.P. van der Aalst, Don't go with the flow: Web services composition standards exposed. Web Services-Been there done that? Trends of Controversies. Jan/Feb 2003 issue of IEEE Intelligent Systems.

[9] W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed, Pattern Based Analysis of BPML (WSCI), QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.

[10] W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed, Pattern Based Analysis of BPEL4WS, QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

[11] B. Benatallah, M. Dumas, M.C.Fauvet and H.Y.Paik, Self-Coorinated and Self-Traced Composite Services with Dynamic Provider Selection. UNSW-CSE-TR-0108, The University of New South Wales, Sydney, 2001.

[12] W.M.P van der Aalst, A.P.Barros, A.H.M. ter Hofstede and B. Kiepuszewski. Advanced workflow patterns. In Proc. of the 5th IFCIS Int. Conference on Cooperative Information Systems, Eilat, Israel, September 2000. Springer Verlag.

[13] Buschmann, F., Meunier, R., Rohnnert, H., Sommerlad, P. & Stal, M. (1996), Pattern-Oriented Software Architecture – A System of Patterns, Wiley and Sons Ltd., USA.

[14] F.Casati, S. Ilnicki, L.-J.Jin, V. krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eFlow. In Proc. Of the International. Conference on Advanced Information Systems Engineering (CaiSE), Stockholm, Sweden, June 2000. Springer Verlag.

[15] F. Ranno, S.M. Wheater and S.K. Shivastava, "A System for Specifying and Co-ordinating the Execution of Reliable Distributed Applications", Conference on Distributed Applications and Interoperable Systems (DAIS'97), Germany, 1997.

[16] J. Magee, N. Dulay, S. Eisenbach and J. Kramer, "Specifying Distributed Software Architectures", Proceedings of the 5th European Software Engineering Conference, Barcelona, 1995.

[17] J. Magee and J. Kramer, "Dynamic Structure in Software Architectures", SIGSOFT 96, ACM Software Engineering Notes, Vol 21 No. 6, November 1996.

[18] BPML.org. Business Process Modeling Language (BPML). Accessed from www.bpml.org, 2002.

[19] BPML.org. Business Process Modeling Notation (BPMN), Working Draft. Accessed from www.bpmi.org, 2002.

[20] Microsoft Corporation. Xlang web services for business design. Accessed from www.gotdotnet.com, 2001.

[21] Aalst, W.v.d., Barros, A., Hofstede, A.t.& Kiepuszewski, B. (2000), Workflow patterns, Technical reports, Queensland University of Technology.

[22] B. Benatallah, M. Dumas, M-C. Fauvet, F.A.Rabhi. Towards Patterns of Web Services Composition. Technical Report, University of New South Wales, Sydney. UNSW-CSE-TR-0111. November 2001.

[23] 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 the International Conference on Data Engineering, San Jose, USA, February 2002.

[24] Curbera F., Nagy W. and Weerawarana S. 2001. Web Services: Why and How?. *http://researchweb.watson.ibm.com/*

[25] G.A. Papadopoulos and F. Arbab. Modelling Electronic Commerce Activities Using Control-Driven Coordination, Ninth International Workshop on Database and Expert Systems Applications, Vienna, Austria, 1998, IEEE Press.

[26] J. Yang and P. Papazoglou. Web Component: A Substrate for Web Service Reuse and Composition. In Proceeding of CAISE'02, 2002.

[27] Aalst, W.v.d., How to Handle Dynamic Change and Capture Management Information, in "Proceedings of the 4th IFCIS International Conference on Cooperative Information Systems", pp. 115-126. 1999.