An On-the-fly Approach to Web-based Service Composition

Qi Zhao, Gang Huang, Jiyu Huang, Xuanzhe Liu, Hong Mei
Key Laboratory of High Confidence Software Technologies, Ministry of Education
School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China;
{zhaoqi06, huanggang, huangjy07, liuxzh}@sei.pku.edu.cn; meih@pku.edu.cn

Ying Li, Ying Chen
IBM China Research Laboratory, Beijing 100094, China; {lying, yingch} @cn.ibm.com

Abstract

The web-based service composition, e.g. mashup, is becoming a popular style to reuse web services. From the perspective of reuse, existing work has limitations on qualifying whether the service or the service composition satisfies user requirements and adapting the service or composition according to the qualification results. For addressing these limitations, this paper proposes an on-the-fly approach to web-based service composition. Firstly, we do not distinguish the design-time and run-time of services and their composition so that they can be qualified in a what you see is what you get manner when services are selected or assembled. Secondly, we propose a component model for separating the service business and user interface so that they can be changed dynamically and independently in the adaptation of service selection and composition. This approach is demonstrated by a browser-based mashup tool.

1. Introduction

Currently, there are many services published via Internet with open APIs. More and more developers are able to access these services through web and assemble them to construct their own applications. Many existing work [1][2] provided web-based service composition environments, in which the service business logic and User Interface (UI) are encapsulated into a single component, called web-based service (WBS) components. However, current WBS components are strict separated between design-time and run-time. This separation brings some serious limitations on software reuse, especially for qualifying WBS components and their composition.

Firstly, developers should qualify each WBS component before assembling them. Each component has its own pros and cons. In particular, rich user experience is one of the most important features for the web-based service composition. Therefore, it is hard to qualify one component whether it satisfies the requirement, without really using it. Nevertheless, the separation between design-time and run-time prevents developers from qualifying WBS components effectively. Because when components are loaded into the environment, they cannot be qualified directly since they are just in design-time and do not have real functionality.

Secondly, the composition of these components should be qualified as well. However, there are many mismatches, such as two different date strings formats, cannot be found if the composition does not execute really. The separation between design-time and run-time makes the mismatch hard to be found.

Unlike traditional off-the-shelf components that need compilation and deployment, services are actively running entities [3]. Such a significant difference implies that it is possible to assemble services in an on-the-fly manner, that is, 1) when a service is loaded into the composition environment, it becomes available with real appearance and functionality immediately, 2) when two services are assembled, they can interact with each other actually. Then the services and composition can be qualified in a "what you see is what you get" way.

Furthermore, qualification is always coupled with adaptation If on-the-fly adaptation is not supported, developers may need to modify component source codes, redeploy components and reload them into the environment. Such offline adaptation makes on-the-fly composition impossible. Therefore on-the-fly adaptation ability is required.

2. Approach Overview

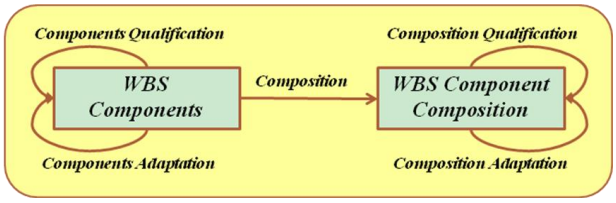

Figure 1 On-the-fly Approach Overview

Our component model is not distinguished as design-time and run-time. Once loaded, each component is at runtime with full and actual functionality. Therefore developers can qualify components by really using it. If components do not satisfy the requirements, developers could adapt them on-the-fly. On the other hand, since components are always at runtime. When developers assemble them, components can be connected and services can be invoked just-in-time. This enables developers to qualify current composition immediately.

3. Web-based Service Component Model

Our WBS component model consists of two parts, interface and implementation. The interface of component model consists of User Interface and Programming Interface since WBS component model encapsulates UI. The Programming Interface exposes business logic of WBS component. The UI responds to users' actions and invokes the corresponding functions in the implementation.

The implementation of our WBS component model adopts the Model-View-Controller pattern. The controller part consists of several element controllers. Each element controller encapsulates the interaction logic of one specific UI element.

The programming interface exposes the business logic of WBS component. It consists of properties, methods and events. On the other hand, our WBS component UI makes elements in UI as the fixed part, while presentation and structure of the elements as the variable part. The structure defines the elements' place and their relationship (i.e. parent-child), while the presentation defines the elements' presentation information such as size and color. When developers reuse the WBS component, they can adjust configurations to adapt UI with particular scenario.

Another problem which prevents on-the-fly adaptation is how to adapt the interaction logic while keeping the component correct when some elements are changed or removed. According to the controller of our WBS component model is divided into element-combined controllers, UI elements can invoke proper business logic and respond to return results through a specific controller no matter where it is placed and how it is presented. Moreover, if a UI element has been removed, the specific controller will be removed automatically while not affect the other UI elements and controllers as well.

4. On-the-fly Service Composition

In our approach, dynamic languages and the separation between design-time and run-time are the most important in design rationales, which lead to a web-based service composition environment with on-the-fly capabilities.

The first step of service composition in our web-based composition environment is retrieving components. Our WBS component model does not distinguish between design-time and run-time. Also, our composition environment does not follow the design-run-debug cycle. The composition environment is hosted in a web browser and implemented by JavaScript, which is a dynamic script language. Hence, the environment can load WBS component definitions without restarting. Also, because a WBS component is implemented by JavaScript, a component can be instantiated without compilation and deployment once its definition is retrieved into composition environment. After being instantiated, a component is at runtime. It connects with a running service and can respond to user actions with full functionalities.

Our environment provides publisher/subscriber connector for developers to assemble components directly. Moreover, data connector is also provided to handle data format mismatch. Since every component is always at runtime, components can be connected just-in-time when developers assemble them. Service invocations may be caused by this connection and these invocations may modify component internal states and UI. Thus developers can qualify the assemble results immediately.

Besides of programming interface composition, our composition environment also allows users to assemble user interface of WBS components. The UI elements of WBS components will be merged into a new UI when developers assemble them. Then developers need to adjust the structure and presentation configuration of this new UI. The new UI's configuration is the same as the atomic component UI's. Furthermore, to support the on-the-fly UI composition, our environment keep a detection on modifications of UI's configuration. Once a new configuration is set, it will be immediately applied on the environment and the new UI will be displayed.

Acknowledgements

This effort is sponsored by the National Key Basic Research and Development Program of China (973) under Grant No. 2005CB321805; the National Natural Science Foundation of China under Grant No. 90612011; and the IBM University Joint Study Program.

References

[1] J. Yu et al. A Framework for Rapid Integration of Presentation Components. In the Proceedings of WWW'07, Banff, Canada, May 2007.

[2] Yahoo! Pipes: http://pipes.yahoo.com

[3] Girish Chafle, Gautam Das, Koustuv Dasgupta, Arun Kumar, Sumit Mittal, Sougata Mukherjea, Biplav

Srivastava. An Integrated Development Environment for Web Service Composition. In the Proceedings of IEEE International Conference on Web Services 2007.