A Browser-based Middleware for Service-Oriented Rich Client

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

Abstract

Along with the proliferation of web-delivered services and the wide adoption of popular Web technologies, it has been an emerging development style that composes service-oriented applications with rich user experiences in the web browser. Currently, these service-oriented rich client (SoRC) applications are usually tightly coupled with specific requirements and scenarios, without the solutions of common problems for development, deployment and operation. It leads to the fact that SoRC applications are exactly done in an ad-hoc manner. In this paper, we propose a new type of middleware, which is embedded in web browsers and encapsulates reusable solutions for common problems. This browser-embedded middleware consists of a container managing component instances, a set of communication mechanisms coordinating both browser-server and inter-browser interactions. Different SoRC applications can be constructed more easily based on the middleware. In the case study, we construct a mashup environment, called iMashup, with the middleware and compare it with some popular environments. The comparison shows that iMashup provides composition capabilities with less implementation efforts, occupies much lower memory consumption and achieves more scalability.

Keywords

service composition, rich client, mashup, middleware

1. Introduction

Over the past years, browser-server architecture was evolved a lot. At server-side, many Web sites and applications, such as Google, Amazon and Facebook, has adopted service-oriented architecture (SOA) more or less. Data and functionalities can be accessed through Web-delivered services (SOAP and RESTful Web services, RSS/Atom feeds, or open JavaScript APIs). At client-side, although web browser was originally a pure thin client to visit static web pages, it has already become much "richer" by offering a series of powerful hosted mechanisms, such as Dynamic HTML (DHTML) and JavaScript engine. This makes the browser be capable of serving as a "***rich client platform***". Therefore developers are now assembling various services to create a plethora of service-oriented rich client (SoRC) applications to solve all types of problems [1].

Figure 1. Service-Oriented Rich Client

Service-oriented rich client aims at supporting service discovery, subscription and composition in web browsers. Figure 2 shows the trend toward the continuously increasing complexity of SoRC application:

- Some relatively simple SoRC applications, such as Google map or Yahoo weather, only subscribe (use) certain built-in web-delivered services in web browser.
- More complex SoRC applications, such as Web portal or Web OS [10], allow users to not only subscribe built-in services, but also they allow users to discovery required service and import them into the applications in browser.
- The browser-based mashup applications, e.g. HousingMaps and WeatherBonk [11], is a kind of SoRC application, which subscribes services and assembles them to create a new service in browser.
- The most complicated SoRC applications fully support service discovery, subscription and composition. These applications are assembled by a series of mashup editor services, through which users can discover, subscribe and assemble other services to create their own mashup applications. Thus, these applications can be considered as an incubator of browser-based mashup applications [6].

Corresponding author: liuxzh@sei.pku.edu.cn

Figure 2. The trend toward the increasing complexity of SoRC application

Figure 3. Many common problems for developing, deploying and operating SoRC applications

Due to the complexity of SoRC, there are many common problems for developing, deploying and operating SoRC applications, including utilization of rich client features, interaction with remote and local services, client-side data management, components composition and quality assurance, as shown in Figure 3. Poor solutions for these common problems definitely put negative impacts on SoRC applications.

However, the hosted mechanisms of web browser are designed for general purposes and cannot deal with these common problems directly. Hence SoRC vendors have to handle them. Since SoRC applications are just in a very early stage, the solutions for common problems are specific to the applications and tightly coupled with their own requirements and scenarios. Obviously, such an ad-hoc manner makes it hard and even impossible for a single vendor to produce optimal or best-of-the-breed solutions. But the monolithic and private design and implementation of each application prevent vendors from sharing and collaborating on the private solutions. On the other hand, building up new SoRC applications has to implement these common solutions again and again. It increases the cost of building new applications.

In this paper, we propose a new type of middleware, which is embedded in the browser and encapsulates the solutions of common problems for developing, deploying and operating SoRC applications. Such middleware provides an open way for producing the optimal solutions in most cases. We implement a browser-based service composition environment, iMashup [1], based on the middleware, and compare it with some other environments [3]. The comparison and evaluation results demonstrate the values of this middleware, that is, better solutions for common problems and easy construction of SoRC applications.

The rest of the paper is organized as follows. Section 2 provides the overview of the browser middleware. Section 3 discusses the implementation of the middleware. Section 4 briefly introduces our case study for this middleware, iMashup. Finally, we give some future work in Section 5.

2. Browser Middleware Overview

Figure 4 provides a general overview of the proposed browser middleware.

[1] It can be downloaded from http://code.google.com/p/imashup/

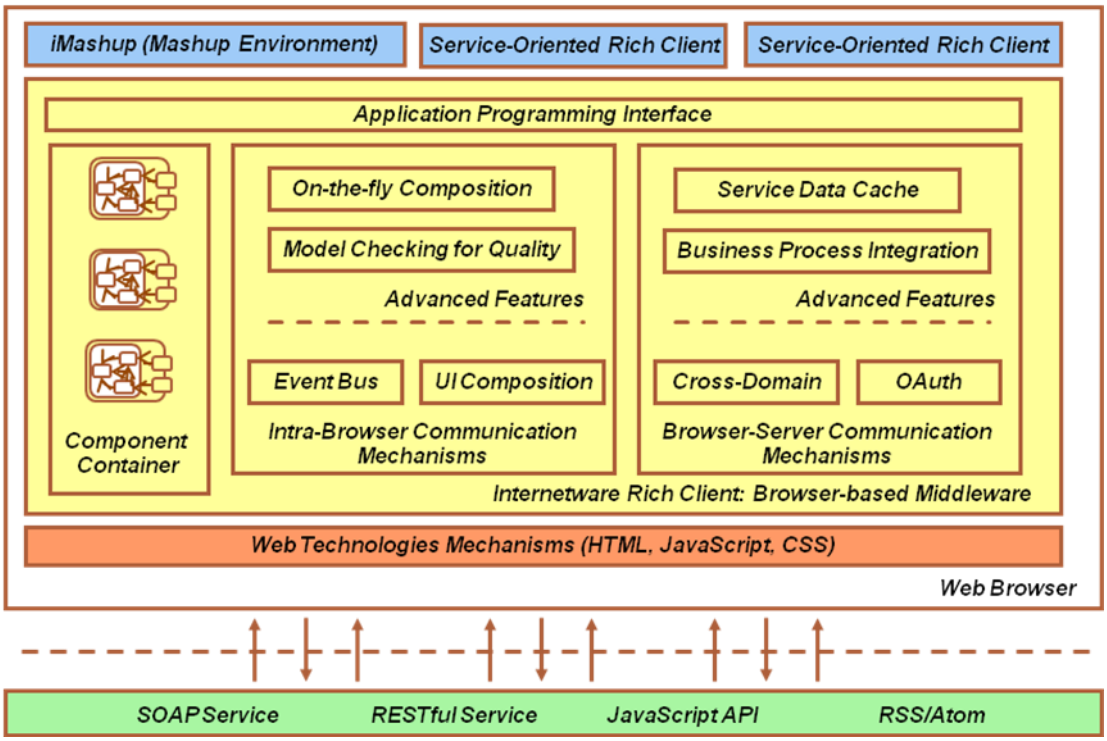

Figure 4. The Overview of the Browser Middleware

The browser middleware is built on the top of the hosted mechanisms of web browser, such as HTML and JavaScript engine. It is mainly composed of four parts:

The container is in charge of component management. Its responsibility is to manage the definitions and the instances of component. The container need not consider the concurrency control of component instances, since the JavaScript engine in the browser runs in single-thread;

The communication mechanisms assist components to communicate with other components inside web browser and web-delivered services on servers. The basic features of the communication mechanisms, such as event bus, cross-domain and OAuth handler, have been described and implemented in [3]. However, the communication mechanisms also provide many advanced features, such as on-the-fly composition or service data cache, which are not included in existing JavaScript frameworks. Depending on the different communication types, the communication mechanisms can be divided into two parts:

- **The intra-browser communication mechanisms** offer some mechanisms to implement an event-based composition model which makes components communicate with others in the same web browser.
- **The browser-server communication mechanisms** provide the solutions of common problems for communication between the browser and web-delivered services on servers. The mechanisms offer a series of handlers supporting the common capabilities for web-delivered services access.

The middleware provides a set of **Application Programming Interface (API)** to expose its functionalities. Developers can build SoRC applications directly based on these APIs.

3. Implementation of the Browser Middleware

3.1 Component Model

Components in the browser middleware encapsulate both the application logic and UI [7]. These components consist of the interface and the implementation. The interface of components comprises the UI (user interface) and the programming interface. The programming interface exposes application logic. The UI responds to users' actions and invokes the corresponding functions in the implementation. The component model is shown in Figure 5.

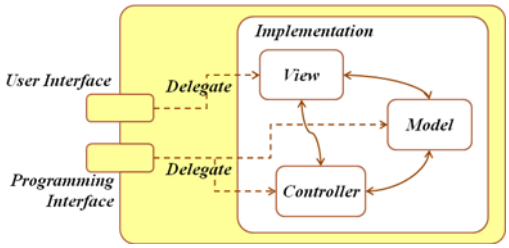

Figure 5. Component Model

The implementation of components adopts the Model-View-Controller pattern. The model implements application logic by invoking web-delivered services. The

view is a fragment of HTML which is rendered and displayed when components are instantiated. The controller manages the interaction logic between model and view.

The programming interface exposes the application logic of components. It consists of methods and events. Methods can invoke services and query and modify the component state. Events notify changes of the component state and can be published into the event bus. The UI is implemented by the view part of component. When developers assemble the components, they can determine whether the UI of components should be shown or not (hidden).

3.2 Event-based Composition Model

The intra-browser communication mechanisms provide an event-based publish-subscribe composition model. The event-based composition model is well suited to browser-based composition environments, since the nature of the browser is strongly event-based [2].

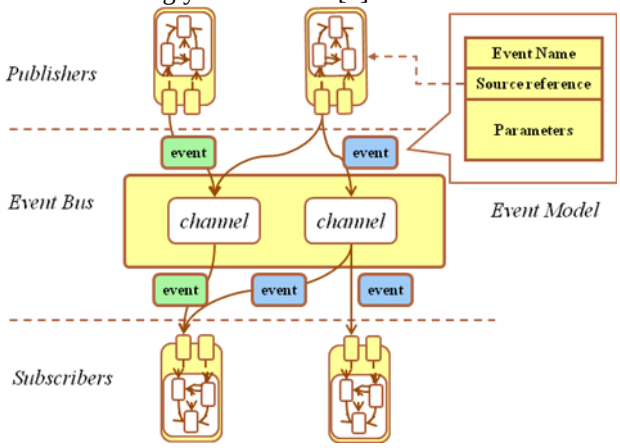

Figure 6. Event Bus and Event Model

To support event-based composition model, the intra-browser communication mechanisms provide a unified event model for all events of components. The event model comprises the name of event, the reference of source instance triggering this event and a hash map containing event parameters.

An event bus which supports publish-subscribe event binding is also provided by the intra-browser communication mechanisms. The event bus has some channels which are set by developers. The events of components can be published to these channels. And the methods of components can subscribe channels.

When an event is triggered, the component sends the event to the specific channel in the event bus. When an event arrives, the event bus will find all subscribe methods with the given channel, traverses the found methods and invokes them with the event as input parameter.

3.3 Advenced Features of Communication Mechanisms

3.3.1 On-the-fly composition

Unlike traditional off-the-shelf components that need compilation and deployment, services are actively running entities. Such a significant difference implies that the separation between design-time and run-time is not an inherent nature of web-delivered service composition. In other words, it is possible to assemble services in an on-the-fly manner.

The browser middleware supports an on-the-fly approach to service composition [5][6]. This approach does not distinguish the design-time and run-time of services and their composition so that developers can qualify the results in "what you see is what you get" manner when services are selected or assembled.

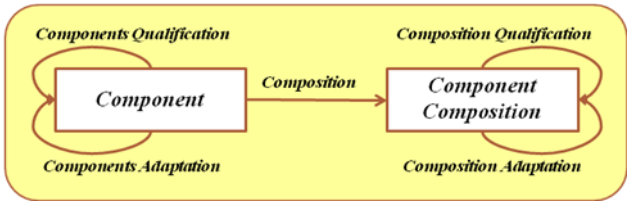

Figure 7. On-the-fly Approach Overview

Figure 7 gives an overview of the on-the-fly service composition approach. Components in the browser middleware are 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. 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 composition immediately.

3.3.2 Model Checking for Quality

Figure 8. An incorrect status of service-oriented rich client

Components in the browser middleware are developed by independent organizations and individuals due to the open nature of Web and SOA. As a result, when these components are assembled, the undesired interactions will

take place and bring negative impacts. For example, Figure 8 displays an incorrect status of SoRC application which is due to the undesired order of Ajax invocations' responses [8].

The browser middleware supports a model-checking based approach for qualifying service composition [4]. At first, the behavior of SoRC application will be automatically specified by analyzing the JavaScript codes downloaded. And it will be combined with the pre-defined environment behavior so that a precise and complete enough behavior model of the application can be generated automatically. With user-defined constraint and refinement specifications, the behavior model is automatically translated to the formal specification (Promela for Spin) as the input of the model checker. If the model is flawed, the SoRC application has correctness and reliability problems.

3.3.3 Service Cache

Due to the fact that a number of web-delivered services fetch data from servers, SoRC applications should employ browser cache in order to improve the performance issues (e.g. traffic and latency). To this end, cache strategies are customized by service providers and brought into effect by browsers. Such pattern of cache leaves little space for developers, who are exactly responsible of composing services, to carry out their own cache strategies. The absence of customized cache strategies may cause unnecessary communications or reduce user experience.

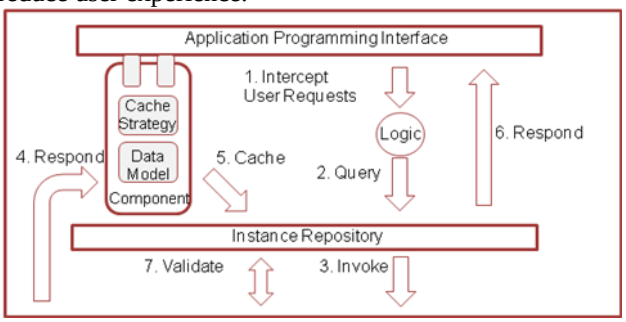

Figure 9. Cache Framework at Run-time

For addressing this limitation, the browser middleware includes a cache framework for composing web-delivered services. The cache framework enables developers to customize their own cache strategies, such as expiration time, cache granularity and so forth. And then at run-time, as shown in Figure 9, the framework first intercepts user requests and then query the cache repository whether a cache hit or not. If a cache data is hit, the framework returns the desired data. If no hit produced, the framework invokes the remote services, parse and store the returned data into cache repository for future use.

3.3.4 SoRC and Business Process Integration

In practice, many business process scenarios require user interaction. However, human computer interactions (HCI) are not covered by current server-side service composition approaches, such as WS-BPEL, which are primarily designed to support automated business processes. To our experience, SoRC applications are well suited for HCI process composition, because of its advantages in rich user experience, user friendliness, UI deep integration and many other features, which are required by HCI process composition.

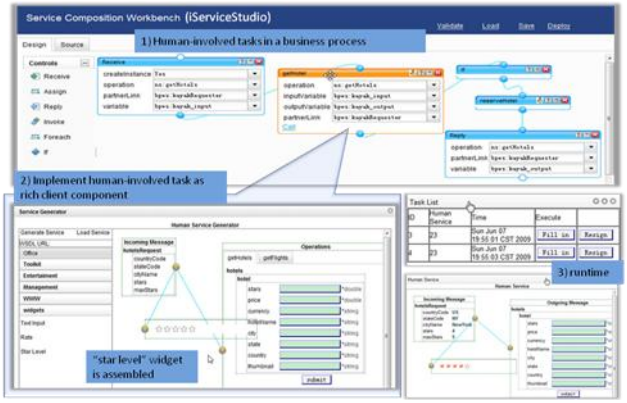

Figure 10. The rich client and business process integration mechanism.

The browser middleware provides a SoRC and business process integration mechanism to support HCI in business process. The mechanism generates SoRC application stubs by parsing the input and output of HCI activities in business process. Developers can refine the stubs to build user-friendly SoRC applications. When the business process running to a HCI activity, the integration mechanism will create an application instance, send the input parameters and notify users of waiting process. When users complete the HCI activity, the mechanism will get the output parameters and return them to the process. The process will be continued its execution.

4. Case Study: iMashup

Owing to the limitation of space, we just give a brief introduction of our case study. More detail about the case study can be found in [9].

To evaluate our browser middleware, we implement iMashup, a mashup environment based on the middleware. To our experience, the main work of environment implementation is building GUI, since the middleware provides most functions an environment required. We compare iMashup with other environments. We focus on two questions: How is the overhead of iMashup? How is the scalability of iMashup?

4.1 Overhead of iMashup

For our first question, the overhead of iMashup, we measured the size of files downloaded and the memory-

consumption. We also compare the result with the overhead of three other composition environments, Microsoft Popfly, Intel Mash Maker and Yahoo! Pipes.

From the evaluation, the download size of iMashup is the biggest of the four environments (iMashup, 603kb; Microsoft Popfly, 347kb; Intel Mash Maker, 259kb; Yahoo! Pipes, 112kb). iMashup is the biggest mainly because the browser middleware is relatively big. It is a typical result when we compare applications with and without middleware. The size of our middleware is a bit large, which means iMashup may start up slower than other environments. Yet the evaluation indicates the memory-consumption of iMashup is even smaller than the others.

4.2 Scalability of iMashup

We consider scalability of iMashup from two perspectives: How is the memory-consumption of component instances in the container? How is the performance of the event bus?

First, we measure the changing memory-consumption of iMashup with increasing numbers of component instances. We test a typically component, a Google Weather component calling RESTful service. From the evaluation result, the memory-consumption of 200 instances is lower than the consumption of Gmail, which is a widely used web application with complex logic executed in the browser.

Second, we measure the performance of the event bus. For this measurement, we set a one-to-many event binding: one event publisher and multiple event subscribers. And then we trigger an event and measure the time spent from the event triggered until the last subscriber receives it. The evaluation result shows the event bus routes the event extremely fast, i.e. handling 25,000 subscribers within 350 milliseconds.

5. Future Work: From Rich to Synergized

Web technologies proved to be well suited for building browser-based SoRC applications. Therefore, many frameworks and runtimes, or even operating system, allow developers to use web technologies to build not only rich client in web browser but also standalone and mobile rich client. For example, Adobe Integrated Runtime (AIR) is a cross-platform runtime environment for building rich clients using web technologies, which can be deployed as a desktop application. And Palm webOS is mobile operating system, on which applications are built using web technologies, but have access to device-based services and data.

These frameworks and runtimes provide more powerful capabilities to SoRC, such as local data access, multi-thread JavaScript engine and so on. Standalone and mobile SoRC applications meet new common problems hence, such as intelligent installation and update, user awareness, offline mode support and so on.

In the future, we will try to integrate new solutions of these new common problems into the SoRC middleware and make the middleware become synergized.

Acknowledgements

This work is partly sponsored by the National Key Basic Research and Development Program of China under Grant No. 2009CB320703; the National Natural Science Foundation of China under Grant No. 60873060, 60933003; the High-Tech Research and Development Program of China under Grant No. 2009AA01Z16; the National S&T Major Project under Grant No.2009ZX01043-002-002; the Program for New Century Excellent Talents in University.

References

[1] E. Michael Maximilien, Ajith Ranabahu, Karthik Gomadam, An Online Platform for Web APIs and Service Mashups. IEEE Internet Computing, 2008.

[2] Jin Yu, Boualem Benatallah, Fabio Casati, Florian Daniel, Understanding Mashup Development. IEEE Internet Computing, 2008.

[3] Gang Huang, Qi Zhao, Jiyu Huang, Xuanzhe Liu. Towards Service Composition Middleware Embedded in Web Browser. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2009.

[4] Xiangping Chen, Gang Huang, Hong Mei. Towards Automatic Verification of Web-based SOA Applications. 10th Asia Pacific Web Conference (APWeb), 2008, pp. 528-536.

[5] Qi Zhao, Gang Huang, Jiyu Huang, Xuanzhe Liu, Hong Mei, ,Ying Li, Ying Chen. A Web-Based Mashup Environment for On-the-Fly Service Composition. SOSE 2008, pp. 32-37.

[6] Qi Zhao, Gang Huang, Jiyu Huang, Xuanzhe Liu, Hong Mei, Ying Li, Ying Chen. An On-the-fly Approach to Web-based Service Composition. Proceedings of IEEE Service Congress and International Conference on Web Services, 2008, pp 208-209.

[7] Qi Zhao, Gang Huang, Xuanzhe Liu, Jiyu Huang, Towards a Component Model for Web-based Service Composition. Journal of Frontiers of Computer Science and Technology, 2008, 2 (4): 378-388.

[8] Qi Zhao, Jiyu Huang, Xiangping Chen, Gang Huang, Feature Interaction Problems in Web-based Service Composition. 10th International Conference on Feature Interactions, 2009.

[9] Qi Zhao, Jiyu Huang, Gang Huang, Case Study: iMashup, A Service Composition Environment based on the Browser Middleware, Technical Report, 2009. http://sei.pku.edu.cn/~zhaoqi06/case_study_imashup.pdf

[10] iGoogle, www.google.com/ig

[11] WeatherBonk, www.weatherbonk.com