Towards End User Service Composition
Xuanzhe Liu, Gang Huang, Hong Mei
School of Electronics Engineering and Computer Science, Peking University, Beijing, China, 100871
liuxzh@sei.pku.edu.cn, huanggang@sei.pku.edu.cn, mieh@pku.edu.cn

Abstract
The popularity of Service Oriented Computing (SOC) brings a large number of distributed, well-encapsulated and reusable services all over internet, and makes it possible to create value-added services by means of service composition. Current composition styles are too professional to those end users when building their own applications. Actually, the end user would prefer rapidly discovering “best-of-breed” services to assemble as well as visually personalizing the presentation to enjoy rich experiences. In our work, we propose a composition approach to facilitate service composition in a fast and simple manner. We try to solve composition difficulties from the end user perspective: on demand service selection, automated QoS negotiation and visualized service composition. In our approach, similar candidate services are aggregated together as a unified resource, whose wide QoS spectrum can be easily manipulated by the end users to satisfy their requirements. Then they can personalize the services and, the composition occurs only at the presentation layer.

The main contributions of the approach are: (i) enabling the end users to personalize the composite application with more powerful presentation; (ii) supporting the end users to dynamically customize the service composition in terms of QoS; (iii) alleviating the end users from the time-consuming task of selecting service to compose.

1. Motivation
Service Oriented Computing has been a hot research topic in both academia and industry. Services are self-describing and platform-agnostic computational elements. The “loosely-coupled” feature guarantees flexibility. With the proliferation of service resources, the composition of value-added applications will bring more benefits to the large amount of consumers over internet. There are some famous composition technologies, such as BPEL and WS-CDL, which have been widely applied to assemble existing services.

Though these composition solutions are powerful, we realize some a fact: they are extremely suitable to enterprise users instead of regular end users. The reason is, these technologies involve relatively strong requirements overhead about user’s professional skills, e.g. XML and BPEL, development environment, e.g. design-time IDE tools, and runtime middleware (SCA server or BPEL engine). In other words, these technologies are too many professional to the end users to rapidly and simply grasp and apply.

However, in the internet environment, the number of end users is much larger than that of professional users. For example, in USA, there had been 90,000,000 end users, while only 3,000,000 professional ones [1]. It is unreasonable and impossible for the end users to grasp professional composition technologies. Therefore, the popularity of service composition will be heavily limited if no consideration from the end user perspective.

The end user composition involves some grant challenges. First of all, as there are so many providers offering services with similar functionalities, current search-based discovery will consequently return a number of candidates and cost end users much time and effort. Secondly, their desired QoS should be guaranteed, so there should be efficient and powerful QoS negotiation according to the requirements. Finally, the end users would rather personalize the composite service application, and usually prefers the visualized customization in “what-you-see-is-what-you-get” manner [8].

2. Approach Overview
We have made some efforts in terms of end user service composition and proposed an approach to solve the grand challenges above. The core principle is to facilitate the end users to freely, rapidly and simply build their own applications according to their preferences. To alleviate the time-consuming discovery, we aggregate similar services as a unified resource view to end users. The wide QoS spectrum such aggregation brings also enables the end users flexibly enough to control QoS negotiation. The localized composition environment makes personalization visually and rapidly.

Figure 1 illustrates the framework for the approach. We briefly introduce the approach from the following perspectives.

2.1 Service Aggregation
Service composition begins from the selecting proper component services. As discussed before, there must be a lot of providers offering a specific functionality, e.g., Google and Yahoo! have already developed map services for location. Such proliferation derives from the open, flexible services and the market competition. The proliferation of provides really increases the options to end users. However, current discovery mechanism, e.g., keyword searching and category browsing, will return a
number of candidates. Therefore, the selection may be very time-consuming.

From our point of view, it is unnecessary and unreasonable that end users face so many providers. In fact, these services are very similar or even the same functionality while different in QoS. Therefore, if the services can be represented in a simple way, and their QoS can also be negotiated by the users, the selection would be much easier.

In our work[1][2][4], we have designed a mechanism to aggregate the similar providers as a unified resource view, named service pool. The service pool construction is based on the similarity retrieval. We carefully studied current service description style, e.g., WSDL, and formalized the meta-information. Based on current search-based discovery, an agent retrieves the candidates’ similarity in terms of service name, operation and input/output. Besides, the agent also measures the domain similarity in a tree-based manner. After this filtering step, the precision of services aggregated can reach over 90% [2]. Then a virtual service description is generated to represent service pool as a single and unified resource.

Another effect service pool brings is the QoS spectrum of aggregated service. Obviously, service pool promises much wider QoS range for users to enjoy. We designed a dynamic QoS negotiation mechanism so that end users can on demand attain desired QoS by manipulating service pool.

### 2. Service Composition

Service pool is transparent to end users as a backend discovery/negotiation engine. It means the end users only face a single provider and adjust their QoS requirements. Therefore service pool becomes the component service to compose.

We have analyzed the obstacle for end users to use current professional composition technologies. Therefore, it is necessary to provide an end user oriented composition environment which makes them feel like using desktop applications. Such environment should provide features including:

- **High usability and personality:** end users would rather visually interact with a set of rich UI instead of reading textual interface specifications. The powerful user experience is the key to the high level of usability [7]. On the other hand, the end users prefer personalizing the composition according to their own preferences.
- **Low latency:** another defect of current composition is the response-time latency from the underlying heavy weight infrastructure. The end users would expect fast response-time from the applications. It implies the use of asynchronous mechanism to perform UI operations while invoking service in the backend. Additionally, the UI should update incrementally instead of frequent page refresh.
- **Consistent look and feel:** the composite service should use the same UI metaphors and conventions as desktop applications, to reduce the end users’ learning curve [8].

Considering the requirements, we have designed a runtime composition model and employed the concept of mashup [10], from web 2.0, to support a web-based composition engine. The model consists of three elements:

- **UI Component:** it is the entity which interacts with end users in the engine. Generally, the UI component is based on current popular presentation technologies, such as HTML widgets including button, text input and form. A UI component provides a set of operation to the end users and invokes the backend service by events. It should be noted that a UI component can be flexibly tied to a service.
- **Service Component:** it is the implementation of service, both local and remote. In our approach, a service component can be a service pool or any other services with clear interface specification.
- **Connector:** it is a set of configuration information between UI component and service component. Once the end users invoke a UI component operation, the engine will raise an event to interpret the configuration including service URL, namingspace and parameter/operation mapping.

In terms of executing composite service, the service composition engine takes charge of registering the UI component and managing the interactions between them. As the service components usually reside on the remote server, the engine employs the popular AJAX (Asynchronous JavaScript + XML). A large portion of UI component code is in client-side JavaScript, which makes
asynchronous network calls to service component via XML messages and then modifies client-side UI through HTML DOM. This mechanism improves user interactivity, since the JavaScript code is able to modify client-side HTML without reloading the entire page. In addition, since the XML messages may be sent asynchronously, users are not blocked from further interactions. Therefore, the composition task end users become easy and lively.

The service execution is incarnated by the service pool. As service pool only provides virtual operations and input/output, the messages raised in the client-side should be routed to the service satisfying end users’ QoS. Therefore necessary mappings are required. In the service pool side, we employ the reflection [5] and dynamic compiling to generate proxy for the service. It should be noted that the increasing concurrent end users and message size may cause performance overhead for service pool, we have also provide efficient mechanisms including for such problem [3]. As the UI component is a set of Javascript code at runtime, invocation results from service pool are processed by a data wrapper to restrict them in a standard format (we employ the JSON [11]).

So far the end users then can visually compose the services (to them is UI component only) by simple “drag-and-drop” action and the presentation is updated with very low latency. They can also make modifications of UI component to customize and personalize the presentation.

3. Demonstration

We have designed a set of tool for end users to fast create UI component and configure it with service component. Figure 2 shows the eclipse-based composition interface. The bottom part is the web-based mashup view. The end users can tie a UI component with a service pool by simply adding some configuration information. Once services are composed, the architecture view shows the whole architecture of the application. The end users can do modification at any moment and synchronization is just-in-time between the two layers.

4. Conclusion and Future Work

End users service composition is a big opportunity with the popularity of SOC as well as grant challenges. The main contributions of our work are: (i) enabling the end users to personalize the composite application with more powerful presentation; (ii) supporting the end users to dynamically customize the service composition in terms of QoS; (iii) alleviating the end users from the time-consuming task of selecting service to compose.

The future work includes: enforcing the fault-tolerance and security for end user composition and the optimizing the dynamic QoS negotiation.

5. References