An Investigation on Web as a Platform for Evolvable Service-Based Software

Teo Hui Ming
Department of Software Engineering
Faculty of Computer Science & Information System
Universiti Teknologi Malaysia, Malaysia
teohuiming@gmail.com

Dr. Wan Mohd Nasir Wan Kadir
Supervisor
Department of Software Engineering
Faculty of Computer Science & Information System
Universiti Teknologi Malaysia, Malaysia
w.wan-kadir@acm.org

Abstract

Recently, service-oriented or service-based software has emerged as a new software development model. Both researchers and commercial vendors are actively pushing and promoting Web services as a standard implementation solution for service-oriented software. This paper reviews the reason behind the needs of service-based software, and draws the reader’s attentions to a different ongoing research area – ubiquitous computing, in order to provide an alternative view for service-based software implementation. Finally, the paper suggests the importance an interface in service-based software, thus it will become a focus in the future research.

Keywords: service-based software, software evolution, service interface, Representational State Transfer, World Wide Web

2. Summary of Literature Review

In service-based software model, software is composed of a collection of small service units. For instance, when a user requires an improvement on one service unit, that component is disengaged, and replaced with another service found in the marketplace (Bennett et al., 2001a).

In contrast with software that shipped as one whole product, users are now able to assemble services from different suppliers based on own needs at a specific time, and only pay for what they have consumed. A change request can be fulfilled by quickly replacing the undesired one with a more suitable service. Thus enables ultra-rapid software evolution (Bennett et al. 2001b).

We summarize service-based software architecture consists of several basic characteristics:

- Software is composed of a number of service units available in an open market from various suppliers and geographical locations.
- Service units are assembled together based on demand, late-bound on execution time and allowed to be disengaged after execution.
- Desired service units can be discovered and several services can be composed into a larger service.

The consequent paragraphs will summarize a different ongoing research area, and later provide an intuitive view on its similarities compared to service-based software architecture, which may result in providing new ideas for service-based software implementation from a different perspective.
Ubiquitous or pervasive computing is a term for an emerging trend where: numerous, casually accessible, often invisible computing devices, being mobile or embedded in the physical environment, and connected to an increasingly ubiquitous network infrastructure composed of a wired core and wireless edges (Moran and Dourish, 2001).

Particularly, one of the ubiquitous computing research efforts (Drytkiewicz et al., 2004) (Kindberg et al., 2001) is to explore suitable communication infrastructures that allow a wide range of computing devices (from PDAs, cameras to printers and passive sensors like motion sensors) interact with each others and consume each others’ service (i.e. printing service).

There are several basic requirements in a ubiquitous system:
- A wide range of discrete devices equipped with different functionalities scattered around an environment without a central control point.
- Independently developed devices able to interact seamlessly and consume each others’ service based on users’ desire.
- Users able to discover nearby devices and utilize their services by cooperating devices to perform a desired task.

Based on the requirements, Kindberg et al. (2001) establish the “CoolTown” lab project to explore utilization of existing Web technologies in physical device communications. Barton et al. (2003) focus on service discovery and ad-hoc device interaction by using a tweaked HTTP content negotiation mechanism and a form-based language, specifically XForms. Continue on this research direction, Radusch et al. (2004) introduce an evolvable middleware system called pREST based on established standards (including HTTP, URI, XML) and REST architectural style (Fielding, 2000) as a platform for ubiquitous device communication.

Interesting to notice that, by comparing service-based software characteristics with ubiquitous system requirements, we can see several obvious similarities. First, both devices and service units can be viewed as discretely, separated units that perform certain functions. Second, both service units and devices are developed independently and can be bound together easily for interaction on runtime. Third, both require a discovery mechanism and a way to compose or cooperate several services to perform certain tasks.

The similarity comparison above gives us an opportunity to explore the service-based software implementation using similar approaches in ubiquitous computing, in particular, Web technologies.

Kindberg and Barton. (2000) utilize Web standards including HTTP and URL as a highly accessible and just enough middleware that capable to support the diversity of rapidly changing devices. On the other hand, Drytkiewicz et al. (2004) make use of the uniform interface in REST architectural style (Fielding, 2000), to allow device interactions being performed on a same resource abstraction layer. The resource abstraction allows every device to be conceptually identified as a resource, and therefore, actions can be performed upon any resources using a same set of methods, regardless actual implementations. Such powerful abstraction provides a consistent conceptual interface for interaction while allowing the implementation beneath it to evolve over time.

3. Critical Discussion

Based on Bennett et al. (2001a), service-based software model itself is a solution for software evolution as changes in a software can be performed as fast as possible to meet rapid-changing user’s needs, particularly by replacing an undesired service with a desired service.

Similar to distributed computing (Waldo et al., 1994) or a network-based system such as World Wide Web (Fielding, 2000), in service-based software, the implementation of a service is encapsulated behind an interface. In other words, an interface is the only thing that being exposed to users. The importance of an interface in service-based software is further emphasized under these two issues:
- Replacing services will become a common task in service-based software paradigm. How much effort or how much changes needed in user-side software every time bound to a new service?
- A service provider will need to perform changes on service-side software over
time. Can changes be made without altering the interface? If the interface must be altered, can user-side software handle the changes?

A distinct difference from most existing software evolution situations where developers have the rights to perform changes on the entire system software, user-side software in service-based software is usually developed independently and mostly a black box to service-side developers. The same situation happens where a service is also a black box to user-side developers. Users are only depending on the interface exposed by a service, which therefore again, emphasis the importance of an interface.

Back to the first issue above, in the best case, a good interface requires zero change in user-side software every time bound to a new service, thus provides a true run-time binding environment. In second issue, a good interface should be able to hide most changes that happened behind the interface, thus minimize the interface changes. If an interface must be changed, user-side software should have no difficulty to handle it with zero change, in other words, the interface itself is change-tolerable or evolvable.

A suitable interface for service-based software will be the one that match closest to the best case described above.

4. Further Work

The best case described above can be further expended to an evaluation framework that measure existing interface proposals or standards (Muehlen et al., 2004) from an evolution perspective. With the emphasis on evolvable interface, the research is expected to explore Web technologies and its uniform interface in order to establish a web-based service-based software implementation.

5. Conclusion

This paper presents a summarized review on service-based software as an upcoming trend in software evolution. Review on another ongoing research field - ubiquitous computing is presented as to provide a different view of possible service-based software implementation. Finally, the paper explains the importance of an interface in service-based software and further provides “the best case” as a reference on the characteristics of a good interface for service-based software.

References


