Service-Oriented Software

Servicing Mash-Ups

Jan Ong
Department of Software Engineering
University of Calgary
Calgary, Alberta
jeong@ucalgary.ca

Tim Tang
Department of Computer Science
University of Calgary
Calgary, Alberta
tcstang@gmail.com

Tim Driedger
Department of Software Engineering
University of Calgary
Calgary, Alberta
tdriedge@hotmail.com

***Abstract*—In this paper, we examine how service-oriented development has led to many Web 2.0 services through a component-based architecture. Through the piecing-together of such loosely-coupled services, more and more complex services can be created; this has eventually given birth to website mash-ups, websites in which developers use existing services to create a new graphical way of displaying data**

Keywords—Service, mash-up, coupling, module, composition

I. INTRODUCTION

Web-oriented software is perhaps one of the most important technologies out there today. In order to understand web-oriented software, we need to understand its predecessor. Web-oriented software is an extension of service-oriented software. Service-oriented software has been around for a while, but recent adaptations by web-services has brought the concept back to the forefront of software development, and as such, it will be important in future application creation.

In the early days of the World Wide Web, static web pages were prevalent. Static web pages simply provided information to the user. This information rarely needed to be updated, and would show the same information to every user, regardless of the platform on which it was viewed. This was a good solution in the past due to low bandwidths and simple browsers, but because of technological advances and the fact that we live in a world with a ubiquitous amount of different platforms, it is no longer practical.

The technological world is ever-changing. We live in an age with Facebook, Twitter, blogs, news feeds, wikis, video-sharing, and many other forms of social media—all of which are products of "Web 2.0", the second stage of development of the World Wide Web. Web 2.0 focuses on user-generated content, dynamic content, social networking, and applying service-oriented architecture to the World Wide Web. Consequently, exposing their functionality to the public to allow for integration between different applications.

II. SERVICE-ORIENTATED SOFTWARE

A. Services

To understand the importance of service-oriented software, we must understand what a *service* means. A service is "a specific instance of a component/components that has a public interface defined and described in XML and that other systems can discover and use by passing messages transported via existing Internet protocols" [1]. For this reason it is

important that services have a public interface. This allows for other services to find it, as well as interact and do something useful with it. These services can be combined as the user sees fit, and will allow for the creation of seemingly complex applications from as many simple components as the designer would like to add.

Another aspect of services is the fact that they are reusable. This means that a service can be used by multiple applications if needed.

B. A Brief History of Service-Oriented Architecture

Service-oriented architecture may have had its history before the term was even coined. In manufacturing factories, there became a need for components in one part of the factory to communicate with other components. For example, the quality inspection department in a factory may want to tell the factory material handlers in another part of the factory when a specific machine part fails the inspection test; thus, the handlers would be able to redirect the faulty part to another department in the factory to resolve that problem. One could even see how service-oriented architecture can even be analogous to other real-life scenarios.

Many other factors have contributed to the need for Service Oriented Architectures. The increase in application development, increase in internet-dependency, and advances in technology are just a few of the contributing factors, though it is by no means an exhaustive list.

It wasn't until the 1990s when the term, "service-oriented architecture" was first coined by the Gartner Group [7] to describe "a software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces…" [8]. This idea is actually very analogous to programming in general in which higher level programs are built around various library functions and system calls. However, it wasn't until the early 2000s when web-oriented services arose to the mainstream that service-oriented architecture became widely accepted and adopted as the web provided a means to glue different modules together.

Now that service-oriented architecture has become more widely-used, it became important to extrapolate the key concepts--such as loose-coupling, encapsulation, interoperability and language/platform independence, reuse--all of these concepts are important because they strive to achieve simplicity.

To this day, IT infrastructure continuously grows larger and more complex. With various possible operating systems, devices, applications, and protocols, it has become imperative that the concepts of service-oriented software be applied to all aspects of software engineering.

C. Putting the Services to Work

Service-orientation is based on two principles: modularization and composition [2]. Services are seen as different modules of a given system. Not only does this make services simpler, but this makes the system highly customizable, flexible, and less complex as additional modules will not have to know the inner-workings of previous modules. Combining modularization and composition yields the service-oriented system's behaviour. One way that this is implemented is through Web Services Description Language. WSDL provides a machine-readable description of how a service could be interfaced with. WSDLs are analogous to method signatures. WSDL is often used with Simple Object Access Protocol, a specification for how different services should exchange structured information. SOAP provides the messaging backbone for web services.

Services are important in that they have a public interface. This allows for other services to find it, as well as interact and use it. These services can be combined as the user sees fit, and will allow for complex applications from simple components.

D. Positives

There are a number of reasons to choose service-oriented development over other styles, such as distributed component architectures. Three such positives of service-oriented architectures will be discussed below. [3]

- **Vendor freedom**. In a service-oriented architecture, you are free to choose between multiple vendor products. You are theoretically free to choose between any number of services from any number of providers. This allows for greater system functionality without being locked into a certain supplier
- **Low coupling**. With other styles of development, high coupling can occur, which leads to fragile solutions. If one part of the system breaks, the entire system can die. This leads to painful maintenance and long debugging sessions. Service-oriented architecture is inherently modular, and therefore promotes low coupling. This allows for greater freedom for your system, since modules don't have to rely on other modules. A consequent of having low coupling is that the service-oriented module is

[generally] language-independent. That is, it doesn't need to be implemented in the same language because it knows nothing about the implementation of the other.

- **Lower complexity.** Other architectures can quickly become too complex to be easily changed. Polymorphism, inheritance, and references can increase the complexity of the system quickly, which can make maintenance and updating a nightmare. Service-oriented software doesn't rely on this, and therefore, yields lower complexity.
- **Autonomy.** Services are self-governing and stand alone. They exist to provide a service to arbitrary consumers. Services should be built independently (perhaps on other services) and as such, this contributes to the longevity and reusability of service-oriented components.

E. Negatives

As with any architectural style, the service-oriented architecture is not perfect. A distributed system adds a lot of complexity. Configuration of services becomes an issue.

- **Maintenance** - Version control of services can become a problem, as making sure many components are up to date can be tedious and time consuming. Any internal update to a service must not violate the intended output of the service and therefore should be completed cautiously.
- **Performance** - The complexity of distributed systems also entails that performance can be negatively impacted. The more spread out everything, the longer it will take to gather the necessary information. Another reason why performance is impacted is because of the arbitrary consumers of services. Because anyone can access these services, services must be coded to be interpretable by almost any consumer, i.e. the lowest common denominator. As such, the performance can be lower than it potentially can be if you assumed a high benchmark for the consumer.
- **More difficulties diagnosing problems** - Another problem that can come up stems from one of the positive attributes of service-oriented software. Loose coupling allows for the modularization of the system, which is key to service-oriented software. However, loose coupling means that there is a thinner safety blanket than if there was higher coupling. In a strongly coupled system, things such as misspellings, missing parameters, and mismatching data types can be caught by the compiler. This means that a strongly coupled system is actually easier to debug than a loosely coupled system, which does not have this advantage. [4] This is especially the case if the service is built upon other services. Determining the exact location of a bug can then become difficult.

F. Decline of Service-Oriented Architecture

Despite all the advantages of the service-oriented model, service-oriented architecture came to a decline during the late 2000's--coinciding the economic recession. It is argued that at this point, the hype of service-oriented architecture was dying down. As companies became more money-conscious, they had realized they poured too much money into the paradigm already and had yet to reap any of its benefits. As a result of the hype, companies had high expectations and were ultimately let down. Companies began to abandon the paradigm and look elsewhere for their business-to-business needs.

As the popularity of service-oriented architecture died down, different service paradigms came out of the woodwork to take its place. Despite the decline of service-oriented architecture, many of these paradigms extrapolated the properties of service-oriented architecture. Some of these services include (but are not limited to) cloud computing, software-as-a-service (SaaS), and mash-ups. The next section will speak more in detail about mash-ups.

III. MASH-UPS

Combining various web services and rearranging them in a particular order to produce a new service will yield what is called a "mash-up". "Mash-up" was first used as a music term to refer to the combination of several pieces of music (often from different musical styles), eventually producing a new (and hopefully interesting) interpretation. A software mash-up is similar, as it involves taking different components and rearranging them in a way to create a new application with functionality that is different than its previous parts. [5] For example, AuroraMap (http://www.auroramap.co.uk, fig. 1) is a website that uses Flickr, Facebook, Google Maps, and Twitter APIs to allow users to see Flickr, Facebook, and Twitter posts related to the Aurora Borealis. It can also show the

related posts within a certain time frame (from 24 hours ago to last year), and even shows the current aurora forecast.

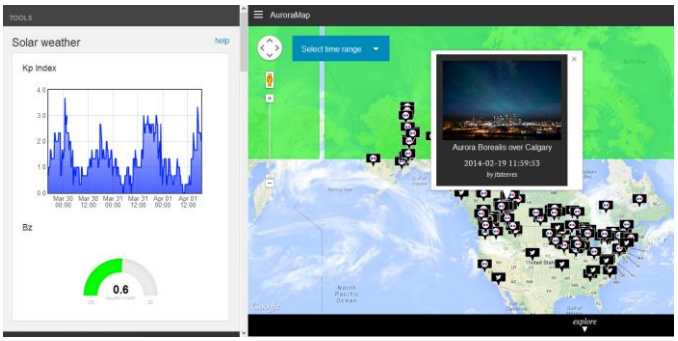

Fig. 1: AuroraMap showing a Flickr post about the Aurora Borealis.

A. *Mash-ups for the Present Day*

Mash-ups are ideal for the present day web development, which focuses on dynamic content rather than static information. Creating mash-ups from existing services to meet the needs of a business is often easier than going through the process of designing, coding, testing, and maintaining the code of new software.

Services can also be reused multiple times for different mash-ups. This is ideal if requirements for a system change. Reusing services from a previous mash-up into a new mash-up with slightly different logic is much easier to do than to write new software from scratch. This is less of an issue with Agile software development (fig. 2), but is a much larger problem in the Waterfall method (fig. 3). This is because the Agile method of software development can adjust to new requirements quicker due to its iterative nature. The Waterfall method does not focus on iteration, so if new requirements for the system come up, the Waterfall method has a more difficult time moving back to the requirements stage and redoing the system.

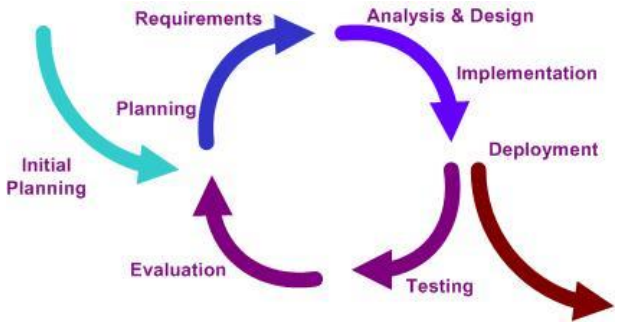

Fig. 2: Agile software development process

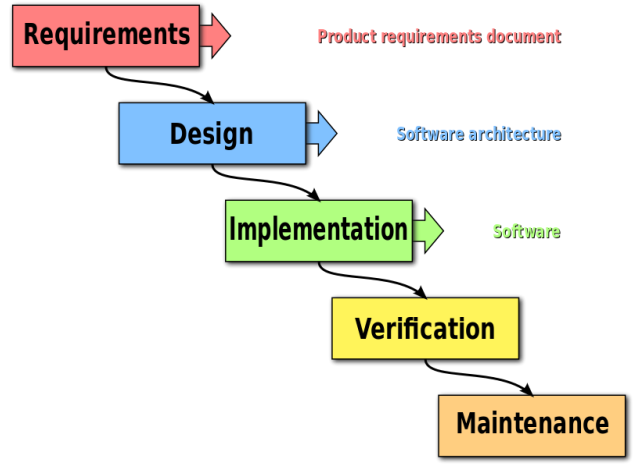

Fig. 3: Waterfall software development process (Peter Kemp / Paul Smith)

B. *Examples of Mash-ups*

- Automatic translation of a quarterly report from English to other languages (German, French, Italian, Spanish, etc.) [6].
- There is a mash-up called MustSeeGreece (fig. 3). It promotes and grows the value of inbound tourism in Greece by providing information about the multiple islands. To achieve this, MustSeeGreece uses content from four different sites (Flickr, Twitter, Google, and Wikipedia) and uses their APIs to obtain data to put on their website. [12]
- Another mash-up is called Shared Count (fig. 4). This mash-up receives a url from the user, and will return the number of shares, likes, comments, views, tweets, etc. for that url from multiple social media sites. Shared Count uses APIs from many social networking sites, including Facebook, Twitter, LinkedIn, and Pinterest. [13]

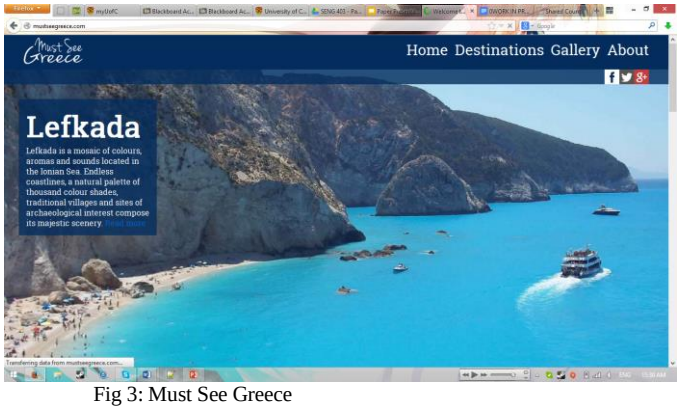

Fig 3: Must See Greece

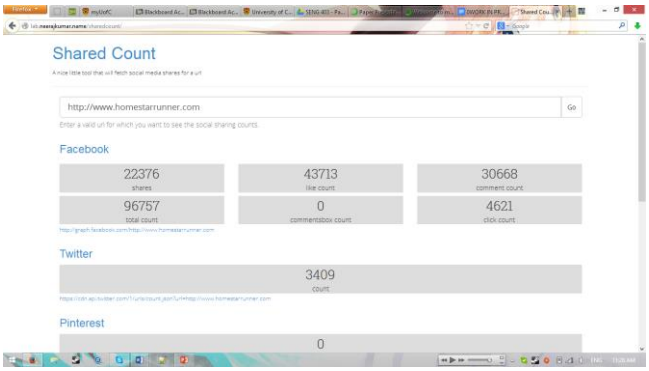

Fig. 4: Shared Count showing shares for http://www.homestarrunner.com

C. *Advantages of Mash-ups*

Mash-ups, when created from many different services, can be very useful tools. They have many advantages over their non-service-oriented cousins. The loosely coupled nature of integration between services. Along with the resilience to change via discarding out-dated services and replacing them with current up to date ones is one of the greatest strengths [15]. Data can be quickly assimilated from multiple sources/services. And this high level of customizability of mash-ups allow for an infinite number of specializations.[9] A side product of this is that there are no requirements to house or track data used by the additional services, this can cut down on server and storage costs. Also, when additional services are developed, they may be integrated into the system without the need to change a large amount of the existing architecture.

D. *Dis-advantages of Mash-ups*

Mash-ups depend on the services they access. If any of the services change or are removed, the mash-up may not be able to add the same level of value [9]. If the site requires that service to provide value, they will need to seek out another source or have one developed. And clearly, the more specialized the service, the more difficult it would be to find an alternative solution.

Another issue to be concerned with is the case where mash-ups access copyright or licensed material. Services providing data under which are licensed under the provision of the Creative Commons [14] will allow free distribution of their contained services [9]. Otherwise, there could be legal ramifications that will need to be considered.

E. *Mash-up Website Data Requirements*

One of the most common services required for a mash-up is data aggregation. Being able to sift through large amounts of data from various sources adds value to the developed product. There are four main considerations to setting up a data service; Organizing, Labeling, Navigating and Searching [10].

Organizing the data into a type of hierarchy will help to make sense of the data, it will provide benefit when navigating and searching the data, and it facilitates turning data into useable information. A well-organized data set will need to find the correct balance between breadth and depth—breadth referring to the number of categories each level of data is classified into and depth referring to the number of levels the data uses [11]. Another important note is the dichotomy between exclusivity and inclusivity, that is to say how large or small should the classifications be.

Labeling the data allows the user to quickly and easily find the data they require. When navigating through a site or application the user will be more likely to use the application if it is laid out in an intuitive manner. Having correct labels is one of the most important aspects. Labeling is also achieved through the Web 2.0 technique of “folksonomy”, which refers to the ability for users to add “tags” or labels to resources. In this way the data is labeled in a manner that best suits the users as common tags will rise to the top and obscure tags will enable users to hunt for specific parts.

A system that is easily navigated is one that will keep the user’s attention and interest. Both of which are particularly important for the Web 2.0 movement, since attracting and engaging users makes the site grow and become more useful to other users and so forth. Therefore, it is important for the user to be able to identify where they are in relation to the site as a whole, where they are able to go and ideally highlight options for where they would likely want to head.

Upon reaching a certain mass of data, some users will want to search for the data or resources that they require. While most users will be content to browse the site (again the importance of a solid navigation), few users will know what they are looking for and will want to be able to access it directly. In these cases an optimised search or query system is paramount. Also for consideration is the way the results are presented, whether it be alphabetically, chronologically or by relevance, if they will be grouped by category or content or popularity.

F. *Publishing and consumption of new mash-up services*

Developing and publishing mash-ups has been simpler than ever before. On the simplest level, one can release a web-oriented service online through transparent means, provide any documentation deemed

necessary to users, and encourage online communities to become users of their web-services.

Additionally, with the rise of popular mobile platforms such as Android and iOS, additional options have become available for independent developers to receive funding for publishing their applications. Android and Apple both make available a process to publish an application (whether a mash-up or not) so that developers can receive funding, and therefore be more motivated to create complex services.

The most important part when trying to garner users to use your service is to create something that is new, interesting, and useful. Developing a mash-up has many steps to it.

1. Figuring out your requirements for the system you are making, like most other systems, is the first step in the process.

2. Pick the components of your data, i.e., which services will fuel the service that you're creating. The data you want to get will probably determine where you get your APIs from. For example, if you are planning to use some social networking information such as likes or shares, using APIs such as the ones for Facebook or Twitter will probably be used. There are many APIs available online for various services. One place to find the APIs you may need is http://www.programmableweb.com. There are also many how-to guides to using some of the more popular APIs, such as Google Maps, Twitter, and YouTube on this site as well.

3. Piece the different component parts together using code. How you build your systems and handle your APIs will depend on how the services will interact with each other.

4. Release your final product once it is ready for consumption through the channel of your choice. http://www.programmableweb.com is one such example of a website where developers publish their mash-up services.

In addition to using multiple APIs and coding them into an application, there is an alternative that requires no coding at all. This alternative is known as Yahoo! Pipes (fig. 5). Yahoo! Pipes is an online application that allows the user to combine web feeds, web pages, and other services into a mash-up from the comfort of a graphical user interface. The application works by piping information from multiple sources into a single output. The information can be filtered by user input, so that only the information the user wishes to see is piped out.

For the example in Fig. 4, Flickr image posts and two web feeds (a sports news feed and a top stories news feed) are being combined and piped into one output. The subject of the sports feed and the Flickr posts are being determined by the user. The same can be said for the location of the Flickr posts.

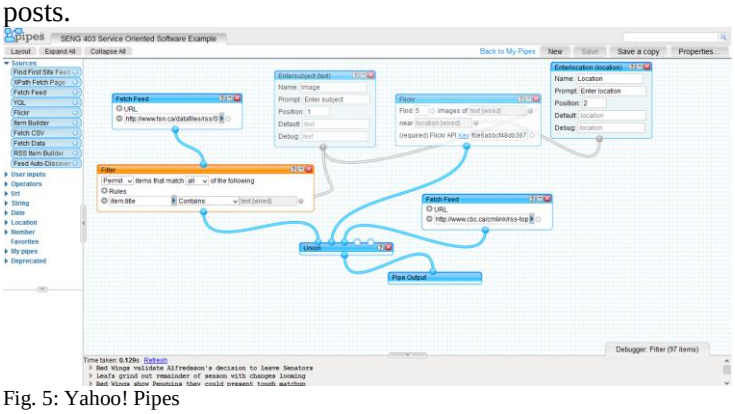

Fig. 5: Yahoo! Pipes

Using Yahoo! Pipes has the same general steps as coding together the different APIs. You would first figure out what you want the system to do, and lay out your requirements. Then you want to find out where your data is coming from. Yahoo! Pipes is limited in this sense, as it only uses services like Flickr and web feeds. More complex use of other APIs would be more suited to the first method. The trade-off to this is that creating a mash-up using Yahoo! Pipes is much easier, due to the fact that you don't have to code and you are creating the mash-up inside of a GUI. As well, Yahoo! Pipes can create a mash-up in a very short amount of time.

IV. Contrast Between Mash-ups and Traditional Service-Oriented Software

As stated before, service-oriented architecture was designed to be focused on the concept of being modular and composed of other services. What made service-oriented architecture very attractive was that it was supposed to be very flexible and efficient at connecting services to consumers. Because of the hype, during the mid-2000's, the SOA Manifesto was released and key-proponents of service-oriented architecture began to push for rigid adherence to "standards". This became paradoxical as it would conflict with the flexible property that was originally intended. As a result, service-oriented architecture became a very formal paradigm.

Mash-ups, however, follow no formal standardization or rules, even. The public is encouraged to create whatever they want with no emphasis on adhering to any specific rules, regulations, or even cosmetics. As a result, mash-ups focus on providing a functional service and often times can look very low-fidelity in comparison to a traditional finished software product.

V. Conclusion

In summary, services are programs that can be added to software to increase their functionality and value. The main emphasis of a server is not the inner-workings, but rather the product itself: the data. Services provide an alternative solution to storing and maintaining large amounts of data. While service-oriented architecture has dwindled throughout the years, other services have taken its place, such as mash-ups. Mash-ups—similar to traditional service-oriented architecture—have a public interface, which is an important factor that contributed to the increase of the popularity of mash-ups. Mash-ups combine the functionality of the smaller services by rearranging and displaying the data in an interesting way. The internet has made it very easy for anyone to create their own mash-ups; by using the public API of various websites, programmers are able to come up with unique and interesting mash-ups.

References

[1] N. Gold, A. Mohan, C. Knight, M. Munro, "Understanding service-oriented software", March/April 2004, pp. 72.

[2] Broy et al, "Service-oriented development", March 2006, pp. 1.

[3] G. Hohpe, "Developing software in a service-oriented world", January 2005, pp. 1-2

[4] G. Hohpe, "Developing software in a service-oriented world", January 2005, pp. 3-4

[5] A. Koschmider, V. Torres, V. Pelechano, "Elucidating the mashup hype: definition, challenges, methodical guide and tools for mashup", pp. 3-4

[6] N. Gold, A. Mohan, C. Knight, M. Munro, "Understanding service-oriented software", March/April 2004, pp. 73-74.

[7] Thomas Erl, Clemens Utschig-Utschig; Bertold Maier, Hajo Normann, Bernd Trops, "Next Generation SOA: A real-World Guide to Modern Service-Oriented Computing", May 1, 2013, chapter 4.1

[8] Gartner Group, "Technology Research", http://www.gartner.com/technology/home.jsp, January, 2014

[9] J. Governor, D. Hinchcliffe, D. Nickull, "Web 2.0 Architectures", May 2009, pp.157-158 .

[10] P. Morville, L. Rosenfeld, "Information Architecture for the World Wide Web, 3rd Edition", July 2008, pp 43.

[11] P. Morville, L. Rosenfeld, "Information Architecture for the World Wide Web, 3rd Edition", July 2008, pp 70.

[12] "Jovar", (2014, March 31), *MustSeeGreece* [Online]. Available: http://mustseegreece.com/about.php

[13] Neeraj Kumar, (2014, April 1), *Shared Count* [Online]. Available: http://lab.neerajkumar.name/sharedcount/

[14] See http://creativecommons.org

[15] "Service Oriented Architecture" - Chapter 1http://msdn.microsoft.com/en-us/library/bb833022.aspx