A Component-based Approach to Developing Thematic Mashups

Xin Liang[1] and Yan Liu[1,2] and Liming Zhu[1,2]
[1]School of Computer Science and Engineering, University of New South Wales, Australia
[2]National ICT Australia Ltd., NSW, Australia, 2015
xliang@cse.unsw.edu.au, jenny.liu@nicta.com.au, liming.zhu@nicta.com.au

Abstract

Mashup provides a way of forming new applications from existing Web content using APIs provided by different Web sites. Such a nature makes mashup a promising technology to deliver a Web-based Enterprise application with rich information of various themes, so called thematic mashup. However, the development of thematic mashup is ad-hoc and mostly from the scratch, which can be a barrier for Enterprise applications to leverage mashups. For an Enterprise application, the complexity of business logic and the requirements to adapt to a changing business environment demand a development method that helps to achieve reusability and maintainability. Component-based development method has proved practical to reduce the complexity of software development by reusing modular components. In this paper, we propose a reference architecture with key components identified for developing thematic mashups. We further demonstrate the usage of this reference architecture in a case study - developing a thematic mashup for a Web-based property valuation application. The contribution of this paper is applying a well-established software engineering discipline to emerging mashup applications, and providing insights of the techniques in developing mashups.

1 Introduction

Mashup is a Web application that uses APIs provided by different sites to reuse the content and form a new application[1]. A number service platform providers support the development of mashup applications such as Yahoo! User Library and Google Maps APIs, eBay APIs and Amazon eCommerce . A simple example of mashup is annotating search results on the Google Maps, such as the restaurants nearby. Mashups are emerging at a very fast speed. Mashup is considered as a technique that allows end-users to merge Web applications. An illustrating example is that users can drag and drop various gadgets onto their iGoogle page. Now mashup also becomes a technology for delivering Enterprise applications [10]. An Enterprise application may need to merge contents from a number of sources either publicly available or privately held, each having its own theme, such as weather, traffic rate, and historical selling prices. For example, a property valuation service company has historical data about the valuation results of an area, and these data are well informative for the valuer[2] to make an accurate decision on-site. Rendering such data on the location map can help the valuer quickly access to the information in a timely manner.

The overlay of a location map with thematic data results in a *thematic map*. A thematic map shows the spatial distribution of one or more specific data themes for standard geographic areas[3]. Compare to the normal annotated map (such as a location map annotated with nearby restaurants), while annotated maps show where a user's interests are in the place, thematic maps tell a story about that place. For example, coloring is one method employed by many thematic maps to show how intensive the data are covering the territory place. Therefore, a key part of developing a thematic map is generating an accurate base map without any thematic data. Once such a base map is available, different thematic maps can be constructed by laying out different data sets and parameters onto the base map. A mashup that involves merging different data sets and parameters with a base map to produce thematic map is referred as the *Thematic Mashup*. More specifically, a thematic mashup has three fundamental elements: (1) a base map, which remains the same for different mashups, (2) sets of thematic data, which determine what to be displayed on the thematic map, and (3) parameters, which customize the thematic map display such as the color or color density.

The current development of a thematic mashup is lacking of principled approaches to architecture and design - the development dealing with the three elements is ad-hoc

[1]http://en.wikipedia.org/wiki/Mashup

[2]a professional that assesses the price of a property by on-site inspection

[3]http://en.wikipedia.org/wiki/Thematicmap

and mostly from the scratch. A developer usually chooses a preferred development environment such as Yahoo Pipes, Google Mashup Editor, Microsoft Popfly or QedWiki, and manipulates the data. Many similar mashups use the same tools and APIs but do not share any pattern or any component in common. This implies that less reusability is gained [5] unlike other Web-based Enterprise applications development using .Net and Java EE. Besides, the development of a thematic mashup also has a number of challenges, including how to effectively deal with multiple sources of thematic information and how to render large amount of thematic information efficiently. Therefore the development of thematic mashups presents a typical software engineering problem and the research question how can software engineering methodology be applied to address such a problem. As mashups are emerging applications, there is little research dedicated to proposing a systematic development metrology, nor best practises are well generalized from the industrial community.

In this paper, we propose a reference architecture to develop thematic mashups using software components. In this architecture, the common operations are refactored into software components. These components encapsulate and separate operations of the map generation from the thematic information processing. Furthermore, the components are designed in such a way as reusable in different thematic mashups, taking into accounts the diversities of thematic data sets and parameters to customize the map. Thus this approach helps to enable the rapid development of mashups composed of reusable software components. We also present our experience using this approach to a thematic mashup as part of a Web-based property valuation application. The thematic mashup is a relatively new concept. To the best of our knowledge, its development issues have not been rigorously addressed in the field. Our contribution is proposing a component-based architecture to improve the reusability and maintainability of thematic mashup development.

The structure of this paper is as follows: section 2 introduces the related knowledge of Web 2.0 to mashups. Section 3 describes the reference architecture and presents the key components. Section 4 presents a case study that uses thematic mashup to display house prices for a Web-based property valuation application. Section 5 discusses the related work and the paper concludes at section 6.

2 Background

The development of mashups is underpinned by Web 2.0 technologies, such as REST (REpresentational State Transfer), AJAX (Asynchronous JavaScript and XML), RSS, and Wiki. According to O'Reilly [7]:

> *Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as a platform, and an attempt to understand the rules for success on that new platform.*

Many popular social and commercial Web sites are leveraging a number of Web 2.0 techniques, including personal Bloggers, Twitter, Facebook, Wikipedia, DoubleClick and Google AdSense. Web 2.0 can be considered as a trend rather than a particular technology that is enabling Internet as a platform to create new applications from existing ones. As a result, the programmability characteristic of Web 2.0 applications is accomplished by providing public APIs, such as Google Maps API and Flickr APIs. As more and more of these kind of APIs become available, developers start using multiple APIs to combine contents from different sources into integrated Web applications - Mashups. A good introduction of Web 2.0 can be found in the article [6]. In this section, we introduce the knowledge and techniques of Web 2.0 relevant to mashup.

2.1 Google Maps and Ajax

The access to Google Maps is via Google Maps APIs. Google Maps APIs enables a user to embed Google Maps in her own application with JavaScript (Google has recently announced a Flash version of Google Maps API too). Google Maps heavily rely on Ajax technology to fast and dynamically load Web content [10].

2.2 Google Local Search

Google Local Search is a Google service that allows users to search a neighborhood. For example, a search with keywords `pizza near University of New South Wales` will find all pizza shops in the nearby suburbs of University of New South Wales. Google Local Search can be embed in a Web application by using the Local Search Module of the Google Ajax Search API, which is a JavaScript API too.

2.3 Thematic Map and Heat Map

A thematic map is one type of heat maps, also known as the color heat map. The other type is the color density heat map. Conceptually, the main difference between these two types is whether the coloring on a map has a clear boundary. In a color density map, the value of data is represented by the density of color. The boundary of values are vague, hence it might be difficult to read exact value of the data colored. On the other hand, color density heat map is a good way to illustrate the data distribution and trend.

Technically speaking, thematic map is more difficult to implement than color density heat map, because the overlay

is strictly restrained by the geographic boundary. Therefore a map API to access to the shape is essential to produce a thematic map. To the best of our knowledge, there is no such a direct API. One alternative is the Google KML APIs[4], which is mainly used by Google Earth developers. Google KML partially supports Google Maps. Now the problem of generating thematic map overlay is the matter of generating KML files annotated with the color information from shape files. The shape files usually follow the standard format of Geographical Information System(GIS).

2.4 Client-side vs. Server-side Mashup

Mashups have two typical styles, client-side mashup and server side mashup, depending on where the mashup logic is executed. Server-side mashup is similar to traditional Web applications using server-side dynamic content generation technologies like Java servlets, CGI, PHP or ASP. The data from multiple sources are aggregated at the server side and the final results are rendered at the client's browser. In the client-side mashup, content mashed can be generated directly within the client's browser using client-side scripts (such as JavaScript or Applets). Mashups following the client-side style are often referred as Rich Internet Applications (RIAs). The benefit of client-side mashup includes the prompt response to user interactions because the data is pre-processed at the client's browser by leveraging Ajax techniques. For example, a page can be updated for portions of its content without having to refresh the entire page. Often a mashup uses a combination of both the server-side and the client-side style to achieve the data aggregation.

2.5 Summary

Web 2.0 promotes the mashups of Web applications with rich content, not only can be leveraged by individual end-user applications but also by Enterprise applications and services [3, 4, 11]. This emerging trends bring new challenges: the development of mashup applications is ad-hoc, largely subject to developers' skills and choice of programming tools and APIs. For Enterprise applications, the lack of coherent development methods and tools often results in less reusability, longer development time and more cost. In this paper, we introduce component-based architecture to thematic mashup development. The aim is to provide a solution for better reusability and maintainability, and improved productivity by reducing the software complexity.

3 The Method

Our method exploits a reference architecture that encapsulates common mashup operations into software components. In this architecture, the interactions between components are driven by thematic data aggregated to deliver map overlays. This architecture is deployed following the server-side mashup style as introduced in the previous section.

3.1 Reference Component Architecture

The reference component architecture is depicted in Figure 1. In this section, we discuss the details of the key components, namely `Adapter`, `Data Storage Component`, `Base Shape Generating Component`, and `Mashup Component`.

3.1.1 Adapter and Data Storage Component

The architecture stems from a number of resources to generate a mashup. According to Hoyer and Fischer [4], the resource of mashup for Enterprise applications can be content (e.g. from a Web site), data (e.g. from a data storage) or application functionality (e.g. from a Web service). The content from a resource contains thematic information, such as the traffic, population distribution, pollution level, nearby petrol stations and historical selling prices. The diversity and heterogeneity of the resource require transforming and aggregating the data from its source format to the target format that can be input to the `Mashup Component`(MC) generating map overlay files. Therefore an `Adapter` component is attached to each mashup resource. Each `Adapter` converts the data format of one resource into a common data format that can be processed by the MC. The `Adapter` component can be implemented in different ways, for example, it can leverage existing Web scraping tools to extract the content from a Web site. OpenKapow[5] is such a powerful tool that supports to interact with a Web site even a public API to access the site is not available. The detailed techniques required to implement the Adapter is out of the scope of this paper. The data transformed from `Adapters` are stored in the Data Storage (such as tables in databases or XML files in file systems) and accessed by the `Data Storage Component`(DSC). DSC is responsible for storing data source in a certain format so that they can be processed by MC. It is straightforward to implement DSC using a central database and stored procedures to query the data.

3.1.2 Base Shape Generating Component

The BSGC produces the skeleton Google Maps overlay - the shape files in KML format without any thematic information. It takes two inputs, the Google Maps and the territory shape files in the standard Geographical Information

[4] http://code.google.com/apis/kml/

[5] http://openkapow.com/

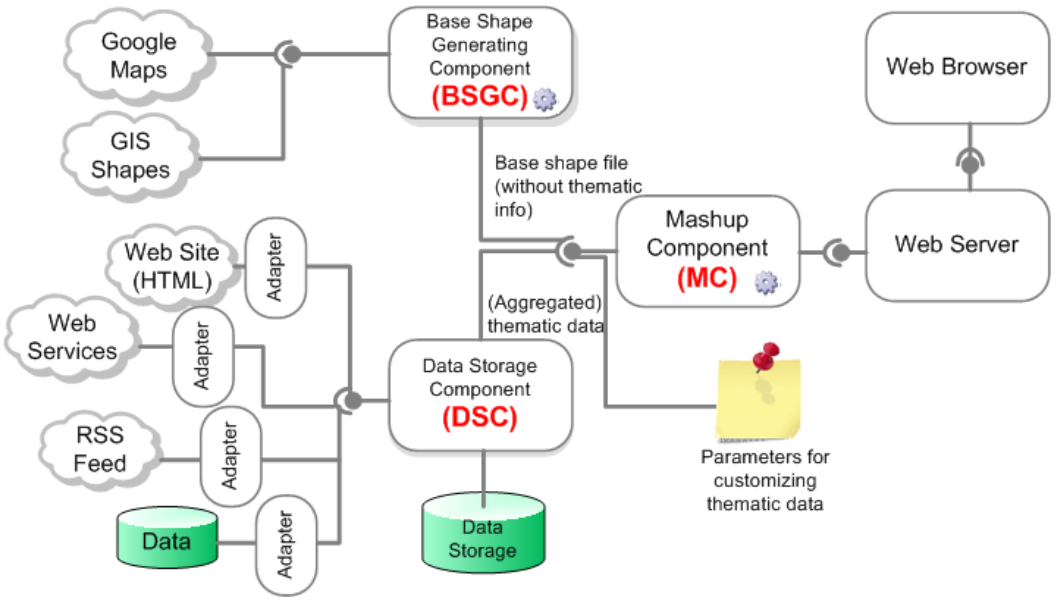

Figure 1. A Reference Component Architecture

System (GIS) format. Google Maps APIs provide a way to create customized overlay if a well-structured KML file is provided on a publicly accessible server. KML is a file format used to display geographic data in an Earth browser such as Google Earth, Google Maps, and Google Maps for mobile. KML uses a tag-based structure with nested elements and attributes and is based on the XML standard. The list below shows a partial KML file describing a shape.

```
<kml
  xmlns="http://www.opengis.net/kml/2.2">
 ...
<Placemark id="1">
  ...
  <LinearRing>
    <coordinates>
    151.131627264,33.8516306946267,0
    151.131732096,33.8510865171267,0
    151.133179072,33.8499404976268,0
    151.133180896,33.8499313771268,0
    151.133204064,33.8498165106268,0
    151.133276064,33.8494426071268,0
    151.13322,33.8494599971268,0
    151.133136128,33.8494760181268,0
    ...
```

The output of the BSGC is the base KML file used by the MC together with thematic data to draw Google Maps overlays. The main functionality of the BSGC involves three steps of operations:

First, **access to the source of the shape files**. The shape files of Australian territories can be accessed from Australian Bureau of Statistics (ABS) in a geographical spatial vector data format. This data format can be processed by most GIS software. A shape file specifies the coordinates of all suburb boundaries in Australia. This shape file contains the following fields for each suburb: (1) Name of the suburb, (2) State that the suburb belongs to, and (3) Coordinate, the list of longitude/latitude pair that represents the boundary of the suburb.

Second, **filtering unnecessary data**. The shape file usually contains more information than needed to produce the KML files. For example, ABS provides shape files to the suburb level. We might only need to obtain the base KML file for a particular state without details of the suburbs included. In this case, we can filter out unnecessary data about the names of suburbs to reduce the size of the final KML file. To do this, the shape file is loaded into a specific GIS software such as MapWindow to edit and filter the shape file. A GIS software can support some editing functions on a shape file. For example, MapWindow has built-in SQL-parsing functionality so that the filtering task can be accomplished by SQL script like `Select ... from ... Where State = 'New South Wales'`.

Finally, **converting shape files into KML files**. There are existing tools available to perform this task. For example, a third party MapWindow Plug-in called Shp2KML can be used to convert the shape files with selected fields to the KML files. A KML file might be generated off-line if its size is too large. The Google Maps overlay in the base KML file then gets loaded into the DSC for further use.

These steps are common across different mashup applications to produce the base shape files. Therefore each com-

ponent can be reused to build other thematic mashups with other thematic data or maps.

3.1.3 Mashup Component

The MC produces the final KML file with thematic data for the Web browser to render a Google Maps overlay. Eventually the thematic information is visualized by different colors. Different colors or color density can represent different levels of the thematic data such as the level of the traffic. Hence the MC first maps the thematic data into the colors. The MC accepts parameters to customize the color display on the final Google Maps overlay, which include the starting color, the ending color, color interval, color density, and opacity. The base KML file and the thematic data should share a common field as a key so that the MC can map the thematic data uniquely to one territory shape. For example, the thematic data of the traffic in suburbs within `City of Sydney` can be aggregated and then a specific color value field is inserted into the KML files to carry the thematic information. The output of the MC is the final KML file which resides on the Web Server. The sample code of the final KML file is illustrated below.

```
<name>CityofSydney</name>
 <placemark id="0">
  <name>Alexandria</name>
  ...
  <polygon>
   ...
   <outerBoundaries>
    <LinearRing>
      <coordinates>151.202134016,
      -33.9034390011261,0 151.20181404...
   ...
  </polygon>
  <style id="C1">
   <LineStyle>
    <color>CCAA3198</color>
    <width>1<width>
   </LineStyle>
   <PolyStyle>
    <color>CC000080</color>
    <fill>1</fill>
   </PolyStyle>
  </style>
```

3.2 Architecture Deployment

The deployment of the architecture follows the server-side mashup style, as shown in Figure 2. The key components are deployed in the Web server. Although client-side mashup can provide rich information applications through scripting languages, the complexity of producing thematic maps may be too demanding for the browser to handle. As discussed above, the KML files might be processed off-line and interact with a fairly large database to retrieve fields from base shape files and thematic data. Therefore, the computing capability of server-side style is more suitable to produce thematic maps. The interactions between servers and components are illustrated in Figure 3.

Figure 2. General Server-side Mashup

3.3 Optimization

A mashup is user-centric. The ultimate goal of a mashup is to provide users with enriched information or applications seamlessly aggregated from multiple resources. Therefore the user experience is a key criterion for the design and implementation of a mashup. As thematic mashup components communicate with data (shape files or thematic data) intensively as shown in Figure 3, optimizing data processing and transmission is essential to achieve efficiency and responsiveness for the benefit of the end users. An effective way is to reduce the size of the KML files. This can be achieved in the implementation of two components, BSGC and MC (annotated with the gear icon in Figure 1). In the BSGC, the optimization occurs when the base KML file is produced. As discussed in section 3.1.2, the source data to produce the base shape may contain more information than needed, and unnecessary data can be filtered to produce a smaller KML file. In the MC, the thematic data is inserted to the base KML files to embed thematic information as identified as color. The resulting XML files can also be optimized by improving the XML file structure. For example, it can reduce the size of the thematic KML file by defining all the color related fields separately and only referring to a specific color definition by its identity instead of repeating the color definition anywhere it is referred. In addition, the KML file in XML format can be compressed when it is transmitted over the network. A technical paper [8] provides detailed evaluation of a set of widely used XML compression tools.

4 Case Study: Embedding Mashup in Property Valuation Web Application

The reference architecture with key components for developing thematic mashups has been applied to a Web-based Property Valuation application (PVA). The application scenario is derived from our collaboration with the Lending Industry XML Initiative (LIXI)[6]. During a loan application process, a lending organization needs to determine

[6] http://www.lixi.org.au

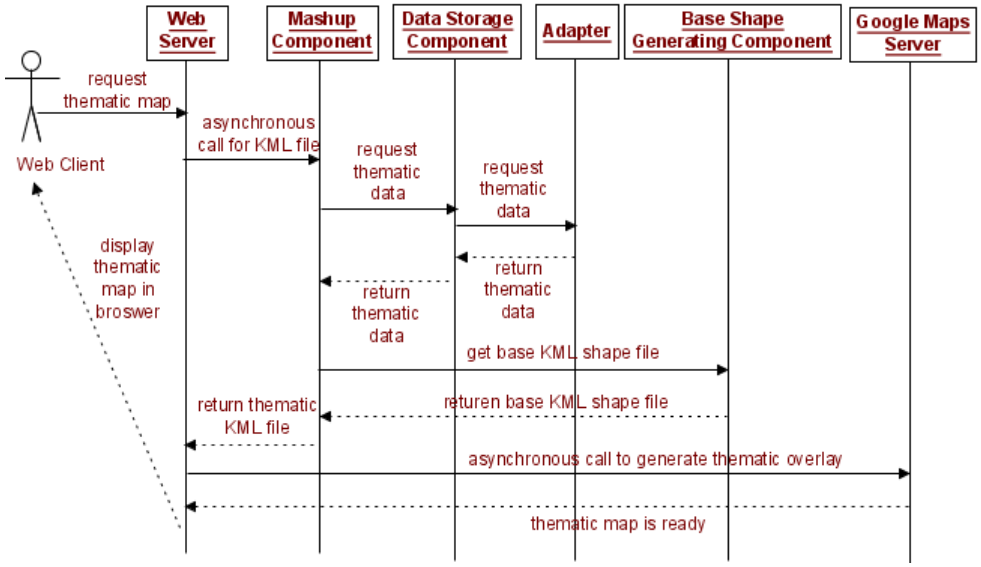

Figure 3. Server-side Mashup for Generating Thematic Maps

the market value of the loan-related property. This process is called "valuation" and is usually conducted by a valuation firm. The firm assigns a job to a valuer who conducts the on-site inspection of the property and fills in a paper-based valuation form.

To complete the valuation form, the valuer needs to do research on many aspects about the property, including the average property value, the airline noise level, the traffic conditions, the business development in the surrounding areas. The red triangle box highlights the fields that require the knowledge from the valuer to know what is happening around the area. Hence the thematic data can provide rich information and assist the valuer to make accurate assessment. However, without an integrated environment to access and visualize the thematic information in a simple and prompt manner, it is hardly possible for the valuer to leverage the raw thematic information given the pressure to finish the task on-time.

Our research is motivated by the goal to provide a Web-based PVA that enables valuers to access thematic information while filling in the form anywhere anytime as far as she is connected to the Internet using mobile devices or computers. Each field in the valuation form is presented in the Web form, and the thematic maps are displayed as part of the application when a suburb of interest is selected. In this section we describe our experience of specializing the reference architecture in the development of this Web-based property valuation application.

4.1 Web-based Valuation Form

This Web-based PVA has three main GUI panels as shown in Figure 4. The left panel contains the valuer's portfolio, such as the properties that the valuer has inspected or will inspect. The right panel is a tabbed panel that contains the valuation form. This panel also contains a tab to show historical prices trend of the areas that the property belongs to. The bottom panel is the map panel, which displays the thematic Google Maps.

This Web-based valuation form has several advantages over the traditional paper-based form. First, it is a 'green' solution that saves papers. All fields are editable by the valuer, and editing can be saved or revised. Second, the valuer can leverage the search function. The valuation form is categorized and searchable through the fields in the valuation form, namely, property name, value, comment, and category. The searched results are displayed on the same form. In this way, historical records of property valuation are easily accessible through a search. Moreover, the valuation form is interactive by embedding the thematic maps in the form. A number of different thematic information can be chosen to highlight the Google Maps. For example, when a valuer clicks on the traffic entry (highlighted in the red box in Figure 4), the thematic map panel will display a traffic thematic map.

4.2 Creation of Thematic Maps

The development of the thematic maps follows the reference architecture discussed in section 3. We use the generation of housing price thematic map of City of Sydney as

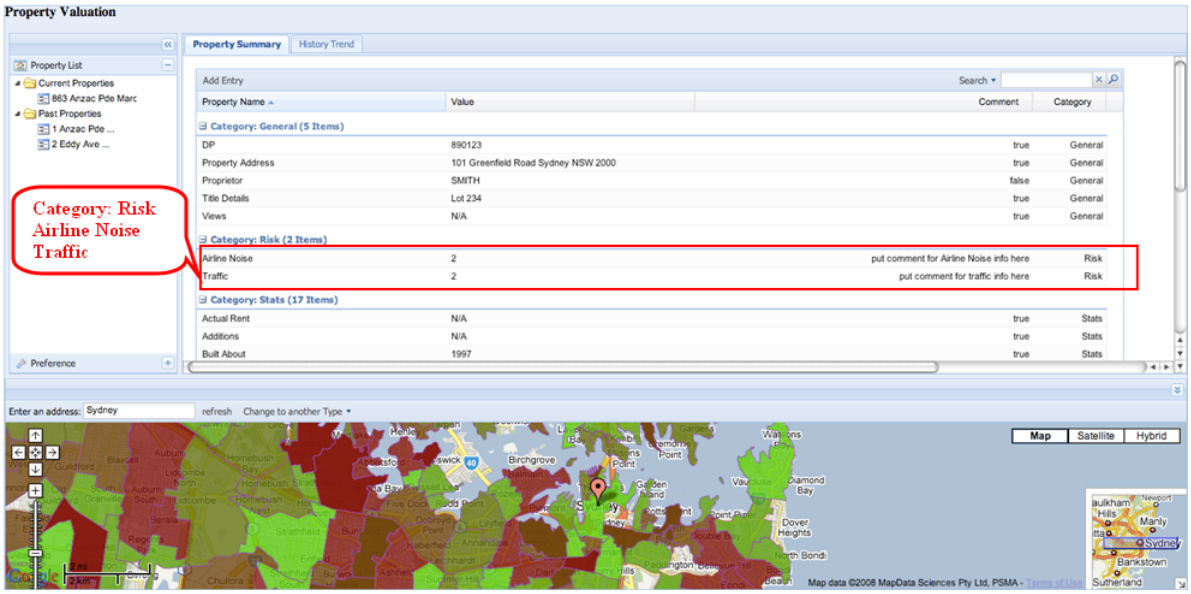

Figure 4. Property Valuation Application

an example to demonstrate the component-based development method. The reference architecture is now realized as shown in Figure 6.

First of all, we need a base KML file of City of Sydney. The shape files are from Australian Bureau of Statistic (ABS) website[7]. We use the BSGC to generate the base KML from the shape files downloaded. The BSGC implementation employs MapWindow for processing the shape files. MapWindow has built-in SQL-parsing functionality, so the filtering task can be accomplished by SQL script, such as `Select ... from ... Where suburb = 'The Rocks'`. The BSGC also includes operations to remove all the unnecessary information from a shape file. For example, any existing coloring information must be removed, otherwise it will confuse the color of thematic information later. Finally, a third party plugin to MapWindow is applied to convert the polished shape file into the base KML file. The screenshot in Figureřeffig:MapWindow illustrate the effects of the base KML file of City of Sydney. The next step is using the DSC to store the base KML files into a database (MySQL6) for further merging of thematic information.

One type of thematic information of interest is the property prices. Domain.com.au provides a suburb profile page for every suburb in Australia, in which average housing prices as well apartment prices are listed. All the information is in a Web HTML page. We developed an adapter in order to transform the pricing data into desired format to store it into the database. Domain.com.au provides both housing and apartment price information. We only need

[7]http://www.abs.gov.au/

Figure 6. Display of City of Sydney in base KML file

the price information of either housing or apartment price depending on the property type. Therefore the adapter requires a parameter to filter out unnecessary information. The adapter uses the DSC to store the resulting prices information in the data storage. Figure 7 shows the partial database table that is populated with housing prices of City of Sydney.

After both base KML file and the thematic information are ready and stored in the database. The next step is to

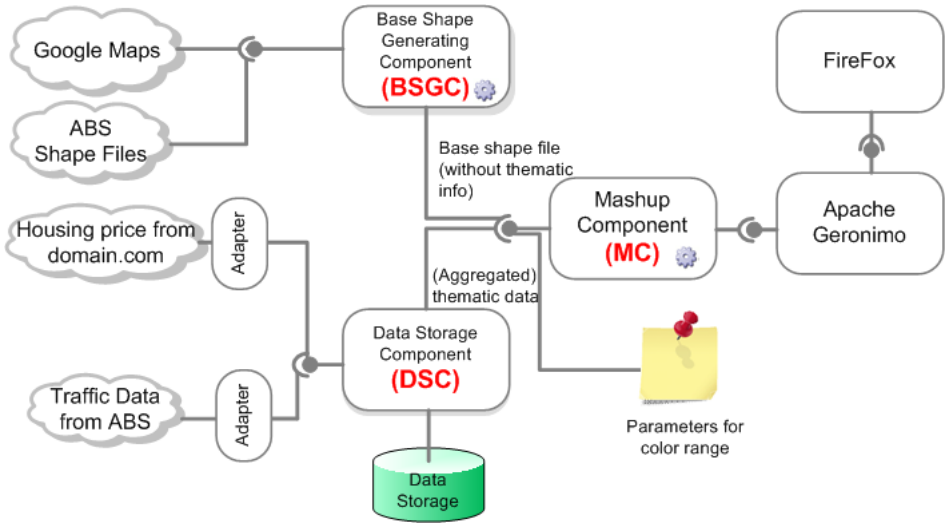

Figure 5. Concrete Component Architecture

suburb	value
Alexandria	655843
Beaconsfield	552000
Camperdown	650000
Centennial Park	820000
Chippendale	670000
Darlinghurst	940000
Darlington (Sydney)	702500
Dawes Point	4525000
Elizabeth Bay	1500000

Figure 7. Housing Prices of City of Sydney in Data Storage

use the MC to color the base KML file by mapping housing prices to a specific color level. In this case, the suburb is the unique identity to link the housing prices to a suburb in the base KML file. The mapping between the color and the housing prices are customized by input parameters of the MC. The color range parameter (or the interval) is used to calculate the color unit, as shown in the equation 1.

$$ColorUnit = \frac{Max\{housingprice\} - Min\{housingprice\}}{Interval} \quad (1)$$

The next step is attaching color to each element (suburb) in the base KML file. It is straightforward to find the corresponding color level using the ColorUnit, given the starting color is determined. The following code snippet shows a Java method that applies the colors for each suburb according to housing prices. Finally, the MC composes a final KML file with thematic information built-in. The screenshot in Figure 8 shows the final display of City of Sydney thematic map.

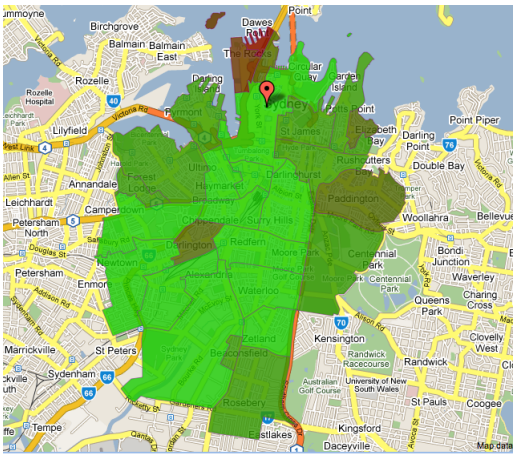

Figure 8. City of Sydney Thematic Map

```
private void applyColors
  (ArrayList<KMLElement> elemlist,
   int intervals){
   ...
   double cunit = (max-min)/intervals;
   for(KMLElement elem : elemlist) {
     Double value = elem.getValue();
     for(int i=1;i<intervals;i++){
       if(value <= (min+cunit*i) ) {
        elem.setColor("C"+i);
        i=intervals;
       }
     }
```

. . .

4.3 Optimization

The aim of optimization is to enhance the user experience when interacting with the Web-based PVA. Since the size of the KML file directly determines the waiting time that the end-user will experience, we focus on the optimization techniques that can reduce the size of KML file - the number of bytes that have to be transmitted over the network. In addition to the general techniques discussed in section 3 to improve the structure of the KML file, we further compressed the KML file using ZIP. Our experience demonstrated that zipped XML is normally only 1/4 of the original size. For example, the KML file containing all suburbs in New South Wales is 110.4MB whereas the compressed KMZ file is 30.9MB.

Another technical we employed is to reduce the number of decimal place on coordinates in the shape file. The more numbers of decimal place on coordinates the more accurate the shape is. A coordinate with 6 numbers of decimal place is with the error of accuracy less than a foot. The accuracy level is sufficient to produce the base KML file in the PVA. This technique also reduces a margin of the KML file size. Finally, a big KML file is split into smaller files to transmit over the network one by one.

All these techniques in combination can reduce the KML file size up to 80%. The communication is using Ajax asynchronous calls so that the end-user is not blocked for one operation and can continue the interaction with other functions of the Web page before the response returns. From our experiment, the user will experience some delay (no longer than loading a normal Google Maps page) the first time the thematic map is loaded, and later refreshing of the content or updating the thematic map is fairly prompt.

4.4 Discussion

The PVA implementation leverages the reference architecture, using the well defined components. By refactoring common operations into components, the overall architecture gains improved reusability, maintainability and reduced complexity which lead to improved productivity.

- Reusability. The components are designed in such a way that they can be reused in different thematic mashups. The architecture separates the processing of the thematic information and generating of the shape files. This separation enables the reuse of the shape generation components and mashup components for different thematic resources. For example, a new thematic map can be generated by adding a new adapter to transform the original data format and the other components in the reference architecture can be reused or with minimal modification.
- Maintainability. When the business logic changes which incurs the adding, updating or modifying the display of the thematic information, individual components can be modified and even replaced without having to affact other components as long as the component interface preserves.
- Improved productivity. The architecture is more maintainable with good modularization. The components are well protected through encapsulation. Interface design ensures that components can communicate through their interfaces. Since modification of the application can be accomplished by changing the only components involved, a reduction of software complexity has been achieved. Thus the architecture improves the overall productivity.

This PVA mashup is still an experiment of leveraging mashup in developing Web-based Enterprise application. Some advanced aspects in the normal Web applications are not considered in our current implementation such as security management. This remains our future work.

5 Related Work

A survey [3] categorizes existing mashup technologies and tools according to their roles in the lifecyle of mashups. These tools can be leveraged to implement the reference architecture presented in this paper. In this paper we focus on the component architecture and its main components, rather than the GUI tools to manipulate the mashup resources. The tools introduced in [3] such as Yahoo! Pipe can be consider to develop a GUI tool to manage adapters in this reference architecture.

A number of research papers have applied mashup as an alternative technique to semantic Web to compose Web services. Liu at al. [9] describe an architecture to build Web service composition using mashup techniques. Cetin et al. [2] proposes a service migration strategy in favor of mashup properties - composition of heterogenous resources. Zou et al. [11] introduces an ontology-based approach to present the obligation and liabilities of different roles involved in mashups. Some research work has also applied mashup techniques to other applications such as searching [1]. Our scope is at the mashup development level and does not consider the composition issues of multiple services in the Enterprise domain. The reference architecture abstracts the key components and operations involved. Our work is complimentary to the several methods discussed at the composition level. It remains our future work to further extend this reference architecture to develop Enterprise mashups composed by several services.

6 Conclusion and Future Work

In paper we propose a reference architecture to the development of thematic mashups. We identify key components in dealing with generating shapes, transforming thematic information from its original source and composing final shape files that is rendered in the Web browser. We also discuss the techniques of implementing, deploying and optimizing such a reference architecture. A case study of developing property valuation application with a thematic map embedded is presented to illustrate the practical usage of this architecture. Through our experience, we observe that component-based development help to achieve good encapsulation of the operations required by mashups, and thus key components can be reused to rapid compose new thematic information on a new map. The current reference architecture is still very focused on the scope of thematic mashup development, and does not consider advanced features required by Enterprise application, such as security management. We believe this simple reference architecture becomes a core to be further extended to address more complicated Enterprise mashup requirements.

7 Acknowledgement

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] D. Braga, S. Ceri, F. Daniel, and D. Martinenghi. Mashing up search services. *Internet Computing, IEEE*, 12(5):16–23, Sept.-Oct. 2008.

[2] S. Cetin, N. I. Altintas, H. Oguztuzun, A. H. Dogru, O. Tufekci, and S. Suloglu. A mashup-based strategy for migration to service-oriented computing. *Pervasive Services, IEEE International Conference on*, pages 169–172, July 2007.

[3] L. Clarkin and J. Holmes. Enterprise mashups. *The Architecture Journal*, 13, 2007.

[4] V. Hoyer and M. Fischer. Market overivew of enterprise mashup tools. In I. K. Athman Bouguettaya and T. Margaria, editors, *International Conference of Service Oriented Computing*, LNCS 5364, pages 708–721. Springer-Verlag Berlin Heidelberg, 2008.

[5] E. Maximilien, A. Ranabahu, and K. Gomadam. An online platform for web apis and service mashups. *Internet Computing, IEEE*, 12(5):32–43, Sept.-Oct. 2008.

[6] S. Murugesan. Understanding web 2.0. *IT Professional*, 9(4):34–41, 2007.

[7] T. O'Reilly. Web 2.0 compact definition: Trying again. *http://radar.oreilly.com/archives/2006/12/web-20-compact.html*, 2006.

[8] S. Sakr. An Experimental Investigation of XML Compression Tools. *ArXiv e-prints*, May 2008.

[9] W. S. Xuanzhe Liu, Yi Hui and H. Liang. Towards service composition based on mashup. In *Services, 2007 IEEE Congress on*, pages 332–339. IEEE Computer Society, 2007.

[10] J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, Sept.-Oct. 2008.

[11] J. Zou and C. Pavlovski. Towards accountable enterprise mashup services. *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pages 205–212, Oct. 2007.