Using Architecture Integration Patterns to Compose Enterprise Mashups

Yan Liu\textsuperscript{1,2}, Xin Liang\textsuperscript{2}, Lingzhi Xu\textsuperscript{1}, Mark Staples\textsuperscript{1,2}, Liming Zhu\textsuperscript{1,2}
\textsuperscript{1}NICTA, Australian Technology Park, NSW 2015, Australia
\textsuperscript{2}School of Computer Science and Engineering, University of New South Wales, Australia
firstname.lastname@nicta.com.au; xliang@cse.unsw.edu.au

Abstract

Enterprise mashups deal with corporate data and various sources of information to compose new value-added applications. The architecture design of enterprise mashups encompasses integration issues—it needs to integrate heterogeneous data and/or compose new situational applications from existing infrastructure. We envisage that architecture integration patterns can be applied not only as architecture solutions to mashup development, but also to help develop practical mashup techniques. In this paper, we combine several common architecture integration patterns, namely Pipes and Filters, Data Federation, and Model-View-Control to compose enterprise mashups. A number of techniques are also developed to customize these patterns for specific mashup needs. We illustrate our approach with a property valuation service derived from a real-world setting.

1 Introduction

The concept of enterprise mashups stems from the mashup development of Web 2.0. Mashup techniques retrieve content from several sources to create a new service or application [6, 11]. The resulting Web page is referred as a mashup of the existing content. Mashup enables individual users to quickly compose highly personalized and convenient Web applications with reusable widgets available online, such as iGoogle.

Mashup development has now been extended to the enterprise domain, where many enterprise systems have legacy applications and resources. Reusing them to improve business services is a challenging issue. At the same time, Web 2.0 technologies enable enterprise applications to access a broad range of information available on the Internet. Leveraging this kind of information brings opportunities such as developing new situational applications to improve business productivity or repurposing existing applications with value-added features to achieve efficiency. For example, a property valuation service (PVS) company has an internal service to retrieve historical data about the valuation results of individual areas. Such data informs the housing price valuation process when combined with other public information such as traffic and crime statistics. A mashup of these sets of information on a location map can help a valuer\textsuperscript{1} to make an accurate decision on-site. The question is how to design an architecture that supports the mashup of historical valuation results, traffic and other information and present the combined information in a clear map overlay. The architecture needs to deal with integration issues as organizing the flows of data and/or transforming data into consistent formats for mashup manipulation.

The architecture design of enterprise mashup has more challenges than developing throw-away end-user widgets. Enterprise applications are facing increasing demands for adaptation to changing business environments. This means the architecture of enterprise mashups should consist of reusable architectural components so that data sources or new logic for mashups can be easily extended.

Among software architecture principles, architecture patterns (or styles) offer the promise to help architects to identify combinations of architecture and solution building blocks that have been proven to deliver effective solutions in the past, which may provide the basis for effective solutions in the future. Reusing established architecture patterns for mashups and building solutions can provide deep insights not only to resolve the architecture design itself, but also give the knowledge of the mashup technologies.

In this paper, we propose an architectural approach that encompasses architecture integration patterns and key components of developing enterprise mashups. The core of this approach consists of the Pipeline and Filter pattern, the Data Federation pattern and the Model-View-Control pattern. Key components to realize these patterns can be customized and reused to meet specific mashup needs. We develop techniques to realize these patterns for a map with overlays, but the approach is generic to other types of mashup presentation such as markers and annotation on a

\textsuperscript{1}a professional that assesses the price of a property by on-site inspection
map. We demonstrate our architectural approach and techniques using a scenario derived from a real-world property valuation service (PVS) used by the Australian lending industry.

Our work demonstrates how existing software architecture patterns can be applied to help the design and the development of emerging enterprise mashups. Given that mashup architecture has not been very well studied in the enterprise area, our contribution is threefold:

- Analyze the issues of designing enterprise mashup architecture from our experience;
- Identify three architecture integration patterns and the key components for mashup architecture; and
- Develop the essential techniques to realize these patterns.

In this paper, we first describe the business scenario of a property valuation service in Section 2. We explain the simplified business process and identify the needs for developing enterprise mashups. Three architecture patterns are discussed in Section 3 to address the mashup design and development. Key components of each pattern are presented in the context of mashup architecture. The practical usage of this architecture using a Web-based PVS is demonstrated in Section 4. In the conclusion of this paper, we summarize the related work and lessons learnt.

2 Motivating Business Scenario

In this section, we introduce a business scenario for enterprise mashups - Property Valuation Service (PVS), which is derived from our collaboration project with an Australian lending organization [12].

PVS is one of the key services in the chain of steps in the mortgage lending procedure. It provides a lender or a broker with an accurate valuation of the real estate in the loan application. PVS is usually provided by an independent valuation firm. When the valuation firm receives a request from lenders or brokers, the valuation process gets started and has a number of tasks to be completed. The business goal of the whole valuation process is to reply to customers with accurate property valuation and with restricted response delays.

The simplified valuation process is shown in Figure 1. Three roles cooperate to accomplish this process, namely administrator, valuer and manager. The administrator contacts the owner of a property, schedules the inspection time and assigns a valuer to do the valuation on site. Once the valuer accepts the job, the valuer conducts on-site inspection, completes the valuation form and sends it back to the valuation firm office. A manager receives the valuation form from the valuer and approves it before the valuation result is sent to the customer.

Although the whole business process is well-defined and the responsibility of each role is clear, the actual interactions between different roles are very subtle, and thus achieving the business goal of the guaranteed responsiveness is non-trivial. The fact is that the outcome depends on the availability of the selected valuer and also the working experience of the valuer in the target suburb. Assigning the job to a wrong valuer (either unavailable or being not familiar with the market in the area) is mostly likely to incur more delays - switching valuers with back and forth communication through phone calls, emails or faxes. It could also result in inaccurate valuation due to the lack of expertise for special purpose valuation such as the commercial property valuation. To avoid these risks, the administrator needs to take into account the valuers record in terms of the valuation quality and quantity. All these information are available in the cooperate system, however to access them the administrators need to get central IT support to retrieve such information.

The lack of flexible and customizable software tools is one major barrier to improve the business process of PVS. One system currently used in the valuation industry is called Valuation Management System (VMS). Currently VMS only satisfies the basic business requirement and does not provide customizable functions to the internal business process of individual valuation firms. For example, the function is not enabled to allow administrators to quickly access to valuers’ record for a particular suburbs.
Enterprise mashups with customizable information overlay provide a feasible solution to address this problem. The mashup architecture is motivated by presenting information to facilitate the tasks in the PVS process. It consists of data or information from three sources: the public territory information, the cooperate records about valuers, and the public information on housing prices. The architecture targets at a mashup with overlays for the administrators to easily view the available valuers for an area. From the valuer’s point of view, the mashup provides a Web-based valuation form so that the valuer can fill in the form and access other information related to the property value, such as the traffic status.

3 Integration Patterns for Mashups

The mashup architecture is driven by the need to combine data or functions from more than one source, depicted in Figure 2. The data and the output from any function are of myriad forms and formats. They can be stored at cooperate information systems, available from public domains using RSS (Really Simple Syndication) feeds or Web services through the Web. They can be accessed by various protocols, such as DB drivers, RSS (Really Simple Syndication), REST (REpresentational State Transfer), HTTP (HyperText Transfer Protocol) or SOAP (Simple Object Access Protocol). Mashup enablers use various tools such as CodeGlide Fusion² to make internal or external data resource available to the mashup builders. Mashup builders define the workflow to connect data and create composite applications. They provide the user interfaces for developers or widgets for end users.

The architectural nature of mashup include three core aspects - the dataflow in procedure, the composition of heterogeneous data and interfaces, and different views on the same sets of data. Such an architecture motivates reusing architecture integration patterns to build mashup design solutions [7].

We develop an architectural solution for composing mashups that initially focuses on using three patterns, namely Pipes and Filters pattern, Data Federation pattern and Model-View-Controller pattern. These patterns stem from the best practices of enterprise systems integration [4] and are widely adopted in practice. They are suitable for the mashup architecture which has typical demands on data and function integration.

Figure 3 depicts the conceptual level architecture that integrates the three patterns. Since these patterns are technology independent, the abstract pattern elements should be mapped to the key mashup components. Moreover, when multiple patterns are combined, the structure of the mashup components should be carefully designed at the intersecting points of patterns to reduce the risk of cross-cutting concerns [5].

3.1 Pipes and Filters

The dataflow in the conceptual architecture (see Figure 2) has a procedural nature. Many mashups leverage the Pipes and Filters integration pattern to create the logic for data combination. The implementation of Pipes and Filters pattern consists of a series of interconnected components. Each component performs a specific function such as filtering unnecessary data, joining records, or forking streams. The connection between components denotes that the output from one component is the input to the other component.

In the context of mashup architecture in Figure 3, one composite component produces data inputs to the mashup component, which involves the Data Federation pattern to be discussed in the next section. A number of filter components are attached to each data source and produce the raw data required by mashup. For example, domain.com.au provides comprehensive housing prices for cities and suburbs in Australia, however the mashup may only requires data of

²http://fusion.codeglide.com
City of Sydney, which only includes several suburbs. The filter is to extract data just for those suburbs and aggregate them for the area of City of Sydney. As a result, the filter component performs the role of the mashup enabler. Software tools such as CodeClide Fusion can be applied as the filter.

The filter can be further decomposed into several Pipes and Filters as shown in Figure 4. It contains a data loader to just load in the data, connects to a couple of components to trim unnecessary data for various kinds, and finally converts the data to the required format by the mashup component. For example, Australian Bureau of Statistics (ABS)\(^3\) provides comprehensive information of the geographic data of Australian territories, which can be used to produce the area boundary on Google Maps. At the moment only a zip file is available for downloading, and the format is in the GIS format. However Google Maps only render map data in the KML (Keyhole Markup Language) format, which is used to display geographic data in an Earth browser such as Google Earth, Google Maps, and Google Maps for mobile. Hence the format convertor is attached at the end of the filter component.

![Figure 3. Architecture Integration Patterns for Mashup](image)

**Figure 4. Pipes and Filters Pattern**

Similar to the filter component, the Data Gen component consists of several Pipes and Filters, see Figure 5. It transforms the data into a different scale or measure for the presentation purpose. Statistical data such as the median property prices of suburbs are normally presented by different color with density. The mapping function, for example, maps the median property price to color units in order to overlay the price information on the map. This is different from the format conversion, which concerns with the structure of the data. Other methods of presenting map overlays include symbols marked on the map, such as 1D (height), 2D (area) and 3D (volume). Given the visual effects of different dimensions, it is one degree harder for the viewer to assess the relative size of 3-dimensional symbols compared to 2-dimensional ones, which again are harder to compare that 1-dimensional symbols [9]. Thus the component of Scaling Function dealing with perceptual scaling might be necessary along the pipes.

![Figure 5. Mashup Data Generation](image)

The advantage of using the Pipes and Filters pattern is the simplicity in modeling data flows. Each pipe or filter in the flow is explicitly defined and their integration is driven by the data stream - data sources are filtered, processed and combined to produce new meshed-up information. In addition, it is easier to maintain the mashups built using this pattern since the mashups execute exactly as they are designed following the flow paths at runtime.

### 3.2 Data Federation

The Data Federation pattern aims to efficiently federating both structured and unstructured data from multiple disparate sources. This pattern supports data operations

\(^3\)http://www.abs.gov.au/
against a transient or virtual view. The source data remains under the control of the source systems and is pulled on demand for federated access. Data federation concepts and techniques were extensively researched in the last two decades. Now this pattern has found yet another use in mashup architectures. The need of aggregating, correlating and correcting relevant data or function also occurs when implementing an enterprise mashup application.

This pattern has a declarative nature, compared to the procedural nature of the Pipes and Filters pattern. The Data Federation pattern may consist of several lines of Pipes and Filters, each providing the mashup component interface to access the data. At design time, views of the data sources are defined and the relationships between such views are also defined as operations including joins, unions, projections, selections and aggregations. Programs (such as stored procedures) or software tools (such as CodeGlide Fusion and Denodo\(^4\)) can be applied to define operations and produce views. A sample screenshot of CodeGlide Fusion suite is shown in Figure 6.

![Figure 6. Example tool for Data Federation](image)

In the context of mashup, one variation introduced to the Data Federation pattern is data redundancy. The original pattern leaves the data in place without creating data redundancy. Tools such as CodeGlide Fusion actually uses an embedded database and keeps some copies of data while operating on data.

The relation between Data Federation pattern and the Pipes and Filters pattern is illustrated in Figure 3, where the Data Federation pattern can be realized by several parallels of Pipes and Filters.

### 3.3 Model-View-Control Pattern

Mashups follow Model-View-Control (MVC) pattern by nature as they render data to present views according to user inputs. In the MVC pattern, the model has access to data from a storage or a service provider site, and organizes the data in a structure to be used by the controller. The controller accepts the browser input, figures out what to query the model for, and decides which view to use and what data to send to the view. The view accepts input from the controller and generates outputs in HTML, XML or JavaScripts to the users browser.

In the mashup architecture of Figure 3, the model can be implemented using the Data Federation pattern. The model is connected to the mashup component that acts as the controller, by means of the Pipes and Filters pattern. The controller also includes the Web server that render HTML or XML code to produce a view to the user’s browser.

There are two ways to construct mashup architecture following MVC, namely the server-side mashup and the client-side mashup. In the deployment of the client-side mashup shown in Figure 7, the logic of composing map overlays is performed by Javascripts at the client browser. This architecture features Google Mashup Editor, XML database and Javascript library such as jQuery. This simple deployment has its drawback due to the browser’s limits to handle complex logic and large amount of data.

![Figure 7. General Client-side Mashup](image)

Alternatively, in the server-side mashup (shown in Figure 8), the integration of data contents occur at the server side. The Web server accepts Javascript invocations (through Ajax) from the Web browser and uses Servlets to dispatch this request to an appropriate handler class, which

\(^4\)http://www.denodo.com
implements the mashup component in Figure 3. Logic of processing and integrating data or functions are done by the handler class. The response is then sent back to the controller Servlet, which in turn sends response back to the client browser. The browser’s JavaScript functions to update the web page with information obtained from the response. The computing capability of the server-side mashup is more scalable to be combined with the Data Federation pattern and the Pipes and Filters pattern.

![Figure 8. General Server-side Mashup](image)

4 Example Application: Web-based Property Valuation

The architecture integration patterns for composing mashups are applied to the Property Valuation Service (PVS) process discussed in Section 2. The application scenario (depicted in Figure 1) is derived from real world scenario provided by the Australian lending industry [12]. In this section we describe our experience of developing mashup for a Web-based PVS that utilizes the architecture integration patterns presented in this paper.

4.1 Web-based Valuation Form

This Web-based PVS has three main GUI panels as shown in Figure 9. The left panel contains the property list 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 area that the property belongs to. The bottom panel is the map panel, which displays overlays on Google Maps such as housing prices and traffic status. The map highlights the areas that a valuer prefers to work on together with the performance ranking and the booking information. The coloring of the map is based on the median price level of the properties in the area.

This Web-based evaluation form is consistent with the paper-based valuation form. The form can be converted to the PDF format and sent to the valuation firm on the fly, which saves paper and the cost of postage. This Web-based valuation form has several other advantages. First, it saves time spent by the administrator on spotting the best available valuer in the target area. The administrator only needs to input the suburbs of the target areas and the map will be automatically produced with a marker to list available valuers. Valuers are displayed only in their working areas, together with their ranks and bookings information. Second, the valuer can leverage the historical prices of the valuation property easily. Moreover, different types of information can be chosen to highlight the Google Maps. For example, when a valuer clicks on the traffic entry in the valuation form, the map panel displays a map with traffic overlay.

4.2 Mashup Development

The core of this Web-based PVS is the embedded mashup display. The mashup development follows the architecture patterns discussed in Section 3. We use the scenario of the valuer distribution in the area of City of Sydney (including several suburbs) as an example to demonstrate this architecture approach.

The key components to produce the mashup is illustrated in Figure 10. There are four sources of data, the territory geographic data for producing shape boundary on Google Maps such as housing prices and traffic status. The map highlights the areas that a valuer prefers to work on together with the performance ranking and the booking information. The coloring of the map is based on the median price level of the properties in the area.

The overall mashup data flow follows the Pipes and Filters pattern. The interactions amongst the client browser, the
Web server and the mashup component employ the MVC pattern and the server-side deployment.

4.2.1 Preparing Data for Mashup

First of all, we need a base KML file of City of Sydney without any overlay attached to display the territory boundary on Google Maps. Google Maps APIs provide a way to create customized overlay if a well-structured KML file is provided on a publicly accessible server. 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
<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
      </coordinates>
    </LinearRing>
  </Placemark>
</kml>
```

The shape files of Australian territories are downloadable from Australian Bureau of Statistics (ABS) in a geographical spatial vector data format. The shape file specifies the coordinates of all suburb boundaries in Australia, including the 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. The coordinate data need to be aggregated for the relevant suburbs and the rest data can be filtered out. To do this, the shape file is loaded into a Shape Filter - we use a COTS GIS software MapWindow that supports 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 similar to Select ... from ... Where State = 'New South Wales'.

Then the data should be converted to KML coordinates. The GIS2KML Converter leverages a third party MapWindow Plug-in called Shp2KML to convert the shape files to the KML files. The effect of the resulting KML file is shown in Figure 11. The next step is to input the base KML file to the mashup component to merge overlays with map coordinates.

One pipeline deals with extracting valuer working area distribution at the bottom of Figure 10. All completed valuation in history are stored in a central cooperate database and the existing VMS is connected to this database to access such information. One filter component contains SQL stored procedures to extract the interested data. Another component Criteria Filter is for valuer’s performance ranking. The ranking is originated from the PVS manager with regard to individual valuation quality and quantity in the past. Such statistical result can be directly generated from independent human resource system in a XML format. Now a XML Parser is used as the filter. The filter reads the XML file, processes the data and then matches up the records of historical valuation results to find out valuers working in a certain area with the satisfactory record.

4.2.2 Producing Map Overlays

Eventually the four pipelines converge to the mashup component to produce a single KML file with map overlays that the Google Maps server can render. The overlay is visualized by colors. Different colors or color density can represent scales of the data. Hence the component of Color Mapping Function maps the data from one measure (price) to another (color). The mapping between the color and the housing prices is customized by the scale parameter of interval, shown in Equation 1.

\[
\text{ColorUnit} = \frac{\text{Max}\{\text{housingprice}\} - \text{Min}\{\text{housingprice}\}}{\text{Interval}}
\]

The next step is attaching the color unit to each element (suburb) in the base KML file. It is straightforward to find the corresponding color level using the color unit, given the starting color is determined. Finally, the mashup component composes a final KML file with map overlays built-in. The sample code of the final KML file is illustrated below. This KML is rendered by the Google Maps server and produces the final display shown in Figure 9 and 12.
4.2.3 Producing Different Views

Three pipelines at the top of Figure 10 deliver different views of the mashup following the MVC pattern. As shown in Figure 12, the mashup enables users to select the interested menu item and the mashup switches between views. Since the communication is using Ajax asynchronous calls (see Figure 13 for details), 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 map and overlays are loaded, and later refreshing of the content or switching the map between views is fairly prompt.

5 Related Work

Mashup Development Tools. A survey [3] categorizes existing mashup technologies and tools according to their roles in the mashup lifecycle. In this paper, we broadly categorize these tools into two groups, mashup builders and mashup enablers. Mashup builders are tools that produce the user interface of a mashup such as IBM QEDWiki and Yahoo! Pipes. These tools enable non-developers to compose mashups by connecting widgets to create composite applications. Mashup enablers serve functionalities to mashup builders by accessing unstructured data, and making internal and external resources available. For example, Kapow Mashup Server, Openkapow and Feed43 belong to this category. Microsoft Popfly combines the functions of both the mashup enabler and the mashup builder in one environment. It has built-in connectors to various public domain Web applications, such as social networks like Twitter.com.

A mashup application leverages these tools, and follows integration patterns to create logic for data or function combination.

Mashup Applications. A number of research papers have applied mashup as a technique complementary to semantic Web for composing Web services. Liu at al. [10] described an architecture to build Web service composition using mashup techniques. Cetin et al. [2] proposed a service migration strategy in favor of mashup properties—composition of heterogeneous resources. Zou et al. [13] in-
introduced an ontology-based approach to present the obligation and liabilities of different roles involved in mashup. Some research work has also applied mashup techniques to other applications such as searching [1].

Our work is complementary to these methods at the composition level, through the connectors associated with each data source. Functions such as semantic filtering and service selection can be added to the connector. Our conceptual architecture still has space to address advanced issues of mashup for enterprise applications. It remains as future work to extend this architecture to manage security and accountability [13] when data are retrieved from third party services, data sources or applications.

6 Lessons Learnt and Conclusion

In this paper, we identify the new usage of three architecture integration patterns in mashup architecture design and development. The useful features of these patterns and their key components are realized through a conceptual architecture. The design of this architecture focuses on the integration and reusability of essential mashup components so that the mashups can be composed subject to business processes. We also discuss techniques for implementing and deploying map-based mashups. A case study of developing a Web-based PVS with an embedded mashup is presented to illustrate the practical usage of this architecture. Through our experience, we observe that component-based development helps to achieve good encapsulation of the operations required by mashups, and thus enables architecture patterns to structure key components of mashups. Through our experience, we summarize the lessons learnt in three aspects.

Mashup Deployment Architecture. The MVC pattern can be deployed as either client-side mashup or server-side mashup (see discussion in Section 3.3). However it is not
straightforward to make the design decision about which deployment architecture is suitable for a specific mashup scenario. We initially adopted the client-side mashup architecture, attracted by its simplicity and comprehensive tool support (Google Mashup Editor). However, as the mashup gets complicated with new features added, different operations need to be performed on different data sets. Eventually the client-side mashup was no long suitable and we started migrating to the server-side mashup.

**Data Redundancy.** The original Data Federation pattern creates virtual views without introducing data redundancy. However, for mashups, data are often reused and thus introducing some level of redundancy can both simplify the design and improve its responsiveness. The outputs from the Pipes and Filters produce the data in a more suitable structure and format to be processed by mashup components. They can be stored in a local database for processing, and reused later without going through the data processing Pipes and Filters every time they are needed.

**Optimization.** The aim of optimization is to enhance the user experience when interacting with the Web-based PVS. A mashup is user-centric, and thus the user experience is a key criterion for the design and implementation of a mashup. As the mashup component communicates with data intensively, optimizing data processing and transmission is essential to achieve efficiency and responsiveness for the benefit of the end users. Since the size of the KML file directly determines the waiting time that the end-user will experience, we focus on optimization techniques that can reduce the size of KML file and the number of bytes that have to be transmitted over the network. In addition to the general techniques 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 technique we employed is to reduce the number of decimal places for coordinates in the shape file. The more decimal places for coordinates, the more accurate the shape is. A coordinate with 6 decimal places has an accuracy of less than a foot. The accuracy level is sufficient to produce the base KML file in the PVS. This technique also reduces the size of the KML file. 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 current architecture is very focused on structuring the mashup flows, and does not consider some advanced features required by enterprise application including security. We envisage that this architecture approach utilizing integration patterns serves as a basis for further extension to address more complicated enterprise mashups requirements.

### References


---

5 A technical paper [8] provides detailed evaluation of a set of widely used XML compression tools.