UNIVERSITY OF GOTHENBURG

A Quality-based Framework for Leveraging the Process of Mashup Component Selection

SAEED AGHAEE

Master Thesis in Software Engineering and Management

Report No. 2009:075
ISSN: 1651-4769

University of Gothenburg
Department of Applied Information Technology
Gothenburg, Sweden, May 2009

Abstract

Mashups are a new and interesting brand of Web 2.0 applications. They are simply built from available mashup components on the Web providing functionality, and content. The ever-increasing number and diversity of mashup components makes the process of selecting proper components a challenging task. Hence, the present thesis work is directed towards presenting a quality-based framework providing a recommendation-based mechanism to enhance this process.

Keywords: Mashups, Mashup component, Service-Oriented Architecture, Quality evaluation.

Supervisor: Dr. Gerardo Schneider

Acknowledgment

In the first place, I would like to show my gratitude to Dr. Gerardo Schneider for his supervision, advice, and guidance from the very early stage of this work as well as giving me extraordinary experiences which will surely continue to inspire me in my future science career. Above all, he gave me the self-confidence to continue working on the present thesis.

Many thanks go in particular to Dr. Urban Nuldén and Dr. Jonas Landgren, who helped me determine the correct path towards the current thesis. Furthermore, I truly appreciate their trust and confidence in allowing me to perform research under their supervision during my master study. I am truly grateful for all that I learned from them so far.

In addition, I thank Dr. Miroslaw Staron, who helped me find a supervisor, and Niklas Mellegård, who provided me with valuable guidance to resolve some issues raised in the beginning of this work.

Finally, words fail me to express my appreciation to my family who supported me emotionally, spiritually, and financially throughout these six years of studying abroad.

Contents

1. INTRODUCTION

Web 2.0 has interestingly turned the Web into a more interactive, collaborative and enjoyable environment by letting user access and share a website content or even contribute to it [29]. The introduction of Asynchronous JAvascript over XML (AJAX) in 2005 [14] is considered as a profound contribution to Web applications as it enables a smooth interface for web applications (similar to desktop applications) by allowing data to be retrieved asynchronously in the background. As a result, user interaction with the Web application does not experience interference caused by client-server data transmission, and consequently it simulates the experience of using desktop applications.

Google has recently caused some buzz by revealing Chrome OS[1], which is a completely browsers-based Operating System (OS). The underlying concept behind the development of chrome OS is as simple as using software as electricity (plug and play), or in other words cloud computing. Regardless of whether or not Chrome OS will grab the market, it is suggestive of the important future of Web 2.0 applications in new generation of operating systems, at least from one of the world-leading IT company point of view, as being a potential replacement for current desktop applications. Therefore, Web 2.0 applications are undoubtedly getting more highlighted, and mashups, as a new brand of Web 2.0 applications, are no exception.

Web mashups owe their recent popularity to the effective balance between the overall development cost (time, resource, and money), and the versatility, efficiency, and quality of the final solutions they can provide. Web mashup simply aggregates data, content, and functionality from numerous heterogeneous sources on the Web into a new solution or service. Data sources are usually provided in the form of RSS/atom/XML feeds. There are also a vast number of companies (or individual developers) on the Web such as Google[2] and Twitter[3] offering functionality in the form of Web services. Mashup composers select and combine these composite Web services and disparate data sources from the Web to create a new composition providing programmable APIs, data, or applications [38]. In this work, such composite Web services are accounted for mashup components, which are accessible over Web via standard Web protocols (e.g. SOAP[4] and REST[5]), and expose an Application Programming Interface (API) allowing access to functionality or content.

It is self-evidence that the quality of a given mashup is highly dependent on some external quality characteristics of its building components [6], which can also distinguish a "proper" component from an "improper" component. The current thesis is intended to present a quality-based framework that can be applied for comparative evaluation of mashup components. The evaluation results will help the composer select a set of proper components, which can potentially result in composing a successful mashup.

1.1 Motivation

Despite the fact that mashups mostly target ordinary Web 2.0 users, by a rapid increase in the number and diversity of mashup components, we have been witnessing the capability of mashups to be deployed in a wide range of specialized domains such as emergency response [36], public health [7], and tourism [35]. At the same time, the growing number of mashup components imposes challenges on searching and selecting suitable components. Web APIs directory Websites such as ProgrammableWeb[6] are actively enhancing access to mashup resources by maintaining an up-to-date list of mshup components classified into several categories based on their functional similarities. So it can potentially reduce the effort required to search for a candidate group of components possessing the functionalities of interest. However, selecting a suitable component among a group of candidates, which all provide the same functionalities, remains as a challenging task.

In the present work, the main contribution is made towards addressing the last mentioned challenge. Additionally, the attempt is also made to provide a suitable platform to address this challenge by presenting a conceptual model for enterprise mashup design (Section 4).

[1] http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html - Accessed 20 November 2009

[2] http://code.google.com/ - Accessed 10 October 2009

[3] http://apiwiki.twitter.com/- Accessed 10 October 2009

[4] Simple Object Access Protocol

[5] Representational State Transfer

[6] http://www.programmableweb.com/ - Accessed 1 December 2009

1.2 Thesis Outline

Section 2 is dedicated to review the relevant literature as a background study. In section 3, we foster the framework by proposing a tailored model for enterprise mashup design. In section 4, we present the framework in detail. To put the framework into practical experience, we apply it on a case-study, which is a mashup proposal for emergency response (section 5). Eventually, we discuss relevant work in this regard in section 6, and draw conclusions highlighting some potential future work in section 7.

2. BACKGROUND

In order to fully understand the rest of the thesis, this section provides a concise introduction on the underlying concepts of this work which were based on a review of relevant literature. These concepts include mashups and key actors involved in mashup development, ISO/IES standard, and the quality model presented in [6].

2.1 Mashup

In spite of the fact that the idea behind the term mashups is not new, yet it is not amenable to precise definition and may vary over time. The reason might be due to the dramatic progress in web-related technologies, as a consequence of which the lens, through which we look at mashups, gradually gets wider. However, the basic definition of mashup, as described in [27], can be conceptualized by the interaction among three disjoint participants: mashup component provider, mashup hosting site, and end-user's web browser (figure 1).

Respectively, API/content providers, ranging from individual developers to industry player companies, integrate content and functionality into a single mashup component, which exposes an Application Programming Interface (API), and is accessible via web protocols such as REST and SOAP. The mashup site is where the mashup is hosted. Mashups can be executed on both server-side (using server-side languages) and client-side (using scripting languages). Eventually, the last participant is user's Web browser, in which mashup is graphically rendered and presented to end-user.

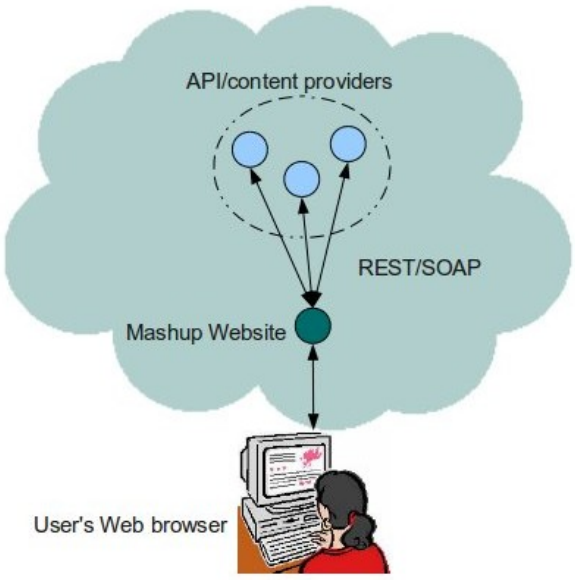

Figure 1. Mashup architecture comprises three disjoint participants: API/content providers, mashup's Website, and user's Web browser.

2.1.1 Consumer and Enterprise Mashups

On the other hand, mashups fall into two types: consumer and enterprise. The most popular mashups are those built for ordinary users and are classified as consumer mashups. This type of mashup provides interesting solutions for general purposes such as weather forecast by aggregating heterogeneous mashup components into a single representation. Figure 2 illustrates a snapshot of Woozor[7] which is a successful mashup recognized by MashupAwards[8] as "mashup of the day". Woozor provides users with 10 days global weather forecasts. It uses functionality from Google Maps[9] and retrieves data from The Weather Channel[10].

There is a widespread interest and acceptance of mashups in today Service-Oriented Architecture (SOA) industry [37]. This type of mashups is specially composed for the enterprise. Such mashups are characterized by a massive collaboration between the end-user, as the main stakeholder, and the mashup composer, so as to meet the stakeholder requirements [18]. However, there is no specific boundaries between consumer and enter-

[7] http://woozor.com/ - Accessed 20 October 2009
[8] http://mashupawards.com/winners/ - Accessed 20 October 2009
[9] http://code.google.com/apis/maps/ - Accessed 20 October 2009
[10] http://www.weather.com/weather/rss/subscription - Accessed 20 October 2009

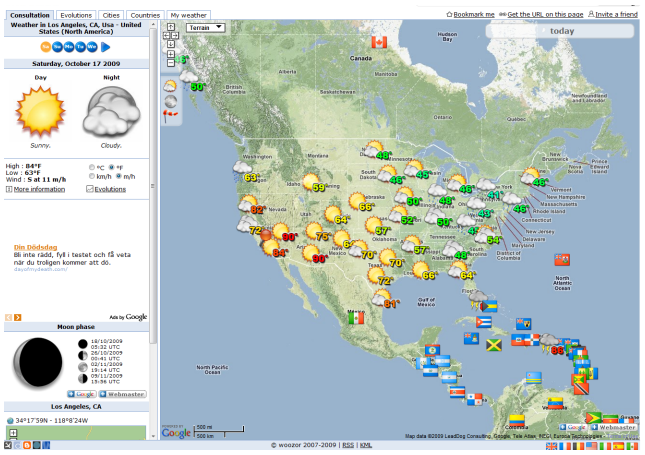

Figure 2. Woozor is a "mapping mashup" based on Google Map. It retrieves data from The Weather Channel and presents it to the users in an effective way.

prise mashup in a sense that both of which relies on the same concept: "resource composition style" [17]. From development perspective, enterprise mashup can not be supported by traditional software development paradigm, but instead has to be underpinned by Agile software development model [18].

2.1.2 Mashup from SOA Perspective

A SOA is essentially a collection of services, which can be discovered, invoked through standard protocol (e.g. SOAP), and published in a public registry (e.g. UDDI). SOA places the emphasis on loose-coupling of services. It refers to the ability of services to function independently from machine.

From SOA perspective, mashup is a service composition style [24], as it aggregates functionality/content from the Web into a single representation. However, as described in [24], mashup can be interpreted as a tailored version of SOA which is characterized by four properties:

- More Reusable: Mashup is a composition of black box components handling their own context and business logic. It makes mashup more reusable, comparing to current SOA technologies such as BPEL (Business Process Execution Language) and WSCI (Web Service Choreography).
- Web-Based: Mashup is a web-based application as it uses scripting languages (PHP, JavaScript), and popular data formats on Web (XML).
- Light Weight: Mashup uses external functionality and content, which is why it is light weight to implement.
- End Consumer Centric: Mashup can be composed by use of available tools on the Web. Consumers without programing skills, therefore, can compose their own mashup.

2.2 Review of ISO/IEC 9126

ISO/IEC 9126 is an international standard which comprises four parts: quality model [19] , internal metrics [20], external metrics [21], and quality in use metrics [22]. It introduces a framework to assess three aspects of software quality: internal quality, external quality, and quality in use. Internal and external quality are concerned with software product quality and are distinguished with the fact that external quality is assessable when the software is executable, whereas in early development stage, when the software is not executable, the evaluation merely stresses the internal quality aspect of the software. Finally, evaluation of quality in use deals with the final software running in the target environment and under real conditions.

According to ISO/IEC 9126-1 [19], quality of software products (internal and external quality) can be evaluated by arriving at a quality model as a hierarchy, derived from a combination of main quality characteristics, which in turn are branched into subcharacteristics, which are further divided into attributes. ISO/IEC 9126-1 specifies a fixed set of main characteristics for the top level: Functionality, Reliability, Usability, Efficiency, Portability and Maintainability. Moreover, the sub-characteristics level is also explicitly outlined in the standard. However, it does not restrict determination of attributes, as a result of which the model becomes more flexible, and versatile to be extended to cover various application domains.

The lowest level of quality model in the hierarchy (attributes) is associated with software metrics to quantify the quality characteristics, which further facilitates the quality assessment. Generally, software metrics enable concerned-people to achieve meaningful measurements on quality of software product and the development process which it utilizes [28].

2.3 A Quality Model for Mashup Components

Considering mashup components as building blocks exposing an Application Programming Interface (APIs), developer information (supported data types, security issues, and information about usage), and occasionally a User Interface (UI) to mashup composers, [6] follow a black-box approach to select proper quality characteristics for external quality assessment of mashup components. The quality characteristics and sub-characteristics are derived from ISO/IEC 9126-1 [19], and are organized into a quality model covering the three logical constituents of mashup components: API layer, data layer, and presentation layer. We consider it as a reference quality model by which to assess the quality of mashup components while aggregating them into a single representation.

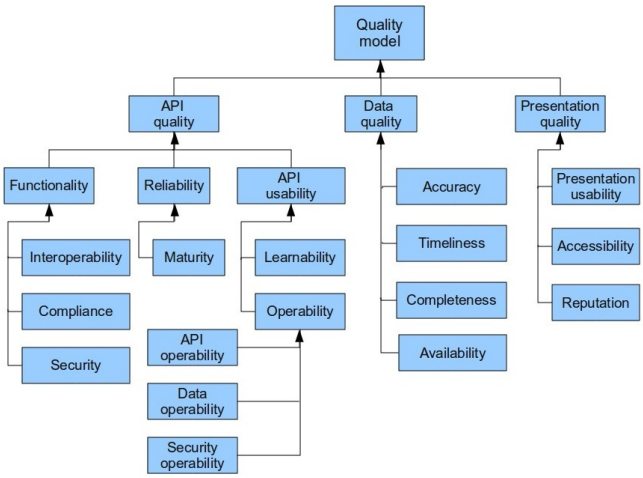

Figure 3. A quality model presented in [6] for external quality assessment of mashup component.

Additionally, [6] presents a set of metrics mapped to the end-point characteristics of the quality model. In this work, we are also intended to reuse a part of these metrics (see appendix A), as well as to to propose a number of new metrics (see section 5.3.2). The main motivation behind the metrics generation mostly comes from the analysis of ProgrammableWeb, according to which every mashup component possess a technical specification page describing: a) protocols and/or languages through which APIs are accessible, b) data formats supported by APIs, c) a brief outlook into the component security (use of SSL, authentication model, and anonymous access privileges) and usage (service endpoint, and client-side usage), d) and provider specifications and licensing.

3. METHOD

In this work, we follow one of the most frequent research strategies in software engineering, which is proposing a method to analyze a piece of software and validating the results through analysis of an example [33].

In Software engineering research, as any other research disciplines, one of the initial steps in finding a proper research method is to assure the research question is crystal clear [12]. To this end, a deep literature review on the subject area is a necessary pre-step to take. Having the research questions in one hand, and the classification of research questions by [26] in the other hand, we could categorize the current research question as "description and classification question" [12].

As a matter of fact research produces knowledge in the form of a specific result [33]. The second step is thus to clarify the type of the result we wish to achieve. In the current thesis, the result is a quality-based framework providing sequential steps to select and evaluate candidate mashup components.

Finally, the last step is to clearly convince that the result is valid [33]. In software engineering research, this step can be done through a variety of techniques namely, as it was stated by [33], analysis, experience, evaluation, persuasion, and example. In this work, we provide evidences that the framework is applicable. For this purpose, we validated the result by applying it (result) on an example (case-study), which is a mashup tool for emergency response.

4. COMPONENT-BASED DESIGN OF ENTERPRISE MASHUPS

The framework, which will be presented in section 5, seeks to leverage the process of mashup component selection. Accordingly, it is worth providing a conceptual model taking a broader view on this process by describing its interaction with other activities undertaken during mashup development. Since the focus of this work is specially on the selection of suitable components, we keep the model simple by disregarding the activities that might be performed after it. We rather

expand the model upward in order to identify complementary activities, which together provide a platform for a proper component selection process. As it is illustrated in figure 4, these sequential activities are respectively requirements engineering, mashup design, and mashup component selection. We postone the description of mashup component slection process to section 4, in which the framework will be explained in detail.

Our tailored model is inspired by Component-Based Software Development (CBSD). Though mashup is not naturally a component-based system, we believe that its design process shares similarities with CBSD in terms of general characteristics, which the main one could be the fact that mashups, like component-based systems, are build from components that maintain their own business model and logic. Therefore, concepts driven from CBSD can provide a skeleton around which to design mashup. Afterward, as seen later in this section, the detailed design, i.e the sub activities, are supported through research in Web Services and SOA.

The widespread of mashup tools on the Web enables even end-users with less programming knowledge to compose simple mashups. Examples are Yahoo Pipes[11], Intel Mash Maker[12], and IBM QEDWiki[13] which provide end-users with a Web-based tool to collect and combine different services (or mashup components) on the Web for the purposes of composing mashups. As a consequence of that, the effort of implementation, i.e. component integration and mashup development, is reduced to simple drag-and-drop of mashup components. In such a case, the provided model expresses all the required activities undertaken as a matter of routine for development of mashup.

To our best knowledge, enterprise mashup design lacks such a top-down structured model, though the activities presented in this model have been scatteredly addressed in a number of papers. Our goal is, therefore, not only to provide a top-level view on the framework, but also to propose a structured approach for enterprise mashup design, inspired by CBSD, by aggregating and incorporating relevant works into a tailored model. Thus, we overview the model by pointing out several works,

[11] http://pipes.yahoo.com/pipes/ - Accessed 20 November 2009

[12] http://mashmaker.intel.com/web/ - Accessed 20 November 2009

[13] http://services.alphaworks.ibm.com/graduated/qedwiki.html - Accessed 20 November 2009

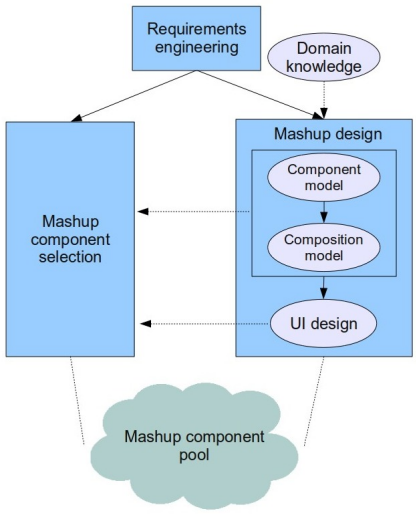

Figure 4. Conceptual model

which are essentially correlated with each activity and sub-activity included in it.

4.1 Requirements Engineering

The starting point is based on a set of requirements, which are derived from end-user needs. Mashup requirements can be captured in many ways; [13] lists and compares traditional techniques for software requirement engineering to be tailored to Web applications. [4] also describes a goal-oriented approach to capture requirements for Web applications. A well-captured set of requirements is the basis for establishing both the design phase and the component selection process.

4.2 Mashup Design

Today, mashups go well beyond a simple UI; instead, mashups may utilize several logical layers that together make up the intended mashup solution. Hence, development of mashups is in need of an architectural view depicting the top level design of the mashup. In this context, the architectural aspect provides an abstracted view on different aspects of a mashup such as component integration, structure, and UI composition. This kind of view has been addressed in several specialized domains, [23] describes a reference architecture for thematic mashups. [34] tailors mashup to Geographical Information System (GIS) tool for emergency response and describes a proposal architecture of such a system.

[1] presents a lightweight enterprise mashup for data integration and outlines its architectural aspects.

As it is illustrated in figure 4, mashup design is undertaken in the light of domain knowledge, by being able to access a mashup component pool. Likewise, component selection process is influenced by the sub-activities performed in design phase including three important aspects of a mashup design: component model, composition model, and UI (User Interface) design.

4.2.1 Mashup Component Pool

It is worth mentioning that having access to a mashup component pool, by which we can assure existence and availability of a particular mashup component, is critical to the mashup design as well the selecting component process. There are few Websites on the Web addressing this need by collecting and cataloging mashup components into a unified representation, examples are ProgrammableWeb and WebMashup[14].

4.2.2 Domain Knowledge

Achieving a successful mashup design requires a general knowledge about the target domain, to which the mashup is intended to be deployed. This is specially demanded when mashup deals with a specialized domain like Bioinformatics [15]. We can look at the term domain knowledge in the light of traditional software engineering, which conceptualizes it as a valid knowledge that should be learned from end-users within the domain.

4.2.3 Component Model

With a set of requirements we can proceed to design the intended composition, which specifically relies on two complementary blocks: components model, and composition model [38]. A component model describes different aspects of mashup components. There have been several papers describing a specific model for mashup components. [38] characterizes a component model by three properties: type, Interface, and extensibility. [8] uses UISDL (UI Service Description Language) to describe component model.

Nevertheless, a big challenge is being able to retrieve component specifications, as well as to assure the validity of those specifications. In this regard, ProgrammableWeb is considered as a valuable resource for mashup composers as it contains daily updated list of Web Mashups and APIs, which are accessible through a unified search interface. Beside, some of the work regarding mashup widely base on statistical analysis of data obtained from ProgrammableWeb [6, 41]. Regardless of the variety and excess of mashup components, ProgrammableWeb collects and categorizes them according to their context of use (table 1). Another important feature afforded by this website is providing evidence on what (and how many) mashups use a specified component and vice versa.

Component category	Number of registered components	Examples
Internet	99	Amazon EC2
Mapping	97	Google Maps
Social	88	Twitter
Financial	77	Kiva
Reference	69	Wikipedia
Videos	59	YouTube
Shopping	58	Amazon eCommerce
Music	57	Last.fm
Messaging	55	411Sync
Search	54	Google Search
Telephony	49	Skype
Photos	47	Flickr
Enterprise	44	Salesforce.com

Table 1. The above table outlines some of the most popular component categories and the number of registered components in each category. Data from this table have been collected from Programmableweb (http://www.programmableweb.com/) in December 2009. Because of the space limitations we could not include each component homepage. Alternatively, there are accessible from either the mentioned website (ProgrammableWeb) or through Google search engine (http://www.google.com/).

4.2.4 Composition Model

A composition model can be described through WS-BPEL (Web Service Business Process Execution language) [5, 30], or simplified composition languages [25, 32] . Basically, it designates how available mashup

[14] http://www.webmashup.com/ - Accessed 1 December 2009

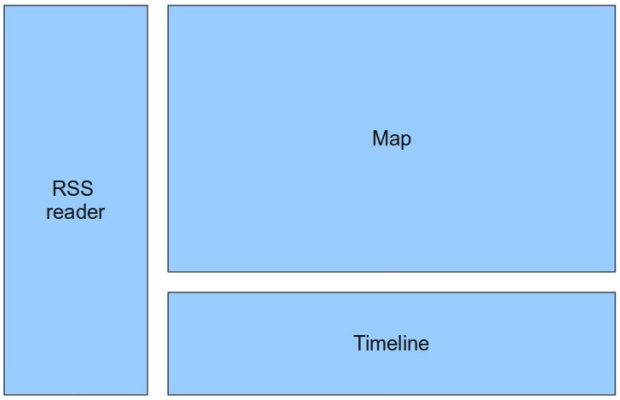

Figure 5. User interface design of a mashup for emergency response. As it can be seen, the timeline is located beneath the map, which makes it easier for users to interact simultaneously with both map and timeline (spatio-temporal analysis). The RSS feed reader, which contains the items that are presented on the map, is located in the left-side.

components are merged together to create the intended mashup. [38] argues that a composition model can be characterized based on three factors. First, the composition output type, which can be classified into Data, API, and application with a user interface. Second, the orchestration style determining how mashup components are supposed to interact with each other at run time. For instance, [8] presents an event-driven approach for creating context aware adaptive composition models. Third, the data-passing style facilitated through two approaches: data-flow, and blackboard. Data-flow approach places the emphasis on data-flow from one component to another. In contrast, blackboard approach writes data to variables, much like programming languages, which are treated as the source and destination of component function invocations.

4.2.5 UI Design

The lower level design of a mashup is the phase in which the final UI is designed. Mashup is a composition of components that generally implement their own UI (UI components), whereby mashup UI becomes an aggregation of non-overlapping UI components. As a result, design of a mashup UI confronts a number of challenges that the most important one could be effective placement of the UI components. One example could be the importance of UI design in time-critical work such as emergency response. In this regard, the UI should be kept as simple as possible and the UI component should be placed in a position where they are able to predict (Figure 6). Though these design principles seems easy to address, they play an important role in the applicability of mashups. Such issues have been comprehensively addressed in a number of previous studies [9, 39, 40].

5. FRAMEWORK

In the present work, the process of evaluating components and selecting one of them is called mashup component selection. This step, which is done during mashup development, is undoubtedly an important influence on the mashup success in terms of fulfilling mashup goals and attracting end-users. Moreover, it can considerably reduce the overhead maintenance time and efforts which might be caused by conflicts and defects occurred after the final mashup is deployed. To leverage this process, we present a framework consisting of the following steps:

- Mapping mashup requirements to components
- Selecting candidate components
- Evaluating the quality of candidate components

The underlying part of the framework is based on cascading mashup-level requirements to (mashup) component-level requirements, enabling us to select a set of candidate components, fulfilling the functional requirements. The next part contains a metrics suite to evaluate the quality of candidate components from the non-functional view. The metrics suite is mapped to the end-point characteristics of the the reference quality model [6] (see section 2.3), and measures the degree to which a component has been selected properly.

5.1 Mapping Requirements to Mashup Components

Mashup is essentially characterized by a resource composition style, meaning that it is merely created by drag and drop existing resources that can be functionality, content, or data. Therefore, except the required code or logic implementing the interaction of the components, mashup rarely implements its own services. This natural property suggests that no mashup requirements exist without having a connection to its components. Therefore, mashup requirements are established through the

joint collaboration of a group of components or explicitly rely on a single component.

As it was described earlier, a composition model clarifies the way in which components are integrated with each other. To be specific, it suggests what type of component (e.g. map component), should be used in the final composition and describes how they interact with each other. In this stage, the requirements have been broken down to function level, which in turn can be easily mapped to defined components in the composition model.

To simplify the requirement mapping process, we separately consider functional and non-functional requirements and amplify our point by giving some examples.

5.1.1 Functional Requirements

Functional requirements describes what a given mashup is supposed to accomplish. To be specific, they outline the essential functions of a given mashup. Consider a mashup for weather forecast, whose composite model defines four types of component: geocoder, map, RSS reader, and timeline. Let r be a functional requirements requiring that mashup shall present weather forecast on the map separately for each 10 day". Given the composition model, this requirements is particularly connected to timeline and map components. It first clearly requires that map component shall be able to manage (show, delete, move) icons. For the timeline, it shall provide a feature enabling scrolling through at least 10 days. Besides, it should provide a functionality to notify the map upon any change in the timeline status (i.e. selecting a different day).

In this stage, the orchestration model suggests the required functions that each component should be capable of performing. We can divide functional requirements of mashup, with respect to the way they are mapped to components, into two groups. The first group can be explicitly mapped to a particular component. For instance, a requirement might be that mashup shall present forecasts using different icons on the map. In this case, it is directly connected to the map component.

To meet the second group of requirements, two or more components should interact with each other. This imposes a set of requirements that each involved component should meet. In order to map such requirements, one can use a check-list approach, in which a given mashup requirement is described in a scenario highlighting the requirements from each involved component. The first mentioned example is fitted into this group.

5.1.2 Non-Functional Requirements

A given mashup requires some criteria, based on which the quality of its functions can be evaluated. Such criteria are considered as non-functional requirements and can describe mashup properties related to e.g. security, performance, usability, functionality, and so forth. Similar to functional requirements, they are affected by one or a group of components.

In the previous example, if the mashup requires the highest-possible level of usability, the UI components (geocoder is excluded) must have the required level of usability. As another example, availability of a mashup is influenced by all of its components in terms of data and services.

5.2 Selecting Candidate Components

By having mapped requirements to components, we can proceed to collect a range of candidate components meeting the specified functional requirements. In this regard, ProgrammableWeb could be a good starting point as it categorizes mashup components based on functional similarities (table 1). A challenge is how we can judge if a component, within the category of interest, can be fit into the candidate group. In fact, the component meets the required functional requirements by providing two categories of functions: internal functions, and API functions.

- Internal functions: are not accessible through its APIs, and are usually identified in the component descriptions. A component may provides API functions to disable or enable internal functions.
- API functions: are invoked through the component APIs. Almost all components have their APIs documented, and ideally the APIs have been categorized into the functionalities the component can provide. based on that, we can realize if a component possesses the desired API functions to meet the necessary functional requirements.

Empirically, when it comes to select candidate components, popular components are better choices to

begin with. ProgrammableWeb makes it feasible by categorizing and sorting components based on the level of popularity. It is a vantage point of using ProgrammableWeb to access mashup component pool.

5.3 Quality Evaluation of Candidate Components

Having targeted a group of candidates for each component type, the next step is determination of the level to which each candidate component satisfies its non-functional requirements. This step is based on measurement-based evaluation of non-functional requirements of one candidate component against another. For this purpose, We first conceptualize component non-functional requirements based on its stakeholders view, from whose point of view the quality model is reviewed again. Secondly, we associate a metrics suite to the end-point characteristics of the quality model.

5.3.1 Component Non-Functional Requirements

In addition to the non-functional requirements that must be met at run-time and are driven from mashup non-functional requirements, mashup composers may also impose a number of non-functional requirements, which designate the level of component adaptation within a given composition model. An adapted component must cause as minimal conflicts as possible in terms of data types, programming languages, and protocols. It should also be able to provide efficient access to its functionalities, which are required within the composition.

Basically, mashup non-functional requirements stress those aspects of a component which are essentially tangible by its end-users such as data and presentation quality. In contrast, mashup composer might consider those quality characteristics concerned with the quality of component API, and are merely important during development. From another angle, component non-functional requirements are directly and indirectly expressed by two main stakeholders: mashup composers, and mashup end-users. The quality model mentioned earlier (see section 2) relies on both the stakeholders [6], which is why it can be used as a reference quality model in this work. Based on the stakeholders view, we divide the quality model (figure 7), so as to provide better understanding for quality measurement and evaluation, though both groups of characteristics are influential on mashup success.

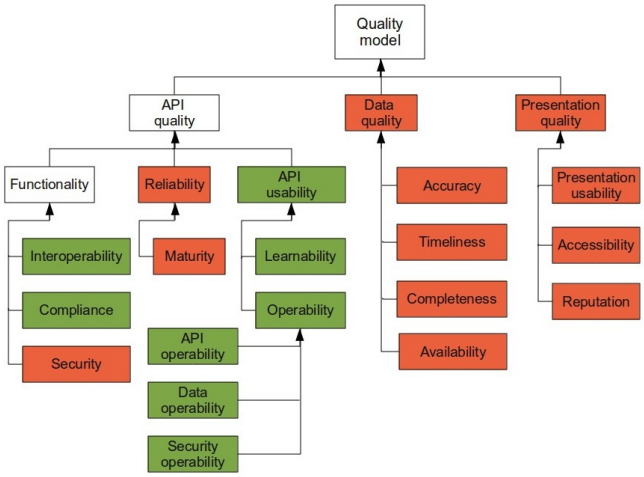

Figure 6. Requirements imposed by mashup composers are shown as green colored characteristics, which reasonably belong to API quality. Red colored characteristics are mapped from mashup requirements gathered from end-users.

As it can be seen in figure 6, non-functional requirements that might be imposed by composer are a subset of some characteristics belonging to API quality. It includes interoperability, compliance, and API usability.

The remaining characteristics, therefore, include all possible non-functional requirements that can be indirectly driven from mashup end-users. A translation of non-functional requirements to quality characteristics and sub-characteristics is sometimes required. For this purpose we first relate the non-functional requirements to one of the logical parts of mashup components (API, data, presentation), we can then correspond it to one or more characteristics or sub-characteristics. For instance trustability is concerned with API quality, and can be corresponded to reliability and maturity.

5.3.2 Metrics Suite

Metrics presented in [6] were basically supposed to be used to assess the component quality from its developer's point of view, so as to contribute to development of high quality mashup components. However, in the present work, the attempt is made to measure the quality characteristics from the mashup composer's viewpoint. From the goal-oriented measurement perspective, metrics determination is potentially subject to in-

fluence from the choice of viewpoint, from which the measurement is performed [2, 10]. Such a change in viewpoint (from component developer to mashup composer) thus results in a change of some metrics, which will be discussed subsequently.

A component developer should take account of two main stakeholders: mashup end-user and mashup composer [6]. Since both mashup composer and component developer's viewpoint are in consistent with respect to potential mashup end-users, the metrics associated to the end-point characteristics, which are mapped from the non-functional requirements driven from end-users (red colored characteristics in figure 6), are not subject to change, but instead we reuse them in this work (see appendix A).

However, the two viewpoints expressed above are not based upon a certain position regarding the non-functional requirements (quality characteristics) imposed by mashup composers. To be specific, mashup component developers look at these quality characteristics in a general and abstract way, whereas mashup composers assess them with special regard to their intended mashups. For instance, interoperability and compliance should be essentially considered in accordance with other components and their interactions in the final composition. From this viewpoint, we will redefine the metrics mapped to the following end-point characteristics:

- Interoperability: will be complemented by adding a new aspect and a metric mapped to it.
- Compliance: will be replaced with a new metric.
- Learnability: will be mapped to a new metrics.

Interoperability

Interoperability is defined as the ability of a mashup component to work with various components in a given mashup composition. Interoperability of a component can be investigated by inspecting the following aspects of its APIs: the number of APIs provided to support required functions, available technologies to access APIs, and supported data types.

The last two aspects of an interoperable component were addressed in previous work [6]. There are different technologies through which components expose their APIs. It can be via either Web protocols or scripting languages. The most well-known Web protocols are REST and SOAP. The scripting language is most commonly JavaScript. The data types used within the APIs are also of importance when interoperability issues arise. Generally, the more technologies, including protocols and scripting languages, and data types supported by a component, the more interoperability capabilities it has with other components.

However, the first aspect also arises when assessing interoperability from a mashup composer's viewpoint. It is thus influenced by the number of offered APIs for the required API functions. The more provided APIs, the more alternative ways to accomplish a task, and consequently, the more interoperability level within the composition. Therefore, interoperability of provided APIs can be obtained by the following metric model:

$$\sum_{i=1}^{n} NAPI_i$$

In the above formula, n is the number of required functions, and $NAPI_i$ is the number of provided APIs for the i^{th} function ($1 \leq i \leq n$). Therefore, interoperability can be obtained as the sum of the metrics mapped to the three aspects (e.g. data, protocol, and API).

Compliance

In order for a mashup component to be compliant, it must provide a standard mechanism for accessing and using its APIs. From a mashup composer's viewpoint, compliance means the ability of a given component to be compliant with the demanded technologies within the intended mashup. These technologies, with respect to component APIs, can be categorized to Web protocols, programming languages, and data types. Thereby, a mashup component is compliant if and only if it can support the required Web protocols, programming languages, and data types. We replace the metric model presented in [6] as follows:

$$RDT \wedge RWP \wedge RPL$$

In the above formula, RDT, RWP, and RPL return $true$ if the component supports, respectively, the required data types, Web protocols, and programming languages.

Learnability

Learnability is an important aspect of API usability. It suggests the capability of a component to enable mashup composers to learn how to work with its APIs. Due to the fact that mashup components are usually published under non-commercial license (i.e. commercial support is not available), adequate learning resources is of special importance. We classify the API learning resources and associate a learnability point to each type:

Resource type	Learnability point
API documentation	1
API tutorial	2
Sample usage	3
Active technical forum	4

Table 2. Learnability resource types

Empirically, most of component providers publish up-to-date documentations of its APIs. Well-known providers commonly provide comprehensive tutorials in some form of HOW-TO documents. Additionally, while programming APIs, sample source codes will serve as a great help. A more valuable resource is an active technical forum, where you can access previously solved problems or request a solution regarding an ongoing technical issue. We propose the following metric model to determine learnability:

$$\frac{sumOfPoints}{10}$$

$sumOfPoints$ is the sum of the points obtained from table 2 by considering the available learning resources a component provides. It is then divided by 10, which is the total number of points $(1+2+3+4)$. The result Will be a number between 0 and 1 ($0 \leq$ and ≤ 1).

6. CASE STUDY

In this work we apply the framework on a mashup implemented as a part of an ongoing project in Crisis Response Lab at the IT University of Gothenburg[15]. We chose emergency response as an interesting potential area for mashup development. Mashups inherit Web 2.0 characteristics like collaborative environment with a high number of users involved. Such characteristics are very important in the area of emergency response, due to the fact that appropriate information sharing, and collaborative work during emergency response are considered as the key success factors in this domain. For these reasons, we believe that mashups are good candidate tools for emergency response work and are capable of proposing interesting solutions to address the current challenges in this domain.

[15] http://crisisresponselab.blogspot.com/2009/05/mashups-for-formal-response.html

6.1 Design

Before applying the framework on our intended mashup, we briefly outline its design aspects. Incident commanders at the command and control center should play a central managerial role by employing corresponding agencies and assuring that all involved agencies and authorities are aligned and coordinated toward the same target. This demands an interactive tool to dynamically monitor the involved agencies at the incident side. Corresponding agencies can be monitored by means of unregulated dynamic feedback from them. Such dynamic feedback are called event log entries produced in the midst of a crisis and contain taken actions, decision made, or reports from the incident side.

The system thus was implemented as a distributed system, whose architecture is decomposed to a monitoring tool at the command and control center and mobile devices at the incident side equipped with positioning capabilities, e.g. GPS (Global Positioning System). Incident-side commanders use mobile devices to generate event log entries. The monitoring tool is a mashup used at command and control center, and receives event log entries in the form GeoRSS feeds. The goal of the mashup, which is the focus of this case study, is to offer a set of highly interactive features for management and monitoring of event log entries such as storing, exploring, and filtering. These features have profound impacts on analytical reasoning processes by boosting the ability of users to detect spatial patterns and conduct exploratory spatio-temporal analysis. In this case study, we focus on the mentioned mashup.

Accordingly, the main non-functional requirements for the mashup include availability, usability, security, and trustability. As key functional requirements it shall: a) show geographical and temporal status of event-log entries; b) filter event log entries based on time and organization; c) enable geographically and temporally

exploration of event log entries; d) notify users upon new event log entries.

To determine service orchestration model we use the approach proposed in [5]. We started by a high level process model expressed using BPMN (Business Process Modeling Notation), and then transferred it to a more detailed WebML (Web Modeling Language) orchestration model. Figure 7 illustrates the mashup design.

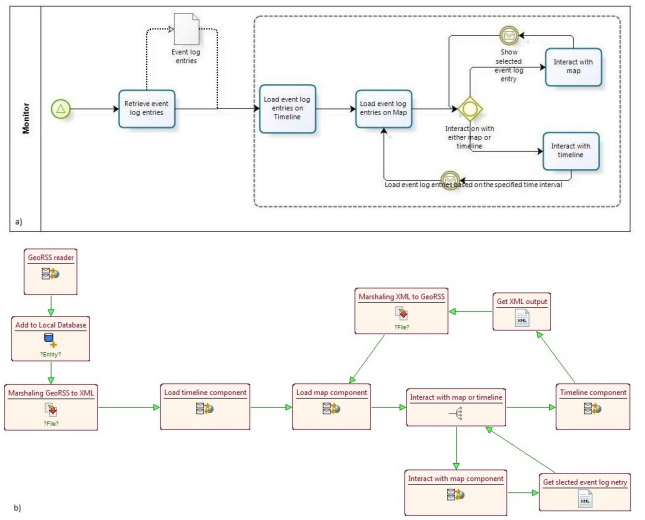

Figure 7. a) BPMN model b) WebML orchestration model transferred from the BPMN model

6.2 Applying Framework

The orchestration model reveals functional decomposition of the mashup into components. According to the orchestration model, the mashup should use three types of component: timeline, map, and RSS reader. In this case study, we merely apply the framework presented in section 5 on the map component.

6.2.1 Mapping Requirements

The first step is mapping the requirements to mashup components. It can be facilitated by investigating the orchestration model clarifying the interactions among the components of the mashup. The main functionalities that the map component should provide are, therefore, outlined and classified as follows:

1. Zoom and navigation (internal function).
2. Manage markers with custom icons (API function).
3. Manage balloons with custom text, image, video (API function).
4. Manage polygons and polylines (API function).

According to the key non-functional requirements of the mashup, a candidate map component should fulfill user's non-functional requirements of availability, usability, and security.

6.2.2 Selecting Candidate Components

As a second step, a group of candidates for map component should be selected in accordance with the key functions specified earlier. Using ProgrammableWeb, we considered the components in mapping category, and then collected a number of component possessing the required API and internal functions. To ease this process we started with the most popular components. The candidates group includes Bing Maps[16], Google Maps, and Yahoo Maps[17].

6.2.3 Evaluating Quality

Finally, we proceed to evaluate candidate components quality from the point of view of both the mashup composer and end user. Based on the reference quality model, The main quality sub-characteristics imposed by mashup composer include interoperability, compliance, learnability, and operability.

According to the component non-functional requirements (mapped from end-user needs), we are interested in measuring the following sub-characteristics: data availability, presentation usability, and security (functionality).

- Interoperability

As mentioned, interoperability of each component can be measured by counting the number of the protocols, and data formats it supports, as well as the number of APIs it provides for the required API functions. Investigating the components website as well as their specification pages in ProgrammeableWeb allows us to assess interoperability of each component as outline in table 3.

[16] http://www.microsoft.com/maps/isdk/ajax/
[17] http://developer.yahoo.com/maps/ajax/

It is worth mentioning that the number of provided APIs for each API function was acquired by investigating the API documentation of each component. The counted API units include both the event types required to perform a specific API function (e.g. managing markers).

Component	Number of supported protocols	Number of supported data formats	Number of provided APIs	Interoperability
Yahoo Maps	3 (Rest, JavaScript, Flash)	2 (XML, GeoRSS)	F1: 27 F2: 1 F3: 4	37
Bing Maps	2 (JavaScript, SOAP)	3 (XML, KML, GeoRSS)	F1: 29 F2: 11 F3: 24	69
Google Maps	1 (JavaScript)	4 (XML, VML, JSON, KML)	F1: 10 F2: 11 F3: 15	41

Table 3. Interoperability

• Compliance

According to the composition model, in order to ensure compliance, the candidate components should be able to support GeoRSS feeds (table 4).

Component	Compliance
Yahoo Maps	Yes (supports GeoRSS)
Bing Maps	Yes (supports GeoRSS)
Google Maps	No (does not naturally support GeoRSS)

Table 4. Compliance

• Security

Security of a component can be assessed by considering its authentication type and whether or not it is done over Secure Sockets Layer (SSL). According to the security metric (see appendix A), table 5 suggests the security assessment of the candidate components.

Component	Security
Yahoo Maps	1 API key
Bing Maps	2 API key+SSL support
Google Maps	3 API key

Table 5. Security

• Learnability

Learnability is also assessed by searching the components Website for the purpose of looking for the available learning resources they offer. Table 6 contains the measurement results for learnability.

Component	Learnability
Yahoo Maps	1 (offers all learning resources)
Bing Maps	1 (offers all learning resources)
Google Maps	1 (offers all learning resources)

Table 6. Learnability

• Operability

Operability, as another aspect of API usability, refers to the ease-of-use of the APIs offered by a component. It can be assessed by measuring three attributes: API operability, data operability, and security operability [6]. Operability assessment of the components is shown in table 7.

• Availability

Emergency response is naturally a time-critical work, in which all activities are to be accomplished in a very condensed time. For this reason, availability necessarily arises as an important issue for evaluation of candidate components. Service availability of candidate components can be assessed by considering any API call volume limitations imposed by each one. In this case study, all the candidate components impose a limitation in terms of the number of API call for geocode

Component	Operability Attributes	Operability
Yahoo Maps	API: 3 (JavaScript) Data: 3 (Parameter) Security: 4	10
Bing Maps	API: 3 (JavaScript) Data: 3 (Parameter) Security: 4	10
Google Maps	API: 3 (JavaScript) Data:3 (Parameter) Security: 3	9

Table 7. Operabililty

requests per day. This number is 50000 for Yahoo Map and Bing Map, and 15000 for Google Map.

- Presentation Usability

All the candidate components provide the same features for navigation and interaction with object (markers) on the map. Thereby, presentation usability can be assessed by considering the quality of the satellite images the candidate map components provide within Gothenburg city (since the mashup is intended to be used within Sweden). As it can be seen in figure 9, Yahoo Map presents old satellite images, in a way that some buildings are missing. Additionally, it offers a lower zooming level compared to Bing Map and Google Map. In contrast, both Google Map and Bing Map present good quality images with the same zooming level. However, Google Map, compared to Bing Map, provides sensibly blurry satellite images, and with lower quality.

6.2.4 Component Selection

According to the quality evaluation (table 8), Bing Map is compliant, more secure, available, and with better presentation quality. The second choice is, therefore, Google Map fulfilling the most important non-functional requirements (usability, availability, security), though it is not naturally compliant with GeoRSS. Yahoo Map is considered as the last choice, since it presents low quality satellite images, even though it properly fulfills other non-functional requirements.

7. DISCUSSION AND RELATED WORK

Related works in connection with recommendation-based mashup component selection can be differentiated based on whether the focus is on replacing components of an executing mashup, or selecting suitable components when composing mashups. The first mentioned focus is to be understood in the light of adaptivity and context awareness suggesting the ability of mashup to react based on requirement changes in the

Figure 8. Snapshots from Bing map (1), Google map (2), and Yahoo map (3) zoomed into the same place (Lindholmen science park, Gothenburg, Sweden) with the zooming level set to maximum.

Quality (sub)characteristics	Comparison
Interoperability	YM<BM<GM
Compliance	YM=Yes, GM=No, BM=Yes
Security	YM=GM<BM
Learnability	YM=BM=GM
Operability	GM<YM=BM
Availability	GM<YM=BM
Presentation Usability	YM<GM<BM

Table 8. Comparison table. YM=Yahoo Maps, BM=Bing Maps, GM=Google Maps

run-time environment. Mashups are generally composed for temporary needs[38], as a consequence of which mashup requirements may change dramatically at run-time that is also reasonable due to the high dynamic nature of the Web. Therefore, providing support for adaptivity and context-awareness is the major factor ensuring the widespread procreation of mashups [8]. In this regard, [11] proposes a service adaptation approach by combining service capability model and requirement rules, in a sense that changes in run-time environment properties invoke corresponding requirement rules, which is further used to define new service capabilities. After wards, candidate services (mashup components) are clustered, and further ranked in each cluster.

To our best knowledge, less research efforts were conducted with regard to finding and suggesting most relevant mashup components during the development phase, which is also the focus of this work. [3] presents an efficient syntactical approach for searching and ranking relevant candidates for mashup components. It basically relies on clustering component by identifying similar parts in the messages each component provides (input or output). [31] proposes a faceted-based approach for classifying, searching, and ranking mashup components based on utilization and popularity. Their approach is considered superior over traditional Web APIs directories such as ProgrammableWeb specially with respect to tagging system. However, both the mentioned works show a substantial lack of consideration towards other quality characteristics such as security, usability, and data quality which might be of importance when to search for a suitable component. Typically, their works revolve around efficient clustering and ranking of mashup components based on the functionalities they provide.

Our framework takes a further step by relying on a reference quality model entailing a respect towards viewpoints of both the composer and mashup user. The main motivation behind it is the fact that mashup component selection should be goal-oriented and conducted based on evaluation and assessment of the non-functional requirements of one candidate component against another. Accordingly, the frameworks assures that a selected component can fulfill both the requirements of mashup composer and user. Mapping mashup requirements to components respectively allows us to: a) maintain a candidates group of components fulfilling the functional requirements. b) make a subset of the reference quality model containing important quality characteristics based on which the candidate components can be evaluated. It is also of importance that the framework does not explicitly rank candidate components, but rather provides objective measurement-based results on the aspects of importance (quality characteristics of interest), which in turn provides a rationale for objective evaluation. As witnessed in the case-study, despite the fact that Google Maps is so popular and is almost the first choice in most of mapping mashups (according to ProgrammableWeb), following the framework convinced us that Bing Map is a better choice for our specific context of use.

8. CONCLUSION

As the usage of mashups in enterprise environments continuously increases, the expectations from mashups rapidly grow, particularly with respect to properly meeting the user's needs. From the development perspective, the key to building a successful mashup is a proper selection of its building blocks, or in other words mashup components. In the present work, we enhanced the process of mashup component selection by proposing a quality-based framework for measurement-based objective evaluation of candidate components, while aggregating them into an intended mashup solution. The objective evaluation is conducted by utilizing the metrics suite to measure a set of quality characteristics which are deduced from the end-user and composer needs, and are a subset of the reference quality model. Finally, we applied the framework on a case study which is a mashup used as a monitor-

ing tool in emergency response. The result could bias our decision-making regarding selecting a proper map component.

Though the presented framework can be considered as a new approach in recommendation-based mashup component selection, we believe that it remains some potential areas for future work. First, mapping requirements to components can be formalized with the use of existing technologies such as Aspect-Oriented Requirements Engineering [16]. Second, the current metrics do not cover cost of mashup components. In other words, we assumed mashup components are published under a non-commercial license. Finally, the framework shows the potential to support context-awareness and adaptivity that can be taken into account in future work.

References

[1] ALTINEL, M., BROWN, P., CLINE, S., KARTHA, R., LOUIE, E., MARKL, V., MAU, L., NG, Y.-H., SIMMEN, D., AND SINGH, A. Damia: a data mashup fabric for intranet applications. 1370–1373.

[2] BASILI, V. R., AND WEISS, D. M. A methodology for collecting valid software engineering data. *IEEE Trans. Software Eng. 10*, 6 (1984), 728–738.

[3] BLAKE, M. B., AND NOWLAN, M. F. Predicting service mashup candidates using enhanced syntactical message management. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 229–236.

[4] BOLCHINI, D., AND MYLOPOULOS, J. From task-oriented to goal-oriented web requirements analysis. *Web Information Systems Engineering, International Conference on 0* (2003), 166.

[5] BOZZON, A., BRAMBILLA, M., FACCA, F. M., AND CARUGHU, G. T. A conceptual modeling approach to business service mashup development. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 751–758.

[6] CAPPIELLO, C., DANIEL, F., AND MATERA, M. A quality model for mashup components. In *ICWE '9: Proceedings of the 9th International Conference on Web Engineering* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 236–250.

[7] CHEUNG, K., YIP, K., TOWNSEND, J., AND SCOTCH, M. Hcls 2.0/3.0: Health care and life sciences data mashup using web 2.0/3.0. *Journal of Biomedical Informatics 41*, 5 (October 2008), 694–705.

[8] DANIEL, F., AND MATERA, M. Mashing up context-aware web applications: A component-based development approach. In *WISE '08: Proceedings of the 9th international conference on Web Information Systems Engineering* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 250–263.

[9] DANIEL, F., YU, J., BENATALLAH, B., CASATI, F., MATERA, M., AND SAINT-PAUL, R. Understanding ui integration: A survey of problems, technologies, and opportunities. *IEEE Internet Computing 11*, 3 (2007), 59–66.

[10] DIFFERDING, C., HOISL, B., AND LOTT, C. M. Technology package for the goal question metric paradigm, 1996.

[11] DORN, C., SCHALL, D., AND DUSTDAR, S. Context-aware adaptive service mashups. In *IEEE Asia-Pacific Services Computing Conference* (2009).

[12] EASTERBROOK, S., SINGER, J., STOREY, M.-A., AND DAMIAN, D. Selecting empirical methods for software engineering research. 2008, pp. 285–311.

[13] ESCALONA, M., AND KOCH, N. Requirements engineering for web applications - a comparative study. *Journal of Web Engineering 2*, 3 (2004), 193–212.

[14] GARRETT, J. J. Ajax: A new approach to web applications.

[15] GOBLE, C., AND STEVENS, R. State of the nation in data integration for bioinformatics. *J. of Biomedical Informatics 41*, 5 (2008), 687–693.

[16] GRUNDY, J. C. Aspect-oriented requirements engineering for component-based software systems. 84–91.

[17] HOYER, V., AND FISCHER, M. Market overview of enterprise mashup tools. 708–721.

[18] HOYER, V., STANOESVKA-SLABEVA, K., JANNER, T., AND SCHROTH, C. Enterprise mashups: Design principles towards the long tail of user needs. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 601–602.

[19] ISO/IEC. Iso/iec 9126-1 software engineering. product quality - part 1: Quality model. Tech. rep., 2001.

[20] ISO/IEC. Iso/iec 9126-2 software engineering. product quality - part 1: Internal metrics. Tech. rep., ISO, 2001.

[21] ISO/IEC. Iso/iec 9126-3 software engineering. product quality - part 1: External metrics. Tech. rep., ISO, 2001.

[22] ISO/IEC. Iso/iec 9126-4 software engineering. product quality - part 1: Quality in use. Tech. rep., ISO, 2001.

[23] LIANG, XIN; LIU, Y. Z. L. A component-based approach to developing thematic mashups. In *20th*

Australian Software Engineering Conference: ASWEC 2009 (2009), A. Long, Brad ; MacDonald, Ed., pp. 40–49.

[24] LIU, X., HUI, Y., SUN, W., AND LIANG, H. Towards service composition based on mashup. *Services, IEEE Congress on 0* (2007), 332–339.

[25] MAXIMILIEN, M. E., RANABAHU, A., AND GOMADAM, K. An online platform for web apis and service mashups. *IEEE Internet Computing 12*, 5 (2008), 32–43.

[26] MELTZOFF, J. *Critical Thinking About Research: Psychology and Related Fields*. American Psychological Association, 1998.

[27] MERRILL, D. Mashups: The new breed of web app. an introduction to mashups. Tech. rep., IBM developerWorks, http://www.ibm.com/developerworks/xml/library/x-mashups.html, 2006.

[28] MILLS, E. E., MILLS, E. E., AND SHINGLER, K. H. Software metrics - sei curriculum module sei-cm-12-1.1, 1988.

[29] MURUGESAN, S. Understanding web 2.0. *IT Professional 9*, 4 (2007), 34–41.

[30] PAUTASSO, C. Bpel for rest. 2008, pp. 278–293.

[31] RANABAHU, A., NAGARAJAN, M., SHETH, A. P., AND VERMA, K. A faceted classification based approach to search and rank web apis. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 177–184.

[32] ROSENBERG, F., CURBERA, F., DUFTLER, M. J., AND KHALAF, R. Composing restful services and collaborative workflows: A lightweight approach. *Internet Computing, IEEE 12*, 5 (2008), 24–31.

[33] SHAW, M. Writing good software engineering research papers. In *25th International Conference on Software Engineering, 2003.* (2003), IEEE, pp. 726–736.

[34] TANASESCU, V., GUGLIOTTA, A., DOMINGUE, J., VILLARÍAS, L. G., DAVIES, R., ROWLATT, M., AND RICHARDSON, M. A semantic web gis based emergency management system *.

[35] WANG, W., ZENG, G., ZHANG, D., HUANG, Y., QIU, Y., AND WANG, X. An intelligent ontology and bayesian network based semantic mashup for tourism. In *SERVICES '08: Proceedings of the 2008 IEEE Congress on Services - Part I* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 128–135.

[36] WANG, Z., H, M., AND LIN, Z. An open community approach to emergency information services during a disaster. *Information Science and Engieering, International Symposium on 1* (2008), 649–654.

[37] YANG, F. Enterprise mashup composite service in soa user profile use case realization. In *SERVICES '08: Proceedings of the 2008 IEEE Congress on Services - Part I* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 97–98.

[38] YU, J., BENATALLAH, B., CASATI, F., AND DANIEL, F. Understanding mashup development. *IEEE Internet Computing 12*, 5 (2008), 44–52.

[39] YU, J., BENATALLAH, B., CASATI, F., DANIEL, F., MATERA, M., AND SAINT-PAUL, R. Mixup: A development and runtime environment for integration at the presentation layer. 2007, pp. 479–484.

[40] YU, J., BENATALLAH, B., SAINT-PAUL, R., CASATI, F., DANIEL, F., AND MATERA, M. A framework for rapid integration of presentation components. In *WWW '07: Proceedings of the 16th international conference on World Wide Web* (New York, NY, USA, 2007), ACM, pp. 923–932.

[41] YU, S., AND WOODARD, C. J. Innovation in the programmable web: Characterizing the mashup ecosystem. 136–147.

APPENDIX-A

This appendix contains the metrics which are collected from [6], and are intended to be reused in this work. The metrics are mapped to the end-point characteristics of the quality model (see section 2.3). Before presenting the metrics, a number of concepts should be defined:

Protocol: is suggestive of the standard Web protocols, such as REST, and SOAPT, via which the APIs are accessible.

Language: refers to scripting languages through which offered APIs can be used. For instance JavaScript is a well-known scripting language.

Data Format: is the standard data type which is supported by mashup component operations. Examples are XML, and JASON.

- **Interoperability**

Interoperability of component "comp" can be calculated using the following metric

$$Interoperability_{comp} = | P_{comp} | + | L_{comp} | + | DF_{comp} |$$

Where $P_{comp,}$ L_{comp} , DF_{comp} are respectively the protocols, languages, and data formats supported by the component.

- **Security**

Security of a component can be assessed by considering its authentication type and if it is done over Secure Sockets Layers (SSL). Accordingly, the following table can be used to assess the security for a specific component.

Authentication type	Security (no SSL)	Security (over SSL)
No authentication	1	2
API key	2	3
Developer key	3	4
User account	4	5

- **Maturity**

Since mashup components do not provide evidences on their internal performance, maturity can be evaluated by considering the frequency of component updates.

$$\frac{CurrentDate_{Comp} - LastUserDate_{comp}}{\frac{CurrentDate_{comp} - CreationDate_{comp}}{|V_{comp}|}}$$

V_{comp} is the set containing available versions for a specific mashup component.

- **Operability**

Operability of a component consists of three attributes: API operability, data operability, and security operability. The tables below can be used to assess operability.

Protocol	Protocol operability
SOAP/WSDL	1
REST, PHP, Perl, JSP, ASP	2
JavaScript	3

Data format	Data operability
XML	1
JSON, ATOM, RSS, GData	2
Parameters-value pairs	3

Authentication type	Security operability (no SSL)	Security operability (over SSL)
No authentication	5	4
API key	4	3
Developer key	3	2
User account	2	1

- **Completeness**

The ratio between the amount of data retrieved and the amount of data excepted can be used for Completeness assessment.

$$Completeness_{comp} = 1 - (\frac{Number\ of\ Missing\ values}{Total\ Number\ of\ values})$$