Data Integration in Mashups

Giusy Di Lorenzo
Dipartimento di Informatica e Sistemistica
University of Naples Federico II, Via Claudio, 21, 80125 Napoli, Italy
giusy.dilorenzo@unina.it

Hakim Hacid*
Alcatel-Lucent Bell Labs France
Centre de Villarceaux, 91620 Nozay
hakim.hacid@alcatel-lucent.com

Hye-young Paik, Boualem Benatallah
CSE, UNSW
Sydney, NSW 2052, Australia
hpaik@cse.unsw.edu.au
boualem@cse.unsw.edu.au

ABSTRACT

Mashup is a new application development approach that allows users to aggregate multiple services to create a service for a new purpose. Even if the Mashup approach opens new and broader opportunities for data/service consumers, the development process still requires the users to know not only how to write code using programming languages, but also how to use the different Web APIs from different services. In order to solve this problem, there is increasing effort put into developing tools which are designed to support users with little programming knowledge in Mashup applications development. The objective of this study is to analyze the richnesses and weaknesses of the Mashup tools with respect to the data integration aspect.

1. INTRODUCTION

One of the goals of Web 2.0 is to make it easy to create, use, describe, share, and reuse resources on the Web. To achieve that, technologies have flourished around this concept (e.g., blogs, social networks). The capabilities of Web 2.0 are further enhanced by many service providers who expose their applications in two ways: one is to expose application functionalities via Web APIs such as Google Map[1], Amazon.com, or Youtube[2], the other is to expose data feeds such as RSS and ATOM. This opened up new and exciting possibilities for service consumers and providers as it enabled the notion of using these *services*[3] as "ingredients" that can be mixed-and-matched to create new applications.

To achieve this goal, and maybe to anticipate future needs in Web 2.0, a new framework, called *Mashup*, is surfacing [12, 6, 21, 4]. *Mashup* is an application development approach that allows users to aggregate multiple services, each serving its own purpose, to create a service that serves a new purpose. Unlike Web services composition where the focus is on the composition of business (process) services only, the Mashup framework goes further in that it allows more functionalities and can compose heterogeneous resources such as data services, UI services, etc. Applications built using the Mashup technique are referred to as *Mashups* or *Mashup applications*. Their recent proliferation demonstrates that there is high level of interest in the Mashup framework [5, 14, 28, 18, 19]. It also shows that the needs for integrating these rich data and service sources are rapidly increasing. The Mashup approach opens new and broad opportunities for data/services consumers. However, the development investment is still considerable. In fact, a Mashup user needs to know, in addition to how to write code using programming languages (e.g., Java Script, XML/HTML), how to use the different Web APIs[4]. In order to solve this problem, there is an increasing effort put into developing tools which are designed to support users with little programming knowledge in Mashup applications development.

The objective of this study is to analyze the richnesses and weaknesses of the Mashup tools. Thus, we identify the behaviors and characteristics of general Mashup applications and analyze the tools with respect to the data integration aspect. We believe that this kind of study is important to drive future contributions in this emerging area where a lot of research and application fields, such as databases, user machine interaction, etc. can meet. Other studies, focusing on other aspects like process integration, interface integration [11] complement the work presented in this paper to understand different aspects of Mashups.

The remainder of this paper is organised as follows: Section 2 introduces the different levels of Mashups. Section 3 discusses the dimensions of analysis, and describes different Mashup tools according to the considered dimensions[5]. We finish by a discussion and a conclusion in Section 4, summarizing our study and highlighting some future directions.

*This work has been mainly done when the author was a Research Associate at UNSW.
[1]http://maps.google.com/
[2]http://youtube.com
[3]These services can be a data service, such as news, or a process/operation service such as placing an order to Amazon.com.

[4]A lot of APIs are available on the Web: http://www.programmableweb.com/apis/directory/
[5]We have selected these tools not because they are the best or the worst tools, but we have tried to consider as much representative tools as possible.

2. DESCRIPTION OF MASHUP LEVELS

One of the most important characteristics of the Web is certainly its heterogeneity. This heterogeneity can be seen on data, processes, and even user interfaces. Conceptually, a Mashup application is a Web application that combines information and services from multiple sources on the Web. Generally, web applications are developed using the *Model-View-Controller* pattern (MVC)[25] which allows the separation of core business model from the presentation and control logic which use the business model. In the MVC pattern, the *model* represents the data on which the application operates and the business rules used to manipulate the data. The model is independent of the view and the controller. It passively supplies its services and data to the other layers of the application. The *view* represents the output of the application. It specifies how the data, accessed through the model, is presented to the user. Also, it has to maintain its presentation when the model changes. Finally, the *controller* represents the interface between the model and the view. It translates interactions with the view into actions to be performed on the model.

A Mashup application includes all the three components of the MVC pattern. According to Maximilien et al. [23], the three major components of a Mashup application are (1) data level, (2) process level, and (3) presentation level. Moreover, each data source needs to be first analyzed and modeled in order to perform the required actions of retrieval and pre-processing. For the completeness of the study, we first describe those levels. We will then focus our analysis specifically on the data level.

1. *Data Level*: this level mainly concerns data mediation and integration. Challenges at this level involve accessing and integrating data residing in multiple and heterogeneous sources such as web data and enterprise data[17]. Generally, these resources are accessible through REST[15] or SOAP[6] web services, HTTP protocol and XML RPC. Regarding the data mediation, the basic problem comes from structural and semantics diversities of the schema to be merged[17][7]. Finally, data sources can be either structured for which a well defined data model is available (e.g., XML-based document, RSS/ATOM feed), or unstructured (e.g., audio, email text, office documents). In the latter case, the unstructured data needs to be pre-processed in order to extract their meaning and create structured data. So, this level consists of all possible data manipulations (conversion, filtering, format transformation, combination, etc.) needed to integrate different data sources. Each manipulation could be done by analyzing both syntax and semantics requirements.

2. *Process Level*: the integration at the process level has been studied specially in the workflow and service oriented composition areas [13][3]. At the *process level*, the choreography between the involved applications is defined. The integration is done at the application layer and the composed process is developed by combining activities, generally exposed through APIs. The composition is then described using either programming languages such as Java, or dedicated workflow languages such as WS-BPEL [3]. In the Mashups context, those languages are not enough for modeling applications since, for instance, they do not provide the connection to different remote resources, e.g., REST resources, and do not handle the interaction with the client browsers. These limitations make it difficult to directly use these technologies for Mashups. Thus, they need to be adapted in order to model and describe interactive and asynchronous processes. Currently, languages like *Bite*[10] or *Swashup*[23] have been proposed to describe the interaction and the composition model for Mashups.

3. *Presentation Level*: every application needs an interface to interact with the users, and a Mashup application is not an exception. Presentation Level (or User Interface) in Mashup applications is used to elicit user information as well as to display intermittent and final process information to the user. The technologies used to display the result to the user can be as simple as an HTML page, or a more complex web page developed with Ajax, Java Script, etc. The languages for integrating UI components and visualising the front-ends support server-side or client-side Mashups[1][2]. In a server-side Mashup, the integration of data and services is made on the server. The server acts as a proxy between the Mashup application and other services involved in the application. On the other hand, a client-side Mashup integrates data and services on the client. For example, in a client-side Mashup, an Ajax application will do the required composition and parse it into a client's web browser. Currently, the integration at the presentation level in Mashups is done manually. That is, a developer needs to combine the user interface of the wished components using either server-side or client-side technologies. This area is an emerging area and lot of efforts are made in this direction [11][29]. From the Mashup point of view, there is still a lot of work to be done.

In the next section, we introduce the dimensions used in our analysis. As mentioned before, we focus on the data level in Mashups. Other studies focusing on different aspects like presentation level (e.g.,[11]) are necessary and complement our study.

3. ANALYSIS DIMENSIONS

To understand why some automatic support is needed to create Mashups, we give the following example. Suppose that a user wants to implement a *News Mashup* that lets her select news on a news service like CNN International and display both the list of the news and a map that highlights the locations associated with the news; she typically needs to do a lot of programming which involves fetching and integrating heterogeneous data. In fact, the user needs to know not only how to write the code, but also (i) to understand the available API services in order to invoke them and fetch the data output; (ii) to implement screen scraping techniques for the services that do not provide APIs, and (iii) to know the data structure of the input and output of each service in order to implement a data mediation solution.

[6]Simple Object Access Protocol, http://www.w3.org/TR/soap/

The Mashup tools provide facilities to help the user solve some of the above mentioned problems. The analysis provided in this section aims at studying how the tools (see Section 3.1 for the list of concerned tools) address the data mediation problems discussed in the previous section. We will be asking questions such as: How the tools handle data? What kind of processing is performed on the data? What is the output of Mashups? Which operators are provided for data transformation and for creating the data flow? What are the types of data supported by the available operators? etc.

3.1 Analyzed Tools

Currently a number of Mashup tools exist. We have selected only the following tools for three main reasons: (i) these tools were the most popular ones when this analysis was performed (judging from discussions on forums, blogs, etc.), (ii) some other tools have not been reported because of their unavailability at certain stages of the study preventing us to experiment and report results according to our analysis. Finally, (iii) this limitation is motivated by the fact that our objective is not to analyze all the tools but to give a view on the current state of these tools and understand their general approach to data integration.

1. *Damia[5, 27]* is a Mashup tool provided by *IBM*. It allows the users to assemble data feeds from Internet and enterprise data sources. This tool focuses on data feed aggregation and transformation in enterprise environments. Additional tools or technologies like *QEDWiki*[7] and feed readers, that consume Atom and RSS, can be used at the presentation layer for the data feed provided by *Damia*.

2. *Yahoo pipes*[8] is a web-based tool provided by *Yahoo*. The users can build mashup applications by aggregating and manipulating data from web feeds, web pages, and other services. A pipe is composed of one or more modules, each one performing a single task like retrieving feeds from a web source, filter, sort or merge feeds. The output from pipes can be either accessed by a client via a unique URL as RSS or JSON, or visualised on the Yahoo Map.

3. *Popfly*[9] is a web-based Mashup application by Microsoft. It allows the users to create a Mashup combining data and media sources. The Mashup is built by connecting *blocks*. Each block is associated to a service like "Flicker"[10] and exposes one or more functionalities. *Popfly* is much more about data visualization than data manipulation as we will see later, so the "mashed" data can be only visualized using the provided visualization tool.

4. *Google Mashup Editor*(GME)[11] is a Mashup development, deployment and distribution environment by *Google*. The Mashup can be created using technologies like HTML, Java Script, CSS along with the GME XML tags and Java Script API that further allows a user to customize the presentation of the Mashup output.

5. *Exhibit*[18] is a framework for creating web pages containing dynamic and rich visualizations of structured data. Generally, it is used in combination with Babel[12]. It allows users to assemble data obtained in different format such as RDF/XML, N3, Bibtex. *Exhibit* visualises the mashup output on HTML pages, but it also lets the user access the data by explicitly exporting it to various formats including RDF/XML and Exhibit JSON.

6. *Apatar*[13] is a Mashup data integration tool that helps users integrate desktop data with the web. Users install a visual job designer application to create integration schemas called DataMaps. *Apatar*, like *DAMIA*, mainly aims to aggregate and manipulate data that can be reused from other applications, so additional tools that consume the *Apatar* output formats can be used as the presentation layer.

7. *MashMaker*[14] is a web-based tool by *Intel* for editing, querying and manipulating web data. *MashMaker* is different from the other tools in that it works directly on web pages. In fact, *MashMaker* allows users to create a mashup by browsing and combining different web pages. The final goal of this tool is to suggest to the user some enhancements (mashups or widgets), if available, for the visited web pages.

For the analysis, we chose eight dimensions covering various aspects of the data level concerns. Due to space limitations, we only discuss some of the above tools for each dimension for illustration purposes. A summary of the complete analysis is given in Table 1 and a detailed description is available in [22]. The eight dimensions are described in the following.

3.2 Data Formats and Access

In a Mashup application, a user can integrate data described in different formats. For example, web feed format is used to publish frequently updated content such as blog entries, news and so on; tabular format is suitable for describing table-based data models such as csv files or spreadsheets; markup-based format (e.g., HTML and XML) is, of course, commonly used to publish data; multimedia content such as video, audio and images are becoming increasingly prevalent. These types of data can be available to the user from different data sources. The most common data sources can be traditional database systems, local files that are available in the owner's file system, web pages, web services and web applications. To facilitate Web data retrieval, providers often expose their content through web APIs. APIs can be also seen as a useful means for data and application mediation. Here we consider the role of APIs from the data integration point of view in the sense that they offer specific types and formats of data. It should be noted that an API can offer several formats of data, e.g., csv, xml, etc.

[7] http://services.alphaworks.ibm.com/qedwiki/
[8] http://pipes.yahoo.com/pipes/
[9] *http : //www.popfly.net/*
[10] www.flickr.com/
[11] http://code.google.com/gme/index.html
[12] http://simile.mit.edu/babel/
[13] www.apatar.com/

Table 1: Summary of the considered dimensions. (+) means the dimension (i.e. functionality) is provided, (-) means the dimension is not provided. These marks do not imply any "positive" or "negative" information about the tools except the presence or the absence of the considered dimension.

		Damia	Yahoo Pipes	MS Popfly	GME	Exhibit	Apatar	MashMaker
Data Formats/Access	Protocol[a]	P_2	P_2	P_2 ,P_3	P_2	P_2	P_1, P_2	P_1
	Data Format[b]	D_1, D_2 D_7, D_8	D_1, D_2, D_3, D_4, D_5, D_6, D_8, D_9	D_1, D_2, D_9, D_{10}	D_2, D_3, D_4	D_1, D_4, D_6 D_7, D_8, D_9	D_1, D_2, D_6 D_7, D_9	D_5
	Database [c]	DB_1, DB_2[d]	-	-	-	-	DB_3, DB_4, DB_5	-
Internal Data Model	XML-based	+	+	-	+	-	-	+
	Object-based	-	-	+	-	+	+	-
Data Mapping	Manual	+	-	+	-	-	+	+
	Semi-Automatic	-	+	-	-	-	-	-
	Automatic[e]	-	-	+	+	+	-	-
Data Refresh	Pull strategy	+	+	+	+	+	+	+
	Push strategy	-	-	-	-	-	-	-
	Global pull interval	-	+	+	+	+	+	+
	Local pull interval	+	-	-	-	-	-	-
	Interval Setting	+	-	-	-	-	-	-
Mashup's Output	Machine oriented	+	+	-	-	+	+	-
	Human oriented	-	+	+	+	+	-	+
Extensibility	Components	+	+	+	+	+	+	+
	Data	-	-	-	-	-	-	+
Sharing	Total	+	+	+	+	-	+	+
	Partial	-	+	-	-	-	+	-
	No thing	+	+	+	+	-	+	+
	Read only	+	+	+	+	-	+	+
	Read/Write	-	-	-	+	-	-	-
	All users	+	+	+	+	-	+	+
	Groups	-	-	-	+	-	-	-
	Particular user	+	+	+	+	-	+	+

[a] P_1 = HTTP; P_2 = REST; P_3 = SOAP
[b] D_1 = XML; D_2 = RSS; D_3 = ATOM; D_4 = JSON; D_5 = HTML; D_6 = CSV; D_7 = XLS; D_8 = RDF; D_9 = Image; D_{10} = Video
[c] DB_1 = Microsoft Access; DB_2 = DB2; DB_3 = MySQL; DB_4 = Oracle; DB_6 =PostreSQL
[d] These data sources are supported only if DAMIA is used in combination with Mashup Hub
[e] This is considered true if the source data has the same data model as the internal data model.

Murugesan [24] defines an API as an interface provided by an application that lets users interact with or respond to data or service requests from other programs, applications, or web sites. Thus, APIs facilitate the integration between several applications by allowing data retrieval and data exchange between applications. APIs help the developers access and consume resources without focusing on their internal organization. Simple and well-known examples of APIs include Open DataBase Connectivity (ODBC) and Java DataBase Connectivity (JDBC). On the Web, providers like Microsoft, Google, eBay, and Yahoo offer web APIs for retrieving content from their web sites. Such APIs generally use standard protocols such as REST/SOAP web services, AJAX (Asynchronous Javascript + XML) or XML RPC. APIs can also be used to access resources which are not URL addressable such as private or enterprise data [20]. However, some common data sources do not expose their contents through APIs. So, other techniques as screen scraping are needed to extract information.

3.3 Internal Data Model

As stated before, the objective of a Mashup application is to combine different resources, data in our case, to produce a new application. These resources come generally from different sources, are in different formats, and vehicle different semantics. To support this, each Mashup tool uses an internal data model. An internal data model is a single global schema that represents a unified view of the data [7]. A Mashup tool's internal data model can be either (i) *Graph-based* or (ii) *Object-based*.

In a *Graph-based model*, the graph refers to the model based on XML and those consumed as they are (i.e., XML). This can include pure XML, RDF, RSS, etc. Most of the Mashup applications use a Graph-based model as an internal data model. This is certainly motivated by the fact that most of

today's data available on the web are in this format and also, most of the Mashup tools are available via the Web. That is, all the data that are used by the Mashup tools, in this category, transform the input data into an XML representation before processing it. For example, *Damia* translates the data into tuples of sequences of XML data [5]. In an *Object-based model*, the internal data is in the form of objects (in the classical sense of the object-oriented programming). An object is an instance of a class which defines the features of an element, including the element's characteristics (its attributes, fields or properties) and the element's behaviors (methods). It should be noted that in this case, there is no explicit transformation, performed by the tool, like in the case of the graph-based model, but the programmer needs to define the structure of the object according to her data.

To illustrate the differences in the internal data models, let us consider the example of Figure 1 which shows an extract of a spreadsheet (or csv) file containing information about the national parks in the world[14].

title	link	description	pubDate
Grand Canyon National Park	www.grand.canyon.national -park.com /	Grand Canyon National Park...	19/03/2008
Big Bend National Park	http:// www.big.bend.national -park.com /	Big Bend National Park...	19/03/2008
Gulf Islands National Seashore	http:// www.hikercentral.com /parks/ guis /	Gulf Islands National Seashore.....	19/03/2008

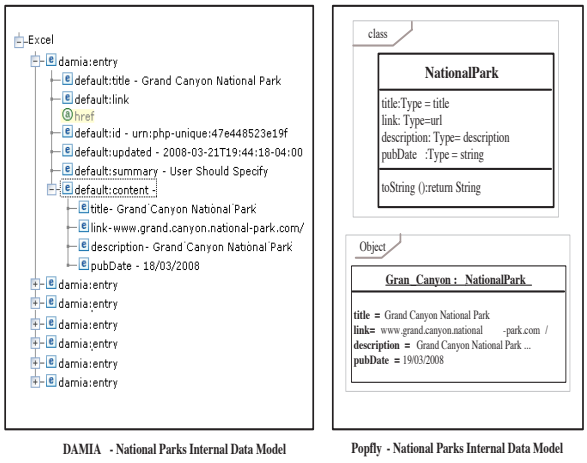
Figure 1: National Parks Data Source

The illustrated data in Figure 1 is given as an input to two different tools, namely *Damia* and *Popfly* and the obtained result is shown in Figure 2. Figure 2(a) shows the translation operated by *Damia* on the input data. That is, each row in the *csv* file is transformed to an XML representation contained in the element $< damia : entry >$. Each entry is composed of some elements containing general information regarding the data source such as file name (i.e., $< default : id >$), last updating (i.e., $< default : update >$) and by the $< default : content >$ element in which the national parks information is stored. Figure 2(b) shows how the data could be represented using an object-based notation is the case of *Popfly*.

3.4 Data Mapping

To instantiate an internal data model from an external data source, the Mashup tools must provide strategies to specify the correspondences between their internal data model and the desired data sources. This is achieved by means of data mapping. Data mapping is the process needed to identify the correspondences between the elements of the source data model and the internal data model [26]. Generally speaking, a data mapping can be: (i) *manual*, where all the correspondences between the internal data model and the source data model are manually specified, one by one, by the application designer. In this case, the tool should then provide some facilities for the user to design the transformation. (ii) *Semi-automatic*, where the system exploits some meta-data (e.g., fields names and types) to propose some possible mapping configurations. However, the user needs to confirm these alternatives and, usually, correct some of them. At this stage,

[14]Some elements are omitted for readability matters



DAMIA - National Parks Internal Data Model

Popfly - National Parks Internal Data Model

Figure 2: Representation of the National Park *csv* file in DAMIA and Popfly

only *Yahoo Pipe* supports the semi-automatic mapping, offering some hints for the user about possible mappings. (iii) *Automatic*, where all the correspondences between the two data models are automatically generated, without user intervention [26]. This is a challenging issue in the data integration area. Since the Mashup area is in its "early stage", this type of mapping is not supported by any Mashup tool. It should be noted that the mapping process may require an intermediary step, i.e., a wrapping step, in order to transform the source format to the internal format, e.g., from csv to XML. It is also interesting to point out that the mapping in the currently available Mashup tools is only done at schema level, while no semantic information is being considered so far. For instance, Semantic Web languages such as RDF[15] and OWL[16] can be exploited to manage the semantic part. However, this will need more investigation since it will introduce other complications due especially to the targeted users.

3.5 Data Flow Operators

Data flow operators allow performing operations either on the structure of the data (similar to the data definition language/operators in the relational model), or on the data (content) itself (similar to the data manipulation language/ operator in the relational model).

More concretely, data flow operators support: (i) restructuring of the schema of the incoming data, e.g., adding new elements, adding new attributes to elements; (ii) elaborating on a data set such as extracting a particular piece of information, combining specific elements that meet a given condition, change the value of some elements; (iii) building a new data set from other data sets such as merging, joining or aggregating data (similar to the concept of *views* in databases).

The implementation of the data flow operators depends strongly on the main objective of the tool, i.e., integration or visual-

[15]http://www.w3.org/RDF/

[16]http://www.w3.org/TR/owl-features/

ization. Some operators, e.g., *Union*, are implemented in different tools, e.g., *Damia* and *MashMaker*, but the attached interpretation is also different, e.g., a materialized union of two data sets in *Damia* and a virtual union of two Web pages in *MashMaker* (virtual in the sense that the schemas associated to the source pages are not altered with the new page). The main data integration oriented operators, implemented in several tools are the following: *Union, Join, Filter,* and *Sort*. We give hereafter a general description of these operators. A more detailed discussion of all the operators of all the considered tools is given in [22].

Table 2: Main and common operators

Operator	Description
Union	Combines two data sets in one containing all the data from the participating sets.
Join	Combines different data sets according to a condition.
Filter	Selects a specific subset (entities and attributes) from an original subset.
Sort	Presents the selected data in a specific order.

3.6 Data Refresh

In some cases, e.g., stock market, data is generated and updated continuously. Various strategic decisions, especially in enterprises, are generally taken according to the last status/values of the data. It is then important that a system propagates the updates of the data sources to the concerned user(s). There are two strategies dealing with the status of the data in the source, depending on the objective of the user: (i)*pull strategy* and (ii)*push strategy*[8]. The pull strategy is based on frequent and repetitive requests from the client. The pulling frequency is set to be lower than the average update frequency of the data in the source itself. The freshness of the data depends on the pulling frequency, i.e., the higher the pulling frequency, the fresher the data and vice-versa. One of the main disadvantages of a high refresh frequency is that unnecessary requests may be generated to the server. In the push strategy, the client does not send requests but needs to register to the server. The registration is necessary to specify/identify the data of interest. Consequently, the server broadcasts data to the client when a change occurs on the server side. The main disadvantage of this model is that the client can be busy performing other tasks when the information is sent which implies a delay in its processing.

Another important parameter to point out here is the way the tools manage the pull interval. We can define two possible strategies to handle this issue: a *global strategy* and a *local strategy*. For the *Global strategy*, the pull interval is set for the whole application. This supposes that the data sources have the same updating interval. That is, the data sources are requested at the same time interval, corresponding to the one of the Mashup tool. As a result, the user keeps better track of some sources (the ones having low refresh interval compared to the defined one) than the others (the ones having high refresh interval compared to the defined one). In the *Local strategy*, each data source is given its own refresh interval. This pull interval is supposed to correspond to the one of the data source refresh itself. As a result, a better trace is kept of each data source. From the tool's point of view, only *Damia* allows the users to define the pull interval and to handle it with the *Local strategy*. In fact, to set the pull interval, each source component has the *Refresh Interval* parameter. After the time has exceeded, the data from the specified URL is reloaded.

3.7 Mashup Output

We consider the output as a dimension in this study since a user might be interested in exporting her Mashup (the data flow) result in another format in order to reuse it or to process it with another particular application (e.g., spreadsheet) for further processing instead of visualizing it. That is, we can distinguish two main output categories: *Human oriented output* and *application oriented output*. In the *Human oriented output*, the output is targeted for human interpretation, e.g., a visualization on a map, on an HTML page, etc. That is, for this category, the output can be considered as the "final product" of the whole process. For the *processing oriented output*, the output is mainly targeted for machine processing. This is interesting in the case where the considered data needs to be further processed for, e.g., a knowledge discovery process. It should be noted that this category can, at some stage, include the first category, e.g., an RSS output can be at the same time visualized on an HTML page and also can be used by other applications for other processing tasks.

3.8 Extensibility

Extensibility defines the ability of the tool to support additional, generally user defined, functionalities. There can be two possible ways to define and use these functionalities. A functionality can be either (i) embedded inside the tool, i.e., the corresponding code of that functionality is added to the tool using a specific programming language, or (ii) external, i.e., invoking the corresponding service containing such function. Extensibility depends mainly on the architecture and the spirit of the tool. In some cases, the extension can be done by embedding the code of the desired functionality in the tool (e.g., *Popfly*); in other cases, services are invoked like REST services, SOAP, etc. (e.g., *Pipes*). In addition, this feature is managed differently by the different tools. In fact, in one case, the added function/service is shared with the whole community that uses the tool (e.g., *Popfly*). In the other case, the extension is visible only for the specific user (e.g., *Pipes*).

3.9 Sharing

Mashups are based on the emerging technologies of the Web 2.0 in which people can create, annotate, and share information in an easy way. Enabling security and privacy for information sharing in this huge network is certainly a big challenge. This task is made more difficult especially since the targeted public with the Web 2.0 is, or supposed to be, a general public and not expert in computing or security. This dimension defines the modality that the tool offers to enable resources sharing by guaranteeing privacy and security in the created Mashup applications. This is a challenging area in the current Mashup and a lot of work remains to be done. This dimension includes the following three indicators: 1) **What** is shared in the Mashup?, 2) **How** is this shared? and 3)**Who** are the users with whom this

(the shared resource(s)) is shared with? For the **What**, the shared resource can be *total*, *partial*, or *nothing*. The shared resource can be given different rights such as read only (user can read all entries but cannot write any entry), read/write (user can read and write all entries in the data), no access (user cannot read or write any entries). The **Who** as for it can be **All people**, **Group**, or **particular User**. Notice that for each member, different sharing policies (what and how) can be specified and applied.

For example, *GME* and *Yahoo Pipes* allow implementing sharing policies. In *GME*, the sharing policy can be: (i) total, i.e., read access to source code, data and output. (ii) Partial, i.e., read access to source code. (iii) Nothing, where the Mashup is not shared. When a Mashup is shared in GME, for the data used to build the application, the designer can decide to share it with a group or with all users by specifying Read/Write policies. In *Yahoo Pipes*, if a private element is used (Private string or Private text input) the code of the shared Mashup is available as well as the Mashup output but it is not possible to visualize the intermediate outputs.

4. DISCUSSION AND CONCLUSION

In this section, we make a general discussion on the tools by considering their advantages and disadvantages. We aim, at the same time, to give possible points to consider for further improvements.

Mashup tools are mainly designed to handle Web data. This can be seen as an advantage, but an inconvenience at the same time. In fact, it is an advantage since it offers access and management of some data available only on the Web, e.g., RSS feeds. To access these Web data, the tools support the two most used protocols for exposing APIs, i.e., REST and SOAP protocol. This is a consequence of the success, utility and the popularity of these protocols. A disadvantage is that data available on desktops can not be accessed and used the same way. This is a considerable disadvantage since users bring a lot of data onto their desktops for cleaning, manipulation, etc. There is a lot of work done to help the user put and manage data on the web [16], but since this is not completely adopted, supporting local data on a user's desktop should be considered.

The majority of the tools have an internal data model based on XML. This design choice is motivated by the fact that the data available on the web is mainly exposed in an XML format. Also, the communication protocols for the data exchanging over the network use generally XML messages. The other dominant internal data model in Mashup tools is object based. This data model is much more flexible to use, even if more programming is required to implement operations on it especially for programmers. This diversity can explain their targeted/origin community. In fact, XML is much more for databases community where as object is for applications and development community.

To manage data, the tools make available only a small set of operators for data integration and manipulation. The set of provided operators is usually designed based on the main goal of the tool. For example, if the tool is visualization oriented, only few operators for data elaboration such as filtering and sorting are available. In addition, the offered operators are not easy to use, at least from a naive user point of view. Also, the tools do not offer powerful expressiveness since they allow expressing only simple operations, e.g., simple joins, and can't be used to express more complicated queries such as joins with conditions, division, etc. This means that, from the expressiveness point of view, these tools are far from reaching the database languages, i.e., integration languages, such as SQL.

None of the analyzed tools implement a *Push strategy* for the data refreshing and the reason is that the majority of the currently available APIs are REST based. The style of the REST protocol requires all communications between the browser and the server to be initiated by the client and no support is offered to maintain the state of the connection [9]. All the analyzed tools use a *Pull strategy* for data freshness handling. This can be motivated also by the fact that the tools providers wish to control (or prevent) the overloading of their servers. In addition, they implement a *Global strategy* for the pull interval setting. This strategy however does not allow developing applications in which processed data are characterized by a high refresh frequency, since it is not possible to explicitly specify the refresh rate for each source.

One of the main goals of Web 2.0 technologies is the creation, the reuse, the annotation and the sharing of web resources in an easy way. Based on these ideas, Mashup tools are all extensible in the sense that new operators, and in some cases data schemas, can be developed and invoked or/and plugged inside the tools. However, at this stage, the majority of tools do not support the reuse of the created Mashups. This feature could allow developing complex applications by integrating the results of different Mashups (also built with different Mashup tools). Some tools start to consider this issue such as Potluck[19][17] which can use the Exhibit output. However, this is a limited cooperation between tools. This is a very important point especially that the tools have a several limitations and a user can't express his wishes using only one tool.

The current development of Mashup tools is mainly focused on offering features to access, manage, and present data. Less consideration has instead been given to the issue of data sharing and security so far. The security criterion needs to be taken into account inside the tools since communication problems could make a Mashup perform too many requests to source data servers, causing overloads for those servers. At this time, only *Intel Mashmaker* takes into account this problem applying some performance restrictions on the Mashup application[14].

Also, all the analyzed tools are server side applications, meaning that both the created Mashup and the data involved in it are hosted on a server owned by the tool provider. Therefore, the tool provider has the total control on the Mashup and if a user wants to build an application containing that Mashup, the dependability attributes of that application cannot be properly evaluated. In addition, from the performance point of view, no tools provide information regarding the analysis of the performances and, in particu-

[17]This tool is not discussed in this paper for some of the reasons introduced in the beginning of this paper.

lar, information regarding the evaluation of scalability. That information is needed to know the capability of a system to handle a growing amount of data and the user requests.

Finally, all the tools are supposed to target "non-expert" users, but a programming knowledge is usually required. Some tools require considerable programming effort since the whole process needs to be implemented manually using instructions expressed in programming language such as Java Script. Others necessitate medium programming effort given that only some functionalities need to be coded in an explicit way using a programming language; a graphical interface is offered to the user to express most of operations. At this time, there is no tool that requires low or no programming effort by the user to build a Mashup, which is necessary to claim that the tools are targeted for end-users.

5. REFERENCES

[1] *Mashup Styles, Part 1: Server-Side Mashups, http://java.sun.com/ developer/ technicalArticles/J2EE/mashup_1/.*

[2] *Mashup Styles, Part 2: Client-Side Mashups, http://java.sun.com/ developer/ technicalArticles/J2EE/mashup_2/.*

[3] *OASIS: Web Services Business Process Execution Language Version 2.0. (2007), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.*

[4] S. Abiteboul, O. Greenshpan, and T. Milo. Modeling the mashup space. In *WIDM*, pages 87–94, 2008.

[5] M. Altinel, P. Brown, S. Cline, R. Kartha, E. Louie, V. Markl, L. Mau, Y.-H. Ng, D. Simmen, and A. Singh. Damia: a data mashup fabric for intranet applications. In *VLDB '07*, pages 1370–1373. VLDB Endowment, 2007.

[6] S. Amer-Yahia and A. Y. Halevy. What does web 2.0 have to do with databases? In *VLDB*, page 1443, 2007.

[7] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[8] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Disseminating dynamic web data. *IEEE Transactions on Computers*, 51(6):652–668, 2002.

[9] E. Bozdag, A. Mesbah, and A. van Deursen. A comparison of push and pull techniques for ajax. In S. uang and M. D. Penta, editors, *Proceedings of the 9th IEEE WSE*, pages 15–22, 2007.

[10] F. Curbera, M. J. Duftler, R. Khalaf, and D. Lovell. Bite: Workflow composition for the web. In *ICSOC*, pages 94–106, 2007.

[11] F. Daniel, J. Yu, B. Benatallah, F. Casati, M. Matera, and R. Saint-Paul. Understanding ui integration: A survey of problems, technologies, and opportunities. *IEEE Internet Computing*, 11(3):59–66, 2007.

[12] C. Duda, G. Frey, D. Kossmann, and C. Zhou. Ajaxsearch: crawling, indexing and searching web 2.0 applications. *PVLDB*, 1(2):1440–1443, 2008.

[13] S. Dustdar and W. Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, August 2005.

[14] R. Ennals and M. N. Garofalakis. Mashmaker: mashups for the masses. In *SIGMOD*, pages 1116–1118, 2007.

[15] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[16] R. Geambasu, C. Cheung, A. Moshchuk, S. D. Gribble, and H. M. Levy. Organizing and sharing distributed personal web-service data. In *WWW*, pages 755–764, 2008.

[17] A. Halevy. Why your data won't mix. *Queue*, 3(8):50–58, 2005.

[18] D. F. Huynh, D. R. Karger, and R. C. Miller. Exhibit: lightweight structured data publishing. In *WWW '07*, pages 737–746, New York, NY, USA, 2007. ACM.

[19] D. F. Huynh, R. C. Miller, and D. R. Karger. Potluck: Data mash-up tool for casual users. In *ISWC/ASWC*, pages 239–252, 2007.

[20] A. Jhingran. Enterprise information mashups: integrating information, simply. In *VLDB '06*, pages 3–4. VLDB Endowment, 2006.

[21] S. Kinsella, A. Budura, G. Skobeltsyn, S. Michel, J. G. Breslin, and K. Aberer. From web 1.0 to web 2.0 and back -: how did your grandma use to tag? In *WIDM*, pages 79–86, 2008.

[22] G. D. Lorenzo, H. Hacid, H. young Paik, and B. Benatallah. Mashups for data integration: An analysis. Technical Report UNSW-CSE-TR-0810, 2008.

[23] E. M. Maximilien, H. Wilkinson, N. Desai, and S. Tai. A domain-specific language for web apis and services mashups. In *ICSOC '07*, pages 13–26, Berlin, Heidelberg, 2007. Springer-Verlag.

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

[25] W. Pree. *Design patterns for object-oriented software development*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[26] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[27] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh. Damia: data mashups for intranet applications. In *SIGMOD '08*, pages 1171–1182, New York, NY, USA, 2008. ACM.

[28] J. Wong and J. Hong. Marmite: end-user programming for the web. In *CHI '06*, pages 1541–1546, New York, NY, USA, 2006. ACM.

[29] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera. A framework for rapid integration of presentation components. In *WWW '07*, pages 923–932, New York, NY, USA, 2007. ACM.