Mashups for Data Integration: An Analysis

Giusy Di Lorenzo[1] Hakim Hacid[2] Hye-young Paik[2] Boualem Benatallah[2]

[1] Dipartimento di Informatica e Sistemistica
Via Claudio, 21
80125 Napoli, Italy
giusy.dilorenzo@unina.it
[2] School of Computer Science Engineering
University of new south wales
Sydney, NSW 2052, Australia
{hakimh, hpaik, boualem}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-0810
April 2008

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Mashup is a new application development approach that allows users aggregate multiple services, each serving its own purpose, to create a service that serves 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 understand how to write code using languages, but also how to use the different Web APIs from all services.

The objective of this study is to analyze the richnesses and weaknesses of the Mashup tools. In particular, we identify the behaviors and characteristics of general Mashup applications and analyze the tools with respect to the key aspects from the Mashup applications. 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.

1 Introduction

One of the goals of the Web 2.0 is to make it easy to create, use, describe, share, and reuse resources on the Web. To achieve that, a lot of technologies have flourished around this concept, e.g. blogs, social networks, etc. As a consequence and in order to follow the rapid development, a lot of service providers are exposing their applications functionalities via Web APIs such Google Map[1], Amazon.com, and Youtube in one hand. In another hand, a lot of data feeds, such as RSS and ATOM, is is creating a flourish of potential data sources that various applications can tap into. This opened up new and exciting possibilities for service consumers and providers as it enabled the notion of using these *services*[2] as "ingredients" that can be mixed-and-matched to create new applications.

To achieve this goal, and may be to anticipate future needs in the Web 2.0, a new framework, called *Mashup*, is surfacing. *Mashup* is then a new application development approach that allows users aggregate multiple services, each serving its own purpose, to create a service that serves a new purpose. Applications built using the Mashup technique are referred to as *Mashups* or *Mashup applications*, which are built on the idea of reusing and combining existing services, i.e. existing search engines and query services, data, etc.

The interest of the Mashup framework can be noticed in the recent proliferation of Mashup applications showing that the needs for integrating these rich data and service sources are rapidly increasing. Examples of Mashup applications can include Mashups with maps where the objective is to plot various data on a map like Google Map of Yahoo Map[3]; Mashups using multimedia content imported from YouTube, Flickr, etc.; Mashups using e-commerce services such as Amazon.com or Yahoo shopping are also flourishing. Finally, The surging popularity of data feeds (such as RSS or ATOM), the most popular example of Mashups is the feeds Mashups, which subscribe to regular data feeds, typically in RSS or ATOM format, to access data such as news, blogs content, catalog updates, etc.

Although the Mashup approach opens new and broader opportunities for data/service consumers, the development process still requires the users to know, not only understand how to write code using languages (e.g., Java Scripts, XML/HTML, Web services), but also how to use the different Web APIs[4] from all 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 application development.

The objective of this study is to analyze the richnesses and weaknesses of the Mashup tools. In particular, we identify the behaviors and characteristics of general Mashup applications and analyze the tools with respect to the key aspects from the Mashup applications. Through this analysis, we will be asking questions such as how the tools handle data, what kind of processes is made, what is the result of Mashups, etc. We believe that this kind of study is important to drive future contributions in this emerging area where a lot of research

[1] http://maps.google.com/

[2] These services can be a data service, such as news, or a process/operation service such as placing an order to Amazon.com.

[3] http://maps.yahoo.com/

[4] A lot of APIs are available on the Web: http://www.programmableweb.com/apis/directory/

and application fields, such as databases, user machine interaction, etc., can meet.

Focusing on the data integration part of the Mashups, we organize the reminder of this paper as follows: Section 2 introduces the different levels of Mashups. After presenting and illustrating, using examples, the different dimensions that we consider for this study in Section 3, Section 4 discusses the analysis of some Mashup tools. We have selected only some tools since our objective is not to analyze all the tools but to give a view on the current state of these tools. We finish by a discussion and a conclusion in Section 5 summarizing our study.

2 A Study of Mashup Applications

To characterize a Mashup application, let's consider the following example. Suppose that a user wants to build a Mashup application that lets her select news from news provider, like CNN International, and display both the list of the news and a map that highlights the location of that news. To implement this example, three types of component services are used: (i) Map Services, (ii) News Services and (iii) Rss-GeoRss Converter Service. In particular CNN RSS or Repubblica RSS News services for News Component, GeoName Web Services for RSS-GeoRSS Component, and Yahoo or Google Map for Map Component. The GeoName service is needed only if the news feed does not contain information about latitude and longitude, necessary for showing the news on the map. Each service exposes its data in various way: GeoName Web Services and Yahoo Map use REST-based APIs, Google Map uses Javascript API and CNN and Repubblica RSS use RSS feed.

In addition, the exchanged data can be represented using different format such as XML, CSV, RSS/Atom, HTML, JSON, plain text and so on. The Mashup application needs to manage: (i) the integration between the different types of data (data flow); (ii) the communication with the components and interaction among them and finally(iii) the displaying of the content to the end-user. According to Maximilien et all., [4], the three major components of a Mashup application are (1) Data Mediation Level, (2) Process Mediation Level, (3) Presentation Level. Moreover, each data source needs to be first analyzed and modeled in order to perform the required actions of retrieval and preprocessing. In the following, we briefly describe the different components of a Mashup application.

2.1 Data Mediation Level

The *Data Mediation* challenges involve accessing and integrating data residing in multiple and heterogeneous sources such as web data, enterprise data, etc. [16]. Generally, these resources are accessible though REST or SOAP web services, HTTP protocol and XML RPC. Regarding the data mediation, the basic problem to be dealt with when integrating data from heterogeneous sources, come from structural and semantics diversities of the schema to be merged[16][7]. Structural diversity should rise problems like: (1) the same schema element, in two different schema are given different names (e.g. latitude or geo:lat) or use two different data types (e.g. integer or double); (2) same elements in two

different schema are grouped in different ways; (3) one schema can cover some aspect of the domain that are not considered in the others (domain definition).

In addition, sometimes the format of the data can be different, e.g. one is XML-based and another is object-based. For example, if one wants to show some news exposed in RSS format on Google map, first the data needs to be parsed and converted in *geo* W3C[1] format. Second, a data format transformation is needed since the news data needs to be transformed to a Java object, which is the expected input of Google Map API.

Besides, from the semantic point of view, the integration between the data schema should be made in order to consider and preserve the meaning of the data[16]. For instance, two different schema elements in two different data sources can have the same name but the meaning is different and vice-versa. For example, two different schema could have both the $< Location >$ field, but one includes name of city and street and another includes the country name.

Finally, data sources can be either structured for which a well defined data model is available, e.g. XML-based document or RSS/Atom feed, or unstructured data, e.g. audio, video, e-mail text, office documents, etc. In the latter case, the unstructured data needs to be processed in order to extract meaning and create structured data. So, *Data Mediation* consists of all possible data manipulations (conversion, filtering, format transformation, combination, etc.) needed to integrate different data sources, i.e. data flow. Each manipulation could be done by analyzing both syntax and semantics requirements.

2.2 Process Mediation Level

The integration at the application level has been fully studied specially in the workflow and service oriented composition areas[12]. A lot of related problems have been investigated such as the definition of models and languages to describe either the component services and the composition process, e.g. *WS-BPEL*[3]. The *Process Mediation Level* defines the choreography between the involved applications. The integration is done at the application layer and the composed process is developed by combining functions, generally exposed by the services through APIs.

In the Service Oriented Architecture (SOA) area, the composition focuses on the behavioral aggregation of services and the interaction is considered between the resources only[10]. In contrast, since the Mashup applications do not only focus on the data integration but also the connection to different remote data services, for instance REST resources or functions available through Java or Java Script methods, the interaction with the clients browsers needs to be handled. So, the models and languages from the SOA approach must to be adapted in order to model and describe interactive and asynchronous processes.

Currently, languages like *Bite*[10] or *Swashup*[4] have been proposed to describe the interaction and the composition model for the Mashup applications. In particular, *Bite* extends the BPEL's composition model to satisfy the development of the interactive process and the interaction between REST services. Besides, *Swashup* is a domain specific language based on Ruby[2] that allows functionalities such as the description of the interaction model and the invocation of

[1] http://www.w3.org/2003/01/geo/ (visited on 28/04/2008)
[2] http://www.ruby-lang.org/en/ Visited on 28/04/2008

synchronous and asynchronous methods. That is, from process mediation point of view, further research needs to be done to fully take of these challenges in the Mashup area.

2.3 Presentation Level

Every application needs an interface to interact with users and Mashup application is not an exception. Presentation Mediation (or User Interface) in Mashup application is used to elicit user information as well as to display intermittent and final process information to the user. In our News Mashup application example, user initiates the application by choosing a news provider by typing *url* of their RSS feed service. Then, as an output for users, the map with location for the events in the news will be displayed on the Mashup application.

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 used to implement the integration of components UI and the front-ends visualization support server side or client-side Mashup[1][2]. If we use a server-side Mashup in the News Mashup application example, the composition of the components for news and maps takes place in the server, which means that the sever does the parsing from the data type from Mashup application to the data type required for the display on the clients web browser. On the other hand, in a client-side Mashup, Ajax application on client-side will do the required composition and parse it into clients web browser. Sometimes client-side approach cannot provide Mashup for certain components because of the so-called cross-domain problem. The cross-domain problem occurs when client-side, e.g. Ajax application, tries to access data in a different domain name. To avoid this problem, using server-side approach such as ASP or JSP is inevitable.

This area is an emerging area and a lot of efforts are done in this direction[11][20]. From the Mashup point of view, there is still a lot of effort to be done.

3 Tools Analysis

In this the section we provide an in-depth analysis of how the Mashup tools can help the user to fetch and integrate heterogeneous data. To understand why some automatic support is needed to create a Mashup application, we give an example. Let us suppose that a user wants to implement the *News Mashup*; he/she typically needs to do a lot of programming which invokes fetching and integrating heterogeneous data. In fact the user needs to know not only how to write the code but also (1) to understand the available API services in order to invoke them and fetch the data output; (2) to implement screen scraping techniques for the services that do not provide APIs, and (3) to know the data structure of the input and output of each service in order to implement data mediation solution.

The Mashup tools provide facilities to help the user to solve some of the above mentioned problems. The analysis provided in this section aims at analyzing how the tools address the data mediation problems discussed in the previous section, we will be asking questions like which facilities do they provide for searching data sources or existing Mashups? which operators do they provide

for data transformation and for creating the data flow? or, which are the types of data supported by the available operators?

The approach used for the comparison of the Mashup tools is based on the analysis of the following eight dimensions, which allow addressing the data mediation problems:

3.1 APIs

Murugesan [19] 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 from/between applications. APIs help to access and consume resources without focusing on internal organization of them; simple and known examples of APIs include ODBC[1] and JDBC[2]. These APIs allow programmers to access data in databases by only sending a query and getting results. The APIs offer then a mean to handle the data by allowing accessing particular items, updating items, removing items, etc.

On the Web, providers like Microsoft[3], Google[4], eBay[5] and Yahoo[6] allow to retrieve content from their web sites by providing web APIs that are generally accessible through standard protocols such as REST/SOAP web services, AJAX (Asynchronous Javascript + XML) or XML Remote Procedure Call.

APIs can also be used to access resources which are not URL addressable such as private or enterprise data. These data are often exposed in many other formats such as office documents, e-mail, database query results, etc. [18]. Using different types of APIs, it is possible to implement an application that combines information from a user's own file and data coming from web APIs. For example, a user may implement an application that combines information regarding insurance policies contained in his own Excel spreadsheet with the web weather service alerts provided by a website [5].

3.2 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 *XML-based* or *Object-based*.

- *XML-based model*: most of the Mashup applications use an XML-based model as an internal data model. This is certainly motivated by the fact that most of todays data, mainly on the Web such as RSS feeds, are available in this format and also, most of the Mashup tools are available via the

[1] Open DataBase Connectivity: http://support.microsoft.com/kb/110093
[2] Java DataBase Connectivity: http://java.sun.com/docs/books/tutorial/jdbc/overview/index.html
[3] http://www.microsoft.com/
[4] http;//www.google.com/
[5] http;//www.ebay.com/
[6] http;//www.yahoo.com/

Web. This model is a graph-based data model used for managing semi-structured data and for integrating heterogeneous data sources. 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].

- *Object-based model*: in this case, the internal data is in the form of objects (in the classical sense of the object-oriented programming). An object is an instance of class which defines the features of an element, including the element's characteristics (its attributes, fields or properties) and the element's behaviors (methods, operations). It should be noted that in this case, there is no explicit transformation, performed by the tool, like in the case of the XML-based model, but the programmer needs to define the structure of the object according to his/her data. *Popfly* is an example of a tool operating using an object-based data model.

To illustrate the differences in the internal data models, let us consider the example of Figure 3.1 which shows an extract of an excel(or csv) file containing the information of the national parks in the world[7].

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

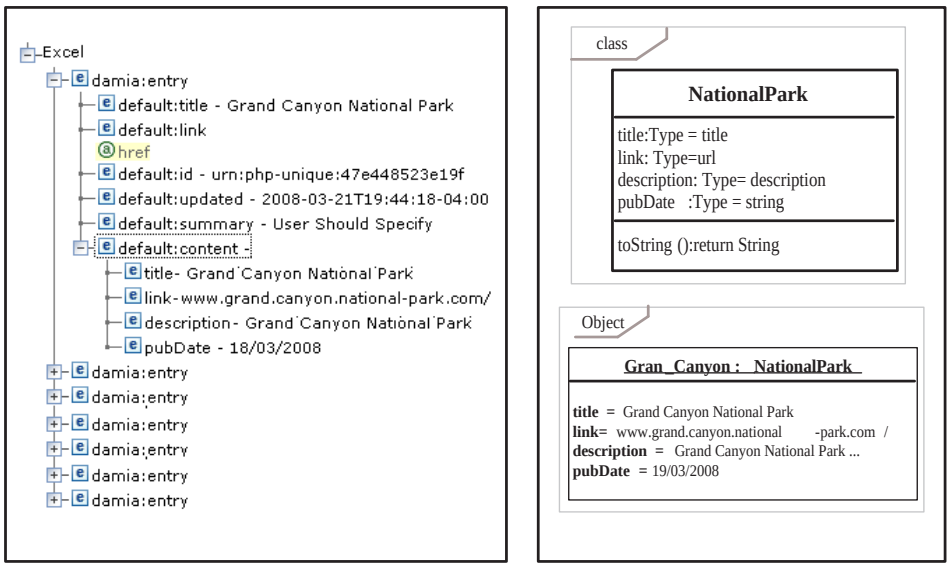
Figure 3.1: National Parks Data Source

The illustrated data in Figure 3.1 is given as an input to two different tools, namely *Damia* and *Popfly* and the obtained result is shown Figure 3.2. Figure 3.2(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 3.2(b) as for it how the data could be represented using an object based notation is the case of *Popfly*.

3.3 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 the mean of data mapping. Data mapping is the process needed to identify the correspondences between the elements of the source data model and the internal

[7]Some elements are omitted for readability



DAMIA - National Parks Internal Data Model **Popfly - National Parks Internal Data Model**

Figure 3.2: Representation of the National Park *csv* file in DAMIA and Popfly

data model [15]. Generally speaking, a data mapping can be manual, semi-automatic, or automatic.

1. *Manual*: in this case, all the correspondences between the internal data model and the source data model are manually specified, one by one, by the application designer. The tool should then provide some facilities for the user to design the transformation as it's the case in *Damia* which provides a transformation operator in which a mapping can be performed.

2. *Semi-Automatic*: in the semi-automatic mapping, the system exploits some metadata (e.g., fields names and types) to propose some possible mapping configurations. However, the user needs to confirm these propositions and, usually, correct some of them. *Yahoo Pipe* is an example of a Mashup tool supporting the semi-automatic mapping, offering some hints for the user about possible mapping.

3. *Automatic*: in this case, all the correspondences between the two data models are automatically generated, without user intervention [15]. 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 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.

3.4 Data Flow Operators

The data flow operators allow to perform 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). Here we consider the operators and the expression languages provided by the tools for processing and integrating data.

More concretely, data flow operators allow: (i)to restructure the schema of the incoming data, e.g. adding new elements, adding new attributes to elements; (ii) to perform elaboration 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) to build a new data set from other data sets such as merging, joining or aggregating data (similar to the concept of *views* in databases).

To illustrate the data flow operators, consider the example where a user wants to implement, in *Damia*, a Mashup that merges two RSS feeds – one containing the weather information for cities in Australia and the other one containing information about the events in the different Australia's cities – on the city name. After getting the necessary data, the following operators need to be applied, as illustrated in Figure 3.3:

1. *Transform* operator to change the structure of the event schema which is done by adding the city field (data transformation). In addition, another operation is needed to populate this filed consisting in extracting the name of the city from the title of items(Data Manipulation).

2. *Filter* operator to select the weather information of the chosen city (Data Manipulation);

3. *Merge* operator to join the previous data sequences – output of filter and transform operator – on the name of the city (Data Manipulation);

4. *Sorting* operator to sort the obtained feeds on the value of date.

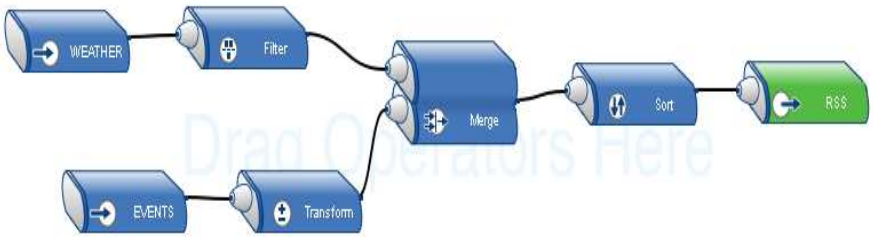


Figure 3.3: Illustration of four data flow operators in Damia

3.5 Data source refresh

In some cases, e.g. stock market, data are generally generated and updated continuously. A lot of strategic decisions, especially in enterprises, are generally made 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 generally two strategies dealing with the status of the data in the data source, depending on the objective of the user: the *pull strategy* and *push strategy*[8].

- *pull strategy*: this strategy is based on frequent and repetitive requests from the client, based on a pulling frequency. This pulling frequency is generally chosen to be lower than the average update frequency of the data in the source. The freshness of the data depends mainly on the pulling frequency, i.e. higher is the pulling frequency, fresher will be 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;

- *push strategy*: in this case, 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 it's processing.

Currently, the Mashup tools implement only the pull strategy. The majority of tools have a static value of the pulling frequency that can be set by either the tool designer or the user. Besides, only few tools allow the user to define different pulling frequencies.

Another important parameter to point on here is the way the tool manages the pull interval. There can be two possible strategies to handle this issue: a global strategy and a local strategy.

1. *Global strategy*: here, 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 the trace of some sources (the ones having low refresh interval) than the others (the ones having high refresh interval).

2. *Local strategy*: in the local strategy, for each data source is affected it's own refresh interval. This pull interval is supposed to correspond to the one of the data source refresh itself. As A better trace is then kept of each data source.

For example, *Damia* adopts a *Local strategy*, associating at each source a *Refresh Interval* that defines the time that the feed from the URL is cached. After the time has exceeded, the feed from the specified URL is reloaded.

3.6 Output

The output of a Mashup can be consumed either visually or via other data formats. We consider the output as a dimension in this study since a user can

be interested in exporting his/her Mashup (the data flow) result in another format in order to process it with another particular application (e.g. Excel spreadsheet) for further processing instead of visualizing it. Considering for instance the example in Fig 3.3, and a user which wants to analyze the resulting data (weather condition and number of events in a certain location) in order to find whether there is a correlation among them. The user will have to export the result of the Mashup in a standard format (e.g. Excel spreadsheet or XML) for further processing. For instance, a data mining algorithm can be applied on this data to understand and extract some correlations.

3.7 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. This feature 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*).

3.8 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 experts in computing or security. This dimension defines the modality that the tool offers to enable privacy and security in the Mashup applications that he/she creates. It should be noted that this is a challenging area in the current Mashup and a lot of work remains to be done.

Also, 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?

1. **What**: indicates the object or the resource(s) that we want to share. For example, if a user creates a Mashup that integrates personal and public data, he may want to share only the components of the Mashup which manipulate the public data. The resources contained in the Mashup can be shared in different ways:

    - **Total**, the resource(s) (e.g. the Mashup application) is completely shared, i.e. all the components (e.g. source code, data, and output) composing the application are shared.

- **Partial**, in this case, only some of the components (e.g. only the source code but not the data) composing the application are shared with others.
- **Nothing**, this is the most restrictive option in which the user prefers to create his/her own application without sharing it with others.

2. **How**: indicates the wrights that a user can give to other users on the shared resource(s). The sharing policies analyzed here are the classical policies considered for the accessing of the data, e.g. 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).

3. **Who**: indicates the users with whom the resource(s) is shared with. This indicator can be: **All people**, **Group**, **particular User**. It should be noted that for each member, different sharing policies (what and how) can be specified and applied.

4 Tool Comparison

In this section we will analyze in more detail some Mashup tools[1] according to the different discussed dimensions. The objective of this section is not to make a comparative, qualitative or quantitative, study of the considered tools but only to analyze how do they manage and deal with the different described issues at the data integration level[2].

4.1 Damia

IBM provides a tool, *Damia[5],* to assemble data feeds from Internet and enterprise data sources. This tool is dedicated for data feeds aggregation and transformation in enterprise Mashups. Additional tools or technologies like *QEDWiki*[3] and feed readers, that consume Atom and RSS, can be used as presentation layer for the data feed provided by *Damia*.

Damia supports *REST* and *SOAP* web services and allows to fetch local Excel, CSV and XML files. It should be noted that these files must be first uploaded to the server, making them addressable, and can be then invoked through REST protocol. In addition if used in combination with Mashup Hub, *Damia* allows to assemble feeds obtained as results of query of data stored in relational databases like *Microsoft Access*[4] and *DB2*[5].

Damia provides two main widgets, for data access: URL and Catalog widgets. *URL* widget is used to extract the repeating elements from a feed, and the *Catalog* widget is used to fetch feeds from the Mashup Hub Catalog[6]. For processing data feeds, *Damia* engine translates all data sources into tuples of

[1]The list of the considered tool can be found in Table 4.1

[2]In Table 4.1, for the automatic data mapping, this is considered true if the source data has the same data model as the internal data model. Also, for data flow operators, we reference here the tables in the text

[3]http://services.alphaworks.ibm.com/qedwiki/

[4]http://office.microsoft.com/access

[5]http://www.ibm.com/db2

[6]http://services.alphaworks.ibm.com/Mashuphub/

Table 4.1: Summary of the considered dimensions for the tools analysis

		Damia	**Yahoo Pipes**	**MS Popfly**	**GME**	**Exhibit**	**Apatar**	**MashMaker**
API	HTTP	+	+	+	+	+	+	+
	SOAP	+	+	+	-	-	+	-
	REST	+	+	+	+	+	+	-
Data Model	XML-based	+	+	-	+	-	-	+
	Object-based	-	-	+	-	+	+	-
Data Mapping	Manual	+	-	+	-	-	+	+
	Semi-Automatic	-	+	-	-	-	-	-
	Automatic	-	-	-	+	+	-	-
Data Flow Operators		4.2, 4.3	4.4, 4.5	4.6	4.7	4.8	4.9	4.10
Data Refresh	Pull strategy	+	+	+	+	+	+	+
	Push strategy	-	-	-	-	-	-	-
	Global pull interval	-	+	+	+	+	+	+
	Local pull interval	+	-	-	-	-	-	-
	Interval Setting	+	-	-	-	-	-	-
Output	Data	+	+	-	-	+	+	-
	Visualization	-	+	+	+	+	-	+
Extensibility	Components	+	+	+	+	+	+	+
	Data	-	-	-	-	-	-	+
Sharing	Total	+	+	+	+	-	+	+
	Partial	-	+	-	-	-	+	-
	No thing	+	+	+	+	-	+	+
	Read only	+	+	+	+	-	+	+
	Read/Write	-	-	-	+	-	-	-
	All users	+	+	+	+	-	+	+
	Groups	-	-	-	+	-	-	-
	Particular user	+	+	+	+	-	+	+

sequences of XML, which constitutes it's internal data model. That is, if the data source is not in the same formalism like the internal data model, e.g. MS Excel, and since the source data are all stored in the 'content' field of the internal data model without taking in account the schema of the data source, a special container is created to receive that data. In this case, the mapping is manually performed between the internal data model if needed to perform more operations.

To consume and produce data, several operators are made available by *Damia*. We can distinguish between two categories of operators:

1. *Data elaboration and presentation operators*: these operators, shown in the Table 4.2, are used to perform modifications on the data or their structure,.

2. *Building operators*: these operators, shown in the Table 4.3, are used to

Table 4.2: Data Elaboration and Presentation Operators offered by DAMIA

Operator	Description
Transform	used for restructuring the schema of an incoming feed by adding/removing elements, adding/removing attributes to elements, or manipulating values of the elements. The transformation is accomplished by creating an output structure that is used to create a new feed.
Sort	used to sort feeds based on their values, in an ascending or descending order. Also, multiple sort keys can be used to perform the sort.
Group	used to gather entries with similar elements into a single entry based on a grouping expression. The grouping expression evaluates to a text value.

produce new data starting from a data source.

Table 4.3: Building Operators offered by DAMIA

Operator	Description
Merge	the operator is used to combine two source feeds based on an expression that is applied to the feeds. The expression compares an item value from the first feed with an item value from the second feed. All of the entries that satisfy the condition of the expression are merged, or joined, resulting in a new feed.
Union	the Union operator is used to combine two or more feeds into one feed. The entries from the first feed are added to the new feed, then the entries from the second feed.
Filter	used to extract specific items from a feed that meet the filter conditions.
Augment	allowed to combine the data from two incoming feeds into a single output feed of data. One must link an expression from the upper feed to a variable that he/she defines in the lower feed.

Damia caches all the data declared as data sources in a Mashup on its own server. A *pull strategy* is implemented to update the data on the *Damia* server and the pull interval is handled with a *Local strategy*. To set the pull interval, each source component has the *Refresh Interval* parameter for defining the time that the data from the specified URL is cached. After the time has exceeded, the data from the specified URL is reloaded. By doing this, *Damia* offers the possibility to consume it's output using other task specific tools, techniques, etc. (e.g. analysis tools).

As mentioned above, *Damia* aims to aggregate and manipulate data that can be reused from other applications. The output is exported using the *Publish* operator which transforms the output of the data flow into RSS, Atom, or XML feed by adding header information and content specific to the feed, and converting the tuples of sequences to the specified output feed type. Also, *Damia* is an extensible tool in that the user can either embed new functionalities inside the tool or invoke external services. The new operators can be written in PHP

language and can be plugged into the engine or can be made available as web services(SOAP or REST).

From a sharing policy management point of view, the tool offers the possibility of sharing the whole Mashup, i.e. total sharing, the output of the Mashup, i.e. partial sharing, or no thing. In the first case, another user might access all the information used by the Mashup creator. The Mashup is completely shared meaning that other users have access to source code, data and output. In the second case, the only shared thing is the final output. The resources are shared following one policy, which is the read only policy. The user can't specify another policy. The Mashup can be shared either with all people or no one. There is no possibility to share the application with a specific user or with a specific group of users.

4.2 Yahoo pipes

Yahoo Pipes allows to build mashup applications by aggregating and manipulating data feeds 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 modules are broken into different categories such as **Data Source** for data accessing, **Operators** for data manipulation [7] and so on.

Yahoo pipes supports mainly *REST* web services, but provides also specific modules to access services as *Flicker* for searching for photographs by keyword and geographic location, *Google Base* for allowing anyone to create and publish web-accessible information, *Yahoo Local* for searching for services in a particular area, *Yahoo Search* to build custom searches as a starting point for Pipes and to fetch the source of a given web site(*Fetch Page Module*) and a CSV file(*Fetch CSV Module*).

To combine data feeds, *Yahoo Pipes* translates the source formats (which can be RSS, Atom or RDF) into its internal RSS feed data model. The data mapping between the source data model and the internal data model is **Semi-Automatic**. In fact, if the name of the input fields of a feed match with the name of the RSS fields, the conversion into the is done automatically; otherwise many facilities are provided to help the user for the data mapping. An example is the "Item Building" module which is used to restructure and rename multiple elements in the feed, in order to convert a source data model to the Internal RSS feed data model.

To restructure the schema of incoming data, *Pipes* provides three operators described in the Table 4.4.

For data flow specification, *Yahoo pipes* provides only the *Union* operator to combine a list of item into a single list. Besides, to perform an elaboration on the data set, the following operators described in Table 4.5 are made available.

Pipes caches all the feeds it visits on its own server. A *pull strategy* is also implemented here to update the data on the server and the pull interval(that in our tests resulted to be 1 hour) is set for the whole application(*Global strategy*). The created pipes are hosted at Yahoo server and can be either accessed by a RSS or JSON client via it unique URL or visualized on the provide yahoo map. Besides, the pipes can be used like Mashup components to build a more complex

[7] http://pipes.yahoo.com/pipes/

Table 4.4: Data flow operators offered by Yahoo Pipes

Operator	Description
Regex	allows to modify fields in an RSS feed using regular expressions.
Rename	used to rename elements of the input feed and add new items in the inputs feeds.
Sub-Element	allows extracting sub-elements from the feed, which are buried in its hierarchy.
Union	allows to combine a list of item into a single list.

Table 4.5: Data flow operators offered by Yahoo Pipes

Operator	Description
Reverse	If the feeds are ordered, the **Reverse** module provides a way to change the order of feeds, by flipping the order of all items in a data feed.
Sort	Used to sort a feed by any item element, such as title. The items can be sorted in ascending or descending order.
Truncate	This module returns a specified number of items from the top of a feed.
Tail	This module truncates a feed to the last N items, where N is a number you specify.
Count	This module counts the number of items in the input feed, and outputs that number.
Filter	The Filter operator is used to extract specific items from a feed that meet the filter condition.
Unique	This module removes items that contain duplicate strings.
String Operators	These modules help manipulate and combine strings.
Simple Math	This module performs basic arithmetic, such as addition and subtraction.
Date Operators	These modules can perform elaboration on date as create a date object from a string value.
URL	This module builds URLs in either traditional or Web 2.0 style query-string format from a series of input fields.

pipe or their outputs can be combined with other tools that can process RSS feeds.

Yahoo Pipes is an extensible tool. If some functionalities the end-user needs are not offered, he/she can create a web service and invoke it from the system through the **Web Service** interface. This external service is accessible through JSON Interface and its output has to be a data type supported by the tool. The added functionality is visible only to the owner and cannot be shared with the whole community.

Finally, the Mashup can be shared with either all people or no one, in particular the sharing can be: (1) Total, meaning that there is a read access to source code, to the data, and to the output. The source code of the pipe and the output are shared. In this way another user might access all the information used by the Mashup creator. (2) Partial sharing meaning that the people with whom the Mashup is shared have read access to source code and the output

only. The data are not shared in this case. If a private element is used (Private string or Private text input) the code of the shared Mashup is available, but it is not possible to visualize the intermediate outputs. The Mashup output is available. The most restrictive policy is the Nothing policy which allows to have a read access to output. In this case, only the Mashup output is shared.

4.3 Popfly

Popfly is a web-based Mashup application by Microsoft[8] that allows 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"[9], "Facebook"[10] and "Virtual Earth"[11], and exposes one or more functionalities. A block is characterized by one or more operations with mandatory, optional input variables and an output. An operation defines the functionality exposed by the block such as display resources like photos or videos. The input variables are the parameters of the query to invoke the services, for instance URL of service. The output represents the way in which the output of the operation is provided to the user. The output can be either a data object or an HTML object that can be added to Mashup web page. The *Popfly* block supports mainly REST and SOAP services. In addiction, a WSDL block generator, that automatically produces a stub for a WSDL file, is made available.

In *Popfly*, the internal data model is in the form of object. The designer defines himself the characteristics and the behaviors of a block based on the source data model. Since, there is not an explicit internal data model, any transformation is performed by the tool for the mapping between the source data model and the internal data model. Therefore, the mapping is **Manual** give that it is manually specified by the designer.

Popfly is much more about data visualization than data manipulation, consequentially few operators are made available for data processing and integration. For data processing, operators for restructuring the schema of incoming data are not provided, since the Popfly's internal data model is object based, but some operators, described in Table 4.6 for object elaborations are made available.

Table 4.6: Operators offered by Popfly

Operator	Description
Sort	Sort operator is used to sort a list of input objects based on the values of the object.
Filter	This module filters the input list based on an arbitrary condition.
Truncate	This module returns a specified number of objects from the top of a input list.
Calculate	This module allow to do different match operation on the numbers.
Text Helper	This module allows to do some operators on the text as: (1)Split (returns an array of the substrings separated by a given separator), (2)getSubString: (Returns a portion of the input text, given a position and length)an so on.

[8] *http : //www.popfly.net/*
[9] www.flickr.com/
[10] www.facebook.com/
[11] www.microsoft.com/virtualearth/

Besides, for data Integration, *Popfly* makes available only the **Combine** module to join two sets of data of different types into one. For the data refresh, a *pull strategy* is implemented by *Popfly* to update the data. The pull interval depends on the frequency with the Mashup web page is reloaded by the user and concerns the whole application(*i.e. a global strategy*).

Popfly does not offer any true output function. Once a Mashup application has been developed and shared, it can be embedded into a web page, downloaded as a gadget, etc. But the mashed data cannot be exported in standard format for further processing instead of visualizing it.

For the extensibility, in *Popfly* the user can create his/her own blocks either by writing the Java Script code or by developing a SOAP web service. The created block can be plugged into the engine and shared with the whole community. Finally, the created Mashup can be shared with either all or no one. That is, two sharing policies are managed: (1) Total sharing where the user are given read access to the Mashup implementation, the data, and the output. The implementation of the Mashup and the output are shared. In this way another user might access all the information used by the Mashup creator. (2) Nothing where the Mashup is accessible only by the owner.

4.4 Google Mashup Editor

Google Mashup Editor(GME)[12] is an interactive environment to build, deploy, and distribute Mashup applications. The Mashup can be created using technologies like HTML, JavaScript, CSS along with GME XML tags and JavaScript API that further allows a user to customize the presentation of the Mashup output. *GME* allows also to consume RSS and Atom feeds accessible via *REST* web services. Local files contained data feed can be uploaded on the *GME* server and can be used through REST protocol. The user can also create his/her own feeds using Gdata API[13] and embed them in the Mashup's web page.

To operate on different types of data from different sources, the data in *GME* applications is managed with an Atom based data model named *Google Data* feed. *Google Data* feed is a data protocol based on Atom. The data from RSS feeds, are automatically converted to *Google Data* by *GME* through an XSL transformation. In reality, there is not an explicit mapping between the source data model and the internal data model, given that they have the same schema.

To handle the data, *GME* makes available operators for modifying the incoming data feed by sorting and filtering, shown in Table 4.7.

Table 4.7: Operators offered by Google Mashup Editor

Operator	Description
Sort	The sorting operator allows sorting the data on various types of elements as title, e-mail address, etc.
Filter	The filtering operator as for it allows retrieving specific data that meet the filter condition. The filter condition can be applied to various types of elements that appear in the feeds[14].

Operators for data merging and data schema manipulation are not explicitly

[12]http://code.google.com/gme/index.html
[13]http://code.google.com/gme/docs/data.html

provided but can be implemented using Javascript APIs, and XPath queries for data field access, and be plugged into the application. All the feeds visited by GME are cached on its own server. A *pull strategy* is implemented to update the data on the server but it does not support variable cache refresh frequencies, i.e. *Global Strategy*.

From the output point of view, *GME*, like *Popfly*, does not offer any true output function. The output of the Mashup can be only visualized using the provided visualization tool. Then, in what concerns the extensibility, if some functionalities the end-user needs are not offered, he/she can write Java Script functions that implement them. Like *Pipe*, the added functionality is visible only to the specific user and cannot be shared with the whole community.

Unlike the other Mashup tools, *GME* allows to specify sharing policies for the Mashup application. The sharing policy can be: (1) total, i.e. read access to source code, data and output. This means that the source code of the *GME* graph and the output are shared. In this way another user might access all the information used by the Mashup creator. (2) Partial, i.e. read access to source code. This means that If data is used to build the Mashup, and "no access" to other users is set, the code of the shared Mashup is available, but it is not possible to visualize or manipulate the data of the Mashup. (3) 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 be define to share it with a group or with all users by specifying Read/Write policies. Finally, the data feeds retrieved from external sources can be read and accessed by any user. The data of the application can be accessed in Read/Write mode by the designer, and can be shared by specifying the classical Read/Write policies for data accessing.

4.5 Exhibit

Exhibit[17] is a framework for creating web pages containing dynamic and rich visualizations of structured data. The generated web page is composed of two main files: 1) an HTML file, containing the description of user interface, and 2) a Data file, containing the data sources.

Exhibit can read data directly from own JSON format. But, if used in combination with Babel[15](which can be directly imported in Exhibit application), it allows to assemble data obtained by RDF/XML, N3, Bibtex, Tab Separated Values and Excel files. The data source can be either web data, accessible via REST web services, or local files which must be first uploaded on the *Exhibit* server and then invoked through REST protocol. The *Exhibit* data model is a graph model, based on JSON format and composed by a set of items and properties. The internal data model can be considered as a sub-model of the RDF data model[17]. When dealing with heterogeneous data, the mapping between the data source and the internal data model is automatically generated by the tool by means of *Babel* service. Anyway, if the conversion is not automatically performed by *Babel* web service, no facilities are provided for the user to design the transformation.

Exhibit like *Popfly* and *GME* is much more about data visualization than data manipulation, therefore few operators are made available. They are described in Table 4.8 Finally, the Besides, operators for data merging and data

[15]http://simile.mit.edu/babel/

schema manipulation can be implemented using Javascript and plugged into the application.

Table 4.8: Operators offered by Exhibit

Operator	Description
Sort	The **Sorting** operator allows sorting the displayed data.
Filter	The **Filtering** operator allows filtering specific data that meets the filter condition. The filter condition is defined using the *Exibit* expression language[16].
Search	**Searching** operator lets the user to search some text in the data.

For data refresh, a *pull strategy* is implemented. The pull interval depends on the frequency with which the Mashup's web page is reloaded by the user(*Global strategy*). In addition to the primary goal of *Exhibit*, i.e. creating web pages containing rich and dynamic visualization of structured data, it also lets the user to pull out the data providing exporters for RDF/XML, Exhibit JSON, Semantic wikitext and Tab Separated Values formats. Also, new functionalities in *Exhibit* can be implemented in Javascript and included like library into the *Exhibit* code. In particular new views can be added and more exporters can be registered.

Finally, the user can share the created Mashup by uploading it on his own web pages, since *Exhibit* does not give any support for the Mashup sharing.

4.6 Apatar

Apatar[17] is a Mashup data integration tool that helps users join desktop data with the web. Users install a visual job designer application to create integration called DataMaps. A Data Map is composed of data storage type and operator (which modify data in different ways) and defines the flow of data from the source(s) to the target(s). *Apatar* supplies the connectivity to applications such as MySQL, PostreSQL, Oracle, MS SQL, Compiere, SugarCRM, XML and flat files. In addition, it has connectors for the most popular Web 2.0 APIs such as Flickr, Salesforce.com and Amazon. The data sources are accessible mainly through REST web services. Besides, local files like Excel, RSS, Text file can be uploaded on *Apatar* server and then invoked through REST protocol. In *Apatar*, the internal data model is an object based. A specific object is automatically created for each data sources. Like *Popfly*, since there is not a specific internal data model, any transformation is performed by the tool for the mapping between the source data model and the internal data model.

Apatar is actually the only Mashup tool that provide a wide range of operators to consume and manipulate different types of data. In particular, to restructure the schema of incoming data, the user must define first the structure of the output, configuring the wished output connector(text file, or database, or whatever connector blocks), and then using the the *Transform* operator which can specify the correspondences between the input and the output fields. To perform elaboration on the data sets, *Apatar* makes available different sets of functions each one associate to a kind of data types such as string, date, number

[17] www.apatar.com/

and so on. These functions are available in each operator blocks. Finally, for data integration the provided operator are shown in the Table 4.9.

Table 4.9: Operators offered by Apatar

Operator	Description
Aggregate	used to combine two different data sources. The user first must define the structure of the output and then in the *Aggregate* operator, can specify the correspondences between the fields of the input data and the output.
Distinct	similar to the 'DISTINCT' operator in SQL, this operator eliminates the data duplications for the columns specified by the user.
Filter	used to extract specific data fields that satisfy the filter conditions.
Join	combined two different data sources based on the join condition that is applied to the input fields.
Split	this operator is named *Validate* in the terminology of *Apatar*, split a data sources in two separate tables according to a specific criteria. The first table contains all data for which the criteria is **True**, the second contains the rest of the records.

To refresh the data uploaded, a *pull strategy* is implemented in *Apatar* to update the data and the pull interval is set for the whole application(*Global strategy*). The created Data Map, one created and shared, is hosted on *Apatar* web site and its output can be either exported in standard format like RSS or can be redirected to whatever storage data such as MS SQL, Compiere and SugarCRM. *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 presentation layer.

Apatar is an extensible tool. In fact new connector and operator blocks and new functions can be developed in Java and plugged inside the engine(like new plug-in). These new functionalities can be also shared with the whole community. Finally, from a sharing policy management point of view, the tool offers the possibility of sharing the whole Mashup, i.e. total sharing, or no thing. Total sharing means that other users have access to source code, data and output. There is the possibility neither to specify policy for the accessing of the data nor to share the application with a specific user or a group.

4.7 MashMaker

MashMaker[14] is a web-based tool for editing, querying and manipulating web data. *MashMaker* is different from the described 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. A Web page is seen as two parts: the presentation and the data through *HTTP* protocol. To handle the data part, an RDF schema[18] is associated to the Web page by the user. For this part, users can use the *Extractor Editor* offered by the tool to edit the data model and formulate the XPath query for data extracting. The extracted schema is used by the tool to extract and structure the corresponding data included in the Web page.

[18]Resources Description Framework: http://www.w3.org/RDF/

It should be noted that the schema of the Web page is stored on the *MashMaker* server. The corresponding data are extracted when the Web page is retrieved by the navigator[19]. To build a mashup, different Web pages are combined in one. This combination is done by mean of widgets, a mini-application that can be added to an existing web page to enhance it in various ways such as provide data to other widgets. The final goal of this tool is to suggest to the user some enhancements, if available, for the visited web pages. The enhancements can be mashups or widgets which have been defined before by other users on top of the visited web page.

Back to the RDF description of a Web page. Such a description is composed of a set of nodes and properties. A node corresponds to a location on the web page. It is characterized by an XPath[20] expression to identify the position of the location and the corresponding value of that location. A node can have only a concrete value (*leaf node*) or properties (*compound node*). To create a mashup application, *MashMaker* must first extract the RDF description of the data, representing the internal data model of the tool, from the HTML pages [13]. For each web page, a different schema can be created: an URI-comprehension mechanism is used for normalizing URIs that are different but refer to the same web page.

The particularity and the interest of *MashMaker* is in it's operators. In fact, the idea is to offer the user the possibility of using operation that he is used to use in his desktop (e.g. copy and paste). That is, *MachMaker* offer the following basic operators, *Other services are added every day to the tool. Here we focus on the main operators which are directly related to data integration.*, described in Table 4.10.

Table 4.10: Operators offered by Intel MashMaker

Operator	Description
Match	This operator is similar to the *left join* operator in SQL. The output contains all the items of the main Web page (left relation or table in SQL) plus the matched items from the targeted Web page (the right relation or table in SQL).
Copy	takes a copy of the selected Web page. The copied object contains the presentation, i.e. the Web page itself, as well as it's RDF description if it is available.
Paste	Reproduces the copied object, i.e. the Web page and it's description. From the visualization point of view, this operator allows to past the data of a new web page into the main web page of the Mashup. From the data point of view, the two data sets, corresponding to the two Web pages, are merged.

As explained before, once the description schema is defined for a Web page and saved on the server, the data of the corresponding Web page are extracted when it's retrieved via a navigator. *MashMaker* implements then a *pull strategy* to update the data. The refresh interval is fixed by default but the user can refresh manually the data. From the output point of view, *Mashmaker* does not offer any true output function. In fact, once a Mashup application has been

[19]The navigator is supposed to have MashMaker installed.
[20]XML Path Language: http://www.w3.org/TR/xpath

developed and shared, the mashed data cannot be exported in standard format, but can be only visualized.

As the other tools, *MashMaker* is extensible since users can create new widgets and add and share them with other users. In addition to the possibility of extending the tool itself with new widgets, a particular schema can be enriched and extended by other users. The sharing of the created RDF schema of a particular Web page is automatically done by the tool. This one is shared with the whole community. For the widgets sharing, the user can create a widget containing private data, this widget and the Mashup containing it cannot be accesed by other users. Finally, the sharing can be done with all the users or no one. This means that it is not possible to manage group of users or particular users.

5 Discussions and Conclusion

In this section, we make a general discussion on the tools by considering their strangeness and weaknesses. We aim in the same time to try to give a look out on the possible points to consider for further improvements.

The mushup tools are mainly designed to handle Web data. This can be seen as an advantage and an inconvenient. In fact, it's an advantage since it offers access and management of some data available only on the Web, e.g. RSS feeds. A disadvantage since by doing this, user's data, generally available on desktops can not be accessed and used. This is a considerable disadvantage since users bring a lot of data on their desktops for cleaning, manipulation, etc. 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, the utility and the popularity of these protocols.

As discussed in the previous point, i.e. consuming Web data, the majority of 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 the XML format. This is due also to the fact that 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.

To manage data, the tools make available a small set of operations for data integration and manipulation. The set of provided operators is usually designed based o 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 don't 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, the reason is that the majority of the currently available APIs are REST based. The style of the REST protocol requires all communication 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 is motivated 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. The latter strategy however does not allow developing applications in which process 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 the Web 2.0 technologies is the creation, the reusing, the annotation and the sharing of web resources in an easy way. Based on these concepts, the 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 which can use the Exhibit output. However, this is a limited cooperation between tools especially that the tools have a lot of limitations and a user can't express his wishes using one tool only.

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. Observe that 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 overload for those servers. At this time, only *Intel Maskmaker* takes into account this problem applying some performance restrictions on the Mashup application[14].

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, which is owned by the tool's provider. Therefore, the tool's provider has the total control on the Mashup and, if a user wants to build an application containing that Mashup, the dependability attributes[6] of that application cannot be properly evaluated.

No tools provide information regarding the analysis of the performances and in particular information regarding the evaluation of the scalability. That information is needed to know the capability of a system to handle a growing amount of the data and the user request.

Finally, all the tools are supposed to target 'non-expert' users, but a programming knowledge is usually required. In particular, 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 require low or no programming effort to the user to build a Mashup, which is necessary to claim that the tools are targeted for end-users.

Bibliography

[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] *A Domain-Specific Language for Web APIs and Services Mashups.* Springer, 2007.

[5] Mehmet Altinel, Paul Brown, Susan Cline, Rajesh Kartha, Eric Louie, Volker Markl, Louis Mau, Yip-Hing Ng, David Simmen, and Ashutosh Singh. Damia: a data mashup fabric for intranet applications. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1370–1373. VLDB Endowment, 2007.

[6] J.-C. Randell B. Avizienis, A. Laprie. Fundamental concepts of dependability. *Technical Report Series - University of Newcastle upon the Computer Science*, 2001.

[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] Manish Bhide, Pavan Deolasee, Amol Katkar, Ankur Panchbudhe, Krithi Ramamritham, and Prashant Shenoy. Adaptive push-pull: Disseminating dynamic web data. *IEEE Transactions on Computers*, 51(6):652–668, 2002.

[9] Engin Bozdag, Ali Mesbah, and Arie van Deursen. A comparison of push and pull techniques for ajax, 2007.

[10] Francisco Curbera, Matthew J. Duftler, Rania Khalaf, and Douglas Lovell. Bite: Workflow composition for the web. In *ICSOC*, pages 94–106, 2007.

[11] Florian Daniel, Jin Yu, Boualem Benatallah, Fabio Casati, Maristella Matera, and Regis Saint-Paul. Understanding ui integration: A survey of problems, technologies, and opportunities. *IEEE Internet Computing*, 11(3):59–66, 2007.

[12] Schreiner W. Dustdar, S. A survey on web services composition. *Web and Grid Services*, 2005.

[13] Rob Ennals and David Gay. User-friendly functional programming for web mashups. In *ICFP '07: Proceedings of the 2007 ACM SIGPLAN international conference on Functional programming*, pages 223–234, New York, NY, USA, 2007. ACM.

[14] Robert J. Ennals and Minos N. Garofalakis. Mashmaker: mashups for the masses. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1116–1118, New York, NY, USA, 2007. ACM.

[15] Philip A. BernsteinDOI Erhard Rahm. A survey of approaches to automatic schema matching. *The VLDB Journal The International Journal on Very Large Data Bases*, 10:334–350, 2001.

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

[17] David F. Huynh, David R. Karger, and Robert C. Miller. Exhibit: lightweight structured data publishing. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 737–746, New York, NY, USA, 2007. ACM.

[18] Anant Jhingran. Enterprise information mashups: integrating information, simply. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 3–4. VLDB Endowment, 2006.

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

[20] Jin Yu, Boualem Benatallah, Regis Saint-Paul, Fabio Casati, Florian Daniel, and Maristella Matera. A framework for rapid integration of presentation components. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 923–932, New York, NY, USA, 2007. ACM.