Elucidating the Mashup Hype: Definition, Challenges, Methodical Guide and Tools for Mashups

Agnes Koschmider
Institute of Applied Informatics and Formal Description Methods
Universität Karlsruhe (TH)
Karlsruhe, Germany
+49 721 6084522
agnes.koschmider@aifb.uni-karlsruhe.de

Victoria Torres
Centro de Investigación Pros
Universidad Politécnica de Valencia
Valencia, Spain
+34 963877000
vtorres@pros.upv.es

Vicente Pelechano
Centro de Investigación Pros
Universidad Politécnica de Valencia
Valencia, Spain
+34 963879350
pele@dsic.upv.es

ABSTRACT

Mashups are a current hype that is attracting high interest by academia and industry now and in the next years. The idea behind a mashup is to create new content by reusing and combining existing content from heterogeneous sources. Advantages of mashups are that even people with no knowledge of programming languages can easily build new Web applications and create new forms of visualizations. To support the mashup construction process several tools have been proposed with easy-to-use functionalities. However, from the research perspective it is dissatisfying that neither a clear definition and classification model for mashups nor a separation between mashups and other forms of application integrations exist. The aim of this paper is to elucidate the mashup hype by providing a definition and classification model for mashups and to sketch a methodical engineering guide for mashups. Additionally, an overview of tools and languages supporting the mashup creation is presented.

Categories and Subject Descriptors
H.5.3 [**Web**]: Web Engineering– *model driven development, mashups, web services.*

General Terms
Design, Languages.

Keywords
Mashups, Mashup Construction, Tools, Method Engineering.

1. INTRODUCTION

The idea behind the term mashup is not new. The integration of different resources is an issue usually faced during software development. In fact, most engineering methods take into account the fact that some data and functionality is provided by external systems and provide mechanisms to specify them properly.

The main reason why mashups are gaining tremendous popularity is that even non-technical people are able to create new content and representations of resources without much effort or knowledge of programming languages. Another advantage is that the execution of mashups is not performed by black-box systems (like in the service-oriented execution) but rather user-driven, which makes the resource integration process much more transparent in comparison to conventional application integration platforms.

When undertaking a literature review about mashups then the search results about methodologies or infrastructures for mashup constructions are disillusioning. The amount of comprehensible approaches proposed so far is minimal and no clear line exists between mashups and technologies such as Service-oriented Architecture (SOA). The confusion even starts when looking for a generic definition of mashups. The definitions proposed in the literature span technical, business (in sense of economic) or industry perspectives. The next confusion follows when searching for a classification model for mashups. Too many references [for instance see 1, 2, 3] have different perspectives on mashups. In particular, the work of Hoyer & Fischer [1] is similar to the work of this paper. But Hoyer & Fischer limit their focus on enterprise mashups, proposes a more specific classification model and does neither discuss challenges for mashups nor investigate a guide for a methodical construction of mashups.

The intention of this paper is to provide a generic definition and a classification model for mashups. The definition of the term mashup affects also a separation of the term mashup to the currently frequently used technology SOA (how do these technologies complement each other and how do they differ?). Furthermore, we aim at discussing challenges of mashups and to present methodical concerns that should be considered when developing mashups. Finally, our paper presents an excerpt of tools and languages supporting the creation of mashups.

The structure of the paper is as follows. The next section discusses the different definitions for mashups and points out one clear definition. Links and differences between mashups and SOA are presented in Section 3. Section 4 describes a classification model for mashups. In Section 5 challenges are discussed that have to be faced with mashups. Section 6 presents concerns that should be considered when developing a consistent mashup application. An excerpt of mashup tools is given in Section 6. The last section summarizes our work and motivates the need for more research on mashups.

2. DEFINITIONS FOR MASHUPS

The term *mashup* has its roots in the musical domain referring to artists that mix several pieces of music, usually from different musical styles, into one single record. However, this term has been generalized and brought to other domains introducing the idea of derivating a new work by mixing two or more pieces. Among these domains we find the digital domain, the video domain and the Web domain. From these domains, in this work, we are interested in the last one, the Web domain. Even though its meaning is quite clear, there is no official definition of mashups. Nevertheless, it is possible to find several definitions such as the one provided by Wikipedia, which states that *a mashup is a Web application that combines data from one or more sources into a single integrated tool.* We regard this definition as too tight since it does not take into account, for example, aspects such as source heterogeneity. In fact, the integration/combination of different sources is not limited to data but also to functionality and layout styles.

Therefore, in order to provide a more precise and complete definition of this term we have extracted the most relevant terms included in several definitions found in the Internet. These terms include *Web page or application*, *integration*, *combination*, *reuse*, *data sources*, *APIs*, *third party data*, *Web 2.0* and *data processing*. Taking into account these terms we propose to define a mashup as *a Web-based application that is created by combining and processing on-line third party resources, that contribute with data, presentation or functionality*. It is important to note that in this definition, on-line third party resources refer to any type of resource available in the Internet, independently of the format in which this is provided (by means of an API, Web Feeds or screen scraping techniques). As a result, a mashup provides a new resource not conceived by the original combined resources. Finally, based on the nature of the original combined resources mashups can be categorized in different groups as explained in Section 4.

3. LINKS AND DIFFERENCES

From the definition provided in the previous section, mashups can be considered as a specific implementation of a SOA within the Web framework. In addition, an important difference between SOA and mashups is that third-party resources in mashups are not limited to services. As we have already mentioned, these can also be given as data sources. Therefore, the SOA paradigm is conceived to organize and utilize distributed capabilities that may be under the control of different ownership domains. However, the SOA paradigm embraces a broader scope providing also means to offer, discover, interact with, and use those distributed capabilities. In order to prove the relationship between SOA and mashups we present the SOA principles and comment the actual state in terms of mashups.

- **Service autonomy**. Services have control over the logic they encapsulate. In the case of mashups, the scope of each source is limited to the own source.
- **Service composability**. Collections of services can be coordinated and assembled to form composite services. In the case of mashups, two or more sources are connected properly to become a new source. In this case, depending on the type of mashup, the complexity of the composability changes.
- **Service contract.** Services adhere to a communications agreement, as defined collectively by one or more service description documents. Third party service description provide usually a simple and well documented API describing the way the resource can be used (method name, return type and textual description).
- **Service discoverability**. Service description languages such as WSDL are designed to be outwardly descriptive so that they can be found and integrated via available discovery mechanisms. In the case of mashups, sources should be described in a way that these could be processed and discovered. This means including some semantic information about the intention of the source as well as the preconditions and postconditions requirements to perform a proper use of the source.
- **Service encapsulation**. Beyond what is described in the service contract, services hide logic from the outside world. Service providers only provide access to the APIs that describe the source and nothing else is said about the logic behind the source.
- **Service Loose coupling**. Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other. The resources combined in a mashup are totally independent.
- **Service reusability**. Logic is divided into services with the intention of promoting reuse. In the case of mashups, each source was conceived for a very specific intention and the main objective is to be reused in different domains in order to get the most out of the own resource.
- **Service statelessness**. Services minimize retaining information specific to an activity. In the case of mashup, neither services nor data sources keep their state. However, the mashup –the composer- can keep the state by means of the used programming language.

This principles analysis reveals that mashups conform to SOA. However, the fact that mashups are strongly linked to the Web 2.0 requires extending these principles with one more. This new principle relates to the facility of use. The reason to introduce this principle is that mashups are targeted to a wider range of users, including non-programmer experts. In the mashup context, end-users play a very important role, they are not just the one in charge of using the applications but they also actively participate in their evolution.

4. CLASSIFICATION OF MASHUPS

To provide a classification of mashups we investigated several models for mashup categorization [1, 2, 3, 4]. To summarize from our point of view mashups can be classified based on the following four questions:

- What to mash up?
- Where to mash up?
- How to mash up?
- For whom to mash up?

In the following subsections we will separately regard each of these four perspectives on mashups.

4.1 What

Depending on the sort of assets being combined or integrated, mashups are assigned to one of the following three categories: *presentation*, *data* or application *functionality*.

A *presentation mashup* focuses on retrieving information and layout of different Web sources without regarding the underlying data and application functionality. For this type of mashup pre-built widgets are simply drags and drops into a common user interface. Usually, the creation of a presentation mashup requires little or no knowledge of programming languages.

A *data mashup* merges data provided by different sources (e.g., Web services, feeds or plain HTML) into one content page (i.e. for a given city combining different services to obtain its weather forecast, upcoming events and photos). The user mixes data from multiply sources and customizes the data flow of for example the Web page containing data from different sources.

A *functionality mashup* combines data and application functionality provided by different sources to a new service. The functionalities are accessible via APIs.

Based on the assets being combined one can find another classification of mashups such as *Mapping-Mashups* (combination of information into maps, e.g., Google maps), *Foto-/Video-Mashups* (combination of information into foto/video files e.g., from flickr), *Search/Shopping-Mashups* (integration of mechanisms for the comparison of product prices into Web pages) or *News-Mashups* (integration of news into personal Web pages). We regard this kind of mashup types as a refinement of the mashups introduced above where for example a presentation mashup can consist of resources being mashed out of news.

4.2 Where

Mashups can be distinguished depending on the location where they are mashed up. *Server-side* mashups integrate resources (e.g., services and data) on the server. *Client-side* mashups integrate resources on the client, often a browser. Usually a mixture of client-side and server-side applications is used for the creation of mashups. An elaborated explanation of server-side and client-side mashups can be found in [5].

4.3 How

Mashups can be further categorized depending on the modality the resources are integrated or combined to one representation.

The *extraction* mashup can be considered as a data wrapper collecting and analyzing resources from different sources and merging the resources to one content page.

In a *flow* mashup the user customizes the resource flow of the Web page combining resources from different sources. The resources are transformed and integrated within the mashup application.

4.4 For whom

Different mashup tools can be used to build mashups that combine content from different sources but distinguishing the target group being addressed. In this context, mashups can be categorized in *consumer mashups* and *enterprise mashups*, also referred to as *business* mashups.

A consumer mashup is intended for public use and combines resources (e.g., layout or data) from different public or private sources in the browser and organizes it through a simple browser-based user interface.

An enterprise mashup merges multiple resources (e.g., data and application functionality) of systems in an enterprise environment. These mashups combine data and application functionalities of different systems e.g., ERP, CRM or SCM in order to respond to their objectives. The creation of enterprise mashups requires considering security, governance or enterprise policies. Enterprise mashups provide a fast way for merging and representing internal and external enterprise resources from different sources without a middleman.

In the next section we will describe challenges that have to be met when constructing mashups.

5. CHALLENGES FOR THE MASHUP CONSTRUCTION

To establish mashups as an efficient technology for resource integration the following seven challenges should be solved where some of these challenges are not only unique to mashups but also to the World Wide Web.

- **Cataloguing**. Some Web pages are already available that list mashups and provide an interface for searching of mashups such as `programmableweb.com.` Mashup creators can insert their mashups in the list and thus share their mashups with others. But what is missing is a directory that stores and catalogues the mashups in a consistent way.
- **Data integrity**. Mashups are a quick way to create new applications but they can raise data integrity problems when changes of end-users are not valid against the underlying commitment. Another concern may raise integrity problems if e.g., an end-user finds a service that brings some value to the data or functionality included in the mashup. Then mashups need to be modified at runtime. Thus, when running a mashup control mechanisms should be considered that ensure the integrity of the mashup against end-user changes.
- **Making data Web-enabled**. Mashups are constructed of different resources that are available on the Web. However, currently a lot of data and functionalities are not set up on the Web and thus are not accessible via feeds, HTML or Web services. To make more resources “Web-enabled” require formats and tools that facilitate an efficient access and the connection of resources to the Web. Additionally, some data that is available on the Web cannot be reused to “mashing” because the data is capsulated with the presentation layer. Thus, mechanisms are needed that support the creation of mashups out of data and also tools that offer functionalities to decouple data from multiple sources from their presentation.
- **Security and identity**. While security challenges have been identified for mashups [7, 8] only few approaches exist that try to handle security lacks and identity of mashups [9, 10, 11]. Lawton [7] sketches that security challenges emerge when end-user connect dynamically to Web sites and not necessarily under the provider’s control. Additional security challenges arise if the mashup contains confidential data or security log-ins are required to enter some data. This requires mechanisms to control the user connection and the data security.
- **Sharing and reusing**. The next concern for mashup construction is that vendors of mashup tools should provide

mechanisms to allow end-users sharing their built mashups with others and thus facilitating the reuse of pre-built mashups. This means also that mashup owners need to give their permission before making the mashup available for the community. Otherwise end-users have to face legal implications of using this technology and have to expect consequences [6]. To facilitate a (legal) resource sharing the mashup should be defined in a format that is readable by different machines and consider accountability. Challenges that have to be met in this context are an easy-to-use access to mashups, efficient mashup search functionalities and light-weight formats that enable even for non-programmers a smooth mashup reuse.

- **Trust certificates**. The owner of such a directory service can issue a license that certificates the mashup. Because so far, no certification mechanisms exist that guarantee end-users the trustworthiness of the mashup. Similar to trust certificates for online shopping it is imaginable that mashup owners grant a licence at the owner of the directory service. In case of a positive certification the mashup owner can assure end-users the trustworthiness of the content and also the integrity of the mapping application.
- **Version control mechanisms**. Mashups consist of different resources collected from various sources. Resource owners are responsible for their content and can change and update its content or respectively its software whenever they regard it as necessary. To keep the mashup content up-to-date a version control mechanism is required that automatically informs the mashup owner about updates of the integrated underlying software (imagine the mashup is build upon several APIs).

In the next section we will present a guide for a methodical construction of mashups, which takes care on the presented challenges.

6. GUIDE FOR A METHODICAL CONSTRUCTION OF MASHUPS

There is a need for a methodology for the construction of mashups. Especially in an enterprise environment, where data and functionalities are stored in multiple systems, a consistent mashup construction methodology can guarantee an efficient merging of data and application functionalities. The service-oriented paradigm, which emerged in the last years can foster enterprise mashup development. Especially, when business process models contain data-driven decision points mashups can be used to call for them. But the novelty of a mashup application is that all data and functionality exists before one creates the application and these resources are given in a particular technology. In the case of composite services, where services invoke again one or more services, a consistent mashup construction methodology can be used to easily find and correctly combine the corresponding available services.

The approaches found in the literature focus more on the development of mashup architectures [12], of mashup systems [13, 14] or concepts for integration of data [15] rather than on a methodology for mashup construction. It would be desirable to consider Model Driven Development (MDD) techniques for the construction of this method. These techniques, used throughout the development process can help to minimize the impact that technology evolution has over software solutions. Among the benefits introduced by these techniques we find (1) technological independence (by keeping system descriptions in models that characterize the target domain), (2) semi-automatic construction of the system (by deriving knowledge from models applying model to model transformations) and (3) automatic code generation (by the application of model to text transformations).

The following guide can be used for a methodical construction of mashups. This guide is more useful for enterprise mashups rather than for consumer mashup where mashups are defined for trial or demonstration purpose.

1. State the problem domain and define:
   - Business objectives and
   - Success factors
2. Identify the IT environment, especially:
   - all application semantics in the corresponding problem domain
   - all resources to be mashed in particular services available in that domain
   - all information sources and sinks available in that domain
   - all processes in that domain
   - if necessary make resources Web-enabled
3. Identify technical requirements, especially:
   - catalogue all interfaces outside of the domain that should be leverage (data, services and simple information)
   - define new resources, services and information bound to those services
   - define new processes, as well as services and information bound to those processes
4. Identify the technology set, especially:
   - select your technology set
   - deploy e.g., by using a SOA technology
   - Test and evaluate
5. Maintain your mashup, especially:
   - define a version control mechanism
   - define a data integrity mechanism

On the one hand, according to MDD techniques, steps 1 and 2 would require the use of models to keep the system description independent of any technological detail. As a result, these steps would just consider IT necessities. On the other hand, steps from 3 to 5 attend to technological issues. In this case, transformations to maintain the consistency between the system description and the actual software solution are required.

7. TOOLS

Several mashup tools have been published that provide functionalities for building, storing and publishing mashups. These tools were conceived as Web 2.0 applications allowing users sharing their created mashups and providing them with very intuitive drag and drop facilities.

The range of these mashup tools spans from open-source tools to highly-cost licence tools. Some of the vendors offer a coding editor while others focus on users with no programming skills and thus provide easy-to-use access and application to their tool suites.

Based on our classification schema of mashups presented in Section 4 we discuss in this section tools supporting the creation of the corresponding mashup types.

Usually resource owners facilitate the access to their data by offering application programming interfaces (API). These APIs follow standard protocols and can easily be used to combine resources by a mashup tool from multiple sources. The whole range of APIs for mashups is listed on the Web page `programmableweb.com`. The APIs can be browsed by the preferred programming languages (e.g., PHP, JAVA or .NET) or predefined categories where the Google Maps API seems to be the most popular one.

The combined resources can quickly be displayed to users in a Web browser using, resulting transparent to users the techniques (SOAP [16], REST [17], Screen scraping or languages such as JSON[1] used to access and combine these resources.

Table 1 shows an overview of mashups tools categorized according to our classification model in Section 4 [2]. An analysis of mashup tools regarding its suitability for data analysis can be found in [18]. Table 1 does not consider server-side and client-side mashup styles because the location where to mashup differs with system configurations. In the following we will introduce four mashups tools covering different mashup categories and also a language for the mashup creation will be sketched.

Presentation, extraction and consumer mashup tool. Example for such a mashup tool is dapper [19]. The term dapper results from data and mapper, which exactly describes the functionality of the tool namely to simply drags and drops (map) pre-built widgets into a common user interface and subsequently to reuse and share the output.

The usage of dapper is very simple and does not require any knowledge of programming languages. Initially the user has to search for the Web pages out of them she would like to extract content. Then she highlights the area that should be extracted and finally dapper composes the extracted content to one representation. The user can make the output available for others who can reuse the representation in their mashup environment. Figure 1 shows a screenshot of the dapper interface. The user is scrolling a specific Web site and aims at extracting a logo of that side. The content to be extracted is highlighted in orange.

[1] http://www.json.org/

[2] In 2009 Google has closed the Google Mashup Editor. Therefore we do not consider this tool in our list.

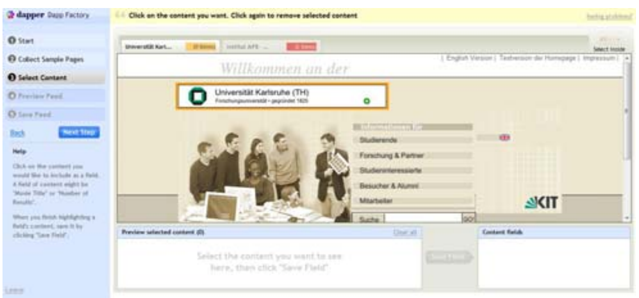


Figure 1. dapper

Data, flow and consumer mashup tool. Example for such a mashup tool that mixes data flow from multiply sources is DERI Pipes [20]. The implementation of this tool was inspired by Yahoo! Pipes but the advantage to this tool (in contrast to Yahoo's tool) is that DERI pipes can handle the RDF format and thus enables to build semantically enhanced mashups. DERI Pipes does not need any knowledge of programming languages but requires an understanding of data formats such as RDF. The final output respectively mashup is defined in XML or RDF and can be published in order to share with other users.

Figure 2 shows the user interface of DERI pipes. In this example the data of Tim Berners-Lee is mixed from three different sources.

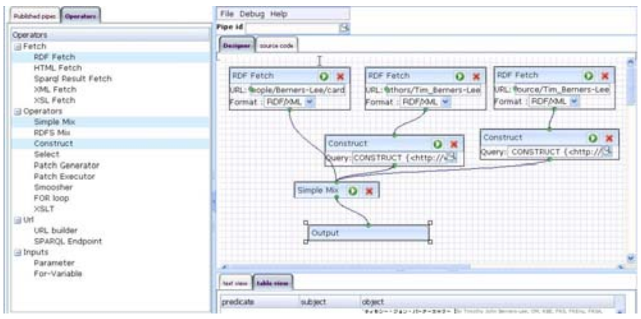


Figure 2. DERI pipe

Data, functionality, flow and enterprise mashup tool. Example for a tool providing functionalities to create enterprise mashups is Serena Mashup Composer included in the Serena Mashup Suite. According to the tool vendors Serena considers the integration of information, business processes and data to one common representation.

Figure 3 shows an example for the creation of a vacation request mashup. The idea of that mashup is that users can submit their vacancy request with several devices such as laptops or blackberries. The used syntax for the mashup creation resembles a service language such as BPEL [21]. But the difference is that Serena Mashup Composer can consume and mix any kind of widgets (feeds, plain HTML, services).

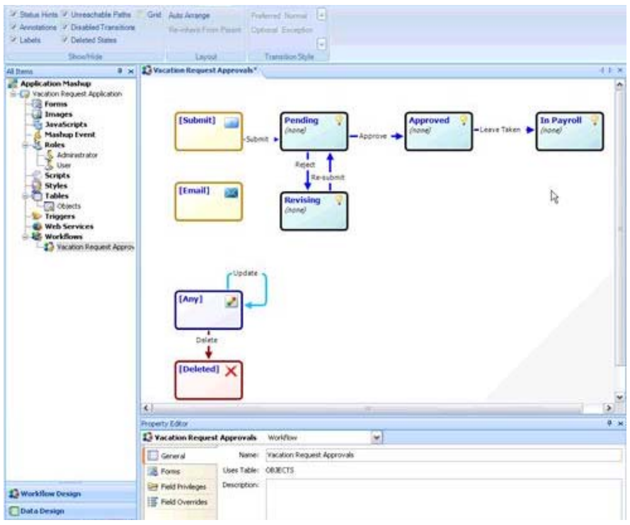


Figure 3. Serena Mashup Composer

Presentation, data, flow and consumer mashup tool. Example for such as mashup is Microsoft Popfly [22]. Microsoft Popfly uses Microsoft Silverlight [23], which is necessary since Popfly uses a great amount of optical effects of Silverlight. Popfly is not restricted to the generation of XML files but also offers possibilities to show the data (e.g., by using modules such as Microsoft Virtual Earth).

To use Microsoft Popfly also does not require any skills of programming languages but the user should come with an understanding of data formats. Figure 4 shows an example for creating a mashup with Popfly. In this example the Facebook API is used to visualize pictures of Facebook in a carrousel.

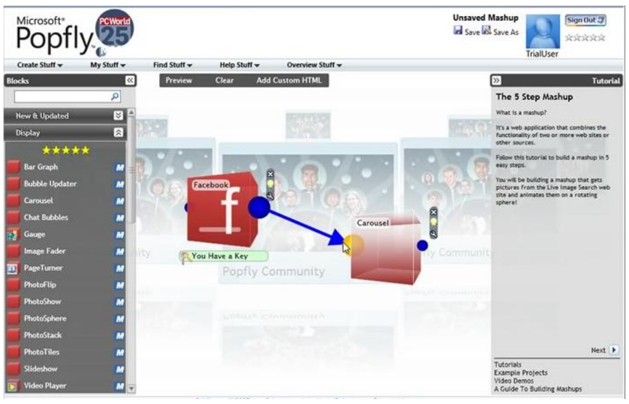
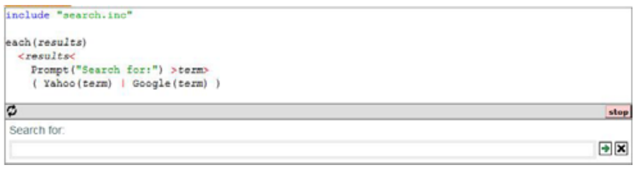


Figure 4. Microsoft Popfly

Language to create mashups. Several languages have been proposed for the construction of mashups [24, 25, 26]. Among all these languages, Orc [26] seems to be best documented and developed. The Web page of Orc offers an editor for the code programming. The code can be directly executed because the editor is connected to a server. Orc is also available as a standalone JAR file or Java application.

To write a mashup application in Orc the user should come with knowledge in functional programming languages since Orc is a concurrent functional programming language. The authors of Orc define three appropriate application areas. Orc can be used as a *general purpose programming language for concise encoding of concurrent and distributed applications, as a Web scripting language to create a Web service mashup* and *as an executable specification language for workflow applications and process coordination problems*. To be used as a language in the workflow field Orc implements several workflow patterns [27]. Thus, Orc is suitable to build process-oriented mashups.

Figure 5 shows an example of Orc syntax. The user is simultaneously searching in Yahoo and Google for a term that can be posed by the user after running the application. The ability to pose a query argument is given by the pre-defined method *Prompt*. The search field will appear when the user pushes the run button (already performed in this example). The classes *Yahoo* and *Google* are pre-defined in the "search.inc" library. The authors of Orc have already defined several libraries but the missing documentation of these libraries hamper the coding with Orc.



```
include "search.inc"

each(results)
  <results<
    Prompt("Search for:") >term>
    ( Yahoo(term) | Google(term) )
```

Figure 5. Orc

Table 1. Classification of Mashup Tools

	Presentation	Data	Functionality	Extraction	Flow	Consumer	Enterprise
Apatar		×			×		×
Data Mashups	×	×			×		×
Dapper	×			×		×	
DERI pipes		×			×	×	
Grazr	×			×		×	
IBM InfoSphere MashupHub		×		×			×
Intel MashMaker	×				×	×	
JackBe Presto		×	×		×		×
Microsoft Popfly	×	×			×	×	
Openkapow	×			×		×	
Procession		×	×		×		×
Rssbus	×			×	×		×
Serena Mashup Suite		×	×		×		×
Snap Logic		×			×		×
TIBCO PageBus		×			×		×
Yahoo! Pipes		×			×	×	

8. CONCLUSION

Mashups are suitable to build novel Web applications and to create new forms of visualization without little knowledge of programming languages. The lists of mashups created so far cover the whole range of Web applications (e.g., finance, government, sports or security).

However, little attention has been paid to classification models and methodological guides for mashups.

The aim of this paper was to question the constructs and methodical issues proposed so far for mashups and to define a consistent understanding of mashups starting with a definition and a classification model for mashups. We sketched a methodical guide for the construction of mashups and we presented several tools that support building mashups.

From our point of view further research is especially needed in the fields of version control mechanisms, mashup certification, mashup quality and data integrity.

Acknowledgement: This work has been developed with the support of MEC under the project SESAMO TIN2007-62894. The first author would like to thank the Karlsruhe House of Young Scientists that has partially funded the research reported in this paper.

9. REFERENCES

[1] Hoyer, V., and Fischer, M.: Market Overview of Enterprise Mashup Tools. In International Conference on Service oriented Computing, Volume 5364 of Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 708-721.

[2] Li, S., and Gong, J.: Mashup: a New Way of Providing Web Mapping and GIS Services. In ISPRS Congress Beijing 2008, Proceedings of Commission IV, 2008, pp. 639-649.

[3] Dion Hinchcliffe: Is IBM making enterprise mashups respectable?ZDNet Blog, 2006. http://blogs.zdnet.com/Hinchcliffe/?p=49&tag=nl.e622

[4] Hartmann, B. and Doorley, S. and Klemmer, S. R.: Hacking, Mashing, Gluing: Understanding Opportunistic Design, In IEEE Pervasive Computing, IEEE Educational Activities Department, 7(3), 2008, pp. 1536-1268

[5] Ort, E., and Brydon, S., and Basler, M.: Mashup Styles, Part 1: Server-Side Mashups, Sun Microsystems, May 2007.

[6] Zou, J. and Pavlovski, Christopher J.: Towards Accountable Enterprise Mashup Services, In Proceedings of the IEEE International Conference on e-Business Engineering, IEEE Computer Society, 2007, pp. 205-212.

[7] Lawton, G.: Web 2.0 Creates Security Challenges, In: Computer, 40(10), 2007, pp.13-16.

[8] Davidson, M. A., and Yoran, E.: Enterprise Security for Web 2.0, Enterprise Security for Web 2.0, 40(11), 2007, pp. 117-119

[9] De Keukelaere, F., and Bhola, S., and Steiner, M., and Chari, S., and Yoshihama, S.: SMash: Secure Component Model for Cross-Domain Mashups on Unmodified Browsers, In Proceeding of the 17th International Conference on World Wide Web, Beijing, China, ACM Press, 2008, pp. 535-544.

[10] Jackson, C. and Wang, H. J.: Subspace: secure cross-domain communication for web mashups, Proceedings of the 16th International Conference on World Wide Web, Banff, Canada, ACM Press, 2007, pp. 611-620.

[11] Vikram, K., and Steiner, M.: Mashup component isolation via server-side analysis and instrumentation. In Web 2.0 Security & Privacy Workshop. IEEE Computer Society, Technical Committee on Security and Privacy, 2007.

[12] Sheth, A. P., and Gomadam, K. and Lathem, J.: SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. IEEE Internet Computing, IEEE Educational Activities Department, 11(6), 2007, pp. 91-94.

[13] Gurram, R., and Mo, B., and Gueldemeister, R.: A Web Based Mashup Platform for Enterprise 2.0, In Web Information Systems Engineering Workshops, Volume 5176 of Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 141-151.

[14] Ennals, R. J., and Garofalakis, M. N.: Mashmaker: mashups for the masses. In Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM Press, 2007, pp. 1116–1118.

[15] DráŠil, P., Pitner, T., Hampel, T., and Steinbring, M.: Get Ready For Mashability! Concepts for Web 2.0 Service Integration. In Proceedings of the 10th International Conference on Enterprise Information Systems. Barcelona, INSTICC, 2008. pp. 160-167.

[16] SOAP Version 1.2 Part 0: Primer, W3C Recommendation, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/, 27 April, 2007.

[17] Fielding, R. T.: Architectural styles and the design of network-based software architectures, University of California, Irvine, 2000, Dissertation.

[18] Di Lorenzo, G., and Hacid, H., and Paik, H., and Benatallah, B.: Mashups for Data Integration: An Analysis, School of Computer Science and Engineering, University of New South Wales, Technical Report 0810 (2008),

[19] dapper: The Data Mapper. http://www.dapper.net/

[20] DERI Pipes: Open Source, Extendable, Embeddable Web Data Mashups. http://pipes.deri.org/

[21] Alves, A., and Arkin, A., and Askary, S., and Barreto, C., and Bloch, B., and Curbera, F., and Ford, M., and Goland, Y., and Guzar, A., and Kartha, N., and Liu, C.K., and Khalaf, R., and Knig, D., and Marin, M., and

Mehta, V., and Thatte, S., and van der Rijn, D., and Yendluri, P., and Yiu, A.: Web Services Business Process Execution Language, Version 2.0, Specification (2007).

[22] Microsoft Popfly. http://www.popfly.com/

[23] Microsoft Silverlight. http://www.microsoft.com/SILVERLIGHT/

[24] Maximilien, E. M., and Wilkinson, H., and Desai, N. and Tai, S.: A Domain-Specific Language for Web APIs and Services Mashups. In Proceedings of the 5th International Conference on Service-Oriented Computing, Volume 4749 of Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 13-26.

[25] Sabbouh, M., and Higginson, J., and Semy, S., and Gagne, D.: Web mashup scripting language. In Proceedings of the 16th International Conference on World Wide Web, Banff, Canada, ACM Press, 2007, pp. 1305-1306

[26] Orc Language. http://orc.csres.utexas.edu/

[27] Cook, W. R., and Patwardhan, S., and Misra, J.: Workflow patterns in Orc. In Proceedings of the 8th International Conference on Coordination Models and Languages (COORDINATION), Volume 4038 Lecture Notes in Computer Science, Springer-Verlag, 2006, pp. 82–96.