An Evaluation of Mashup Tools Based on Support for Heterogeneous Mashup Components

Saeed Aghaee and Cesare Pautasso

Faculty of Informatics, University of Lugano, Switzerland
first.last@usi.ch
http://www.pautasso.info/

Abstract. Mashups are built by reusing and combining building blocks, which are commonly referred to as mashup components. These components are characterized by a high level of heterogeneity in terms of technologies, access methods, and the behavior they may exhibit within a mashup. Abstracting away this heterogeneity is the mission of the so-called mashup tools aiming at automating or semi-automating mashup development to serve non-programmers. The challenge is to ensure this abstraction mechanism does not limit the support for heterogeneous mashup components. In this paper, we propose a novel evaluation framework that can be applied to assess the degree to which a given mashup tool addresses this challenge. The evaluation framework can serve as a benchmark for future improved design of mashup tools with respect to heterogeneous mashup components support. In order to demonstrate the applicability of the framework, we also apply it to evaluate some existing tools. The evaluation results provide a foundation for a useful discussion on the state-of-the-art mashup tools, with an emphasis on supporting heterogeneous mashup components.

1 Introduction

Mashups are Web applications built by reusing and combining mashup components. These components are not only programming and data abstraction but also can deliver Web contents that are not syndicated or made accessible via a public interface [1]. In other words, mashup components can be defined as any kind of reusable elements on the Web that can contribute to developing composite Web application. Thereby, mashup components often possess very heterogeneous characteristics in terms of the access methods through which they are published (e.g., protocols) as well the way they behave inside a mashup (e.g., synchronous and asynchronous interactions).

The heterogeneity of mashup components poses serious challenges on designing mashup tools aiming at lowering the barriers of mashup development through increasing the level of automation. It is due to the fact that one of the most important steps in automating mashup development is to abstract away this heterogeneity to enable seamless composition. Such an abstraction is defined as a component model that describes the characteristics of mashup components as well as the way they can be composed together [2].

A component model for mashups should be able to equally take two aspects into consideration. The first is indeed the level of abstraction that indicates the amount of technical skill required form a user to interact with the underlying mashup components. The higher the abstraction level is, the lower barriers are imposed on the end-users side. The second aspect is the expressive power in terms of how many and how heterogeneous types of mashup components come under its umbrella. The increased level of expressive power results in a more powerful tool. On the other hand, there is a trade-off between obtaining a higher level of abstraction and having more expressive power that should be considered while designing a mashup tool. This, however, requires a formal definition of the expressive power for mashup component models which is currently missing in literature.

The goal of this paper is thus to propose such a definition in the form of a framework. The framework can serve as a benchmark for evaluating mashup tools in this regard. Such an evaluation can then contribute to advancing the state-of-the-art mashup tools by identifying their weaknesses and strengths. The framework enables a white-box evaluation process and comprises six dimensions underlying the level of support for discovery, input/output data types, access methods, recursion, output types, and runtime behavior. To show how the framework can be utilized, we also apply it to evaluate some existing mashup tools.

The rest of this paper is structured as follows. In the next section we present our proposed framework. Section 3 is dedicated to evaluate some selected mashup tools based on the framework. The discussion, including the evaluation summary, will be given in section 4. An overview of the related work will be provided in Section 5. We conclude this paper in Section 6.

2 Evaluation Framework

To define the expressive power of a mashup component model, we need to understand what are required to be expressed by a user that concern the component model. In our proposed framework, these are referred to as dimensions which, in turn, are refined into a set of characteristics. To extract these dimensions, we consider a scenario in which a mashup is developed by a user using a tool. Initially, the user searches for relevant components depending on the goal of the mashup she intends to develop (discovery dimension). If the required components are not found in the library, they need to be newly wrapped by the user (recursion, access method, and output dimensions). Once the required components are ready, the user proceeds with the design process by composing the components to form a new mashup. During the design-time the user determines how the components are supposed to exchange data and control with each other (input/output data type and behavior dimension).

2.1 Discovery

Mashup component discovery is the whole process of retrieving appropriate components with regards to the needs of a developer [3]. Hence, a component model should provide adequate support to facilitate this step. Mashup tools supporting component discovery usually offer a local library storing reusable components. In order to support component discovery, a mashup tool can choose among three approaches outlined below:

– **Semantical Discovery.** Applying this approach requires a model that allows to add semantic descriptions of components (e.g., input/output parameters and functionality). In this approach, a component is discovered based on the information contained in its semantic description.

– **Syntactic Discovery.** In this approach component discovery is guided based on the component syntax exposed in its model (e.g., input/output data types).

– **Keyword Discovery** This approach is based on matching query keywords with tags and textual descriptions contained within the component model.

2.2 Input/Output Data Type

A mashup component interacts with others through its input and output parameters. In order to make use of a component inside a tool, the data types of its parameters should be defined in the tool component model. In general, these data types can be categorized into two main groups as follows.

– **Primitive.** This groups is equivalent to standard variable types of a programming language (e.g., string, int, boolean, etc.).

– **Multipurpose Internet Mail Extensions (MIME).** MIME types can be any standard data formats or media types found on the Web including (but not limited to) XML, JSON, RSS, and JPG.

2.3 Access Method

This dimension is concerned with the way in which a mashup component is made accessible for composition inside a mashup. The access method utilized by various mashup components are highly heterogeneous and can be categorized as follows.

– **Language-dependent.** This method forces the use of a specific programming, scripting or markup language. For instance, JavaScript APIs, HTML IFrame widgets, Plain Old Object Java Objects (POJOs), Enterprise Java Beans (EJB) can be all considered within this category. Though some of these methods are considered outdated (POJO and EJB), they are still being used within enterprise. Moreover, Google Maps [4] which is the most popular mashup components [5] is accessible via JavaScript APIs.

– **Protocol-based.** Using standard protocols for accessing a mashup component eliminates the requirements for a specific language. Popular protocols for mashup components are Web services (e.g., RESTful, HTTP, and SOAP) and Web feeds (e.g., RSS and Atom). According to the ProgrammableWeb [5], the dominant portion of Web APIs currently constitutes REST Web services.

– **Database.** Within a mashup, a database can be considered a component that act as either a read-only or a read/write data source. A database not only can deliver data and functionality (i.e., query and update features) but also can become a permanent storage for writing user-related data (e.g., username and password).

– **Non-Standard.** There are many Websites that do not officially allow any reuse of their content or backend functionality. These mostly follow the Web 1.0 paradigm, in which the content is merely assumed to be readable by humans. Extracting the content and functionality of such websites as mashup components might still be considered valuable, depending on the goal of the target mashup. These kind of mashup components are made accessible through two major non-standard techniques: Web scraping [6], which is the act of converting human-readable data to machine-readable formats, and Web clipping [7], by which only a portion of a Webpage is extracted.

2.4 Recursion

A mashup can be incorporated into another mashup as a component. This process can be called *recursion* whose concept is analogous to service composition [8]. In this sense, mashup components provided by third-party vendors are similar to atomic services. Likewise, a mashup is a user-defined component created through the composition of other mashups and different atomic components (like a composite service). A component model should take the recursion aspect into account, as it can reduce the required effort for mashup development through reuse.

2.5 Output

There are three types of output a mashup component can generate in the final mashup composition: functional, data, and visual. The development of a mashup can span one or all of the integration levels including process integration level, data integration level, and UI integration level, depending on the output types of its building components [9]. It is also of note that, a single component may have multiple output types (e.g., a Web Widget).

– **Functional.** Mashup Components with functional output are delivered as services that contribute to the business logic layer of a mashup. Such components are usually orchestrated together in a workflow to deliver a capability [10].

– **Data.** Components generating data act as external data sources, which deliver data to a mashup either as continuous data streams with real-time properties or as snapshots of a remote or local dataset. Most Web data sources are read-only, but in some cases they may also support updates. Within the mashup, they are likely to be converted, transformed, filtered, or combined with other data sources [11].

– **Visual.** Visual output is generated by UI components [12] or widgets [13]. These components provide some kind of graphical user interaction mechanism which can be reused at the mashup UI level. The visual part of a component is incorporated in the mashup UI independently from other UI elements and component.

2.6 Behavior

At the runtime, the control flow of a mashup determines the sequence of component invocation. Nevertheless, the internal execution mechanism of a mashup component may also affect its parent mashup control flow. This is referred to as the runtime behavior of a mashup component that can be either task-based or event-based.

– **Task-based.** A component with a task-based behavior represents a single invocation of a local or remote operation, which may provide an output given an input. It resembles traditional functions or methods, which execute and transmit responses only when called. In the context of the overall mashup, such components are passive (they are executed only when control reaches them).

– **Event-based.** When a component has an event-based behavior, it is triggered and produces an output only when a specific action (independent from the composition) has been taken (e.g., user interactions or an asynchronous message is received from a remote service). An event-based component is, therefore, an active part of a mashup, which may trigger the execution of a sequence of tasks.

3 Evaluation

In this section, we give an overview of the selected existing mashup tools and evaluate their corresponding component models based on the framework mentioned in Section 2 (Table 1). Considering the fact that our goal is not to evaluate all existing mashup tools but rather to demonstrate how the framework can be applied, we selected a sample group of mashup tools (Yahoo Pipes [14], Presto Cloud [15], Serena Mashup Composer [16], JOpera [17], and Husky [18]) based on two criteria. The first criterion was to ensure the diversity of End-User Programming techniques [19] utilized by the selected tools. These techniques for the selected tools include visual language (Yahoo Pipes, JOpera, Presto Cloud, Serena Mashup Composer), domain specific language (Presto Cloud), and spreadsheet-based programming (Husky). The second one takes into account the availability of the tools which otherwise can hinder the evaluation process.

In order to make the process of evaluation more concrete as well as to motivate and exemplify the need for a more powerful component model, we also benchmark the ability of the selected tools to develop an existing manually developed mashup called TwBe [20]. TwBe is a mashup developed using PHP and JavaScript without utilizing any mashup tools. The goal of TwBe is to provide a stream of YouTube videos as they are retrieved from a user's Twitter stream. Its main components include YouTube player[1], YouTube data API, Twitter API[2],

[1] http://code.google.com/apis/youtube/overview.html

[2] http://apiwiki.twitter.com/w/page/22554648/FrontPage

		Yahoo Pipese	Presto Cloud	Serena Mashup Composer	JOpera	Husky
Discovery	Semantic	-	-	-	-	-
	Syntactic	-	-	-	-	-
	Keyword	X	X	-	X	-
Data Formats	Primitive	X	X	X	X	X
	MIME	XML, RSS, ATOM, JSON	XML, RSS, ATOM	XML, JSON	XML, HTML	-
Access Method	Language-dependent	-	JS, HTML	JS, HTML	JS, HTML, POJO	-
	Protocol-based	REST, RSS	HTTP, SOAP, REST	SOAP, REST	HTTP, SOAP, REST	SOAP
	Database	-	SQL	-	SQL, JDBC	-
	Non-standard	Scraping	-	-	Scraping	-
Recursion		X	X	-	X	-
Output	Data	X	X	X	X	X
	Functionality	X	X	X	X	X
	Visual	-	X	X	X	-
Behavior	Task-based	X	X	X	X	X
	Event-based	-	X	-	X	-

Table 1. Evaluation of Mashup Tools

Twitter OAuth library[3], and a local MySQL database (Figure 1). In order to authenticate with the twitter, TwBe uses the Twitter PHP library for OAuth (there are also libraries for other languages such as JavaScript). The Twitter API is then invoked to retrieve tweets of a current user and periodically check for new ones. After the new tweets are available, those that do not contain link to a YouTube videos are filtered out. The videos that are going to be played by YouTube player are fetched from YouTube data API. Finally, the database is used to cache video tweets belonging to the current user in order to accelerate further access.

[3] https://github.com/abraham/twitteroauth

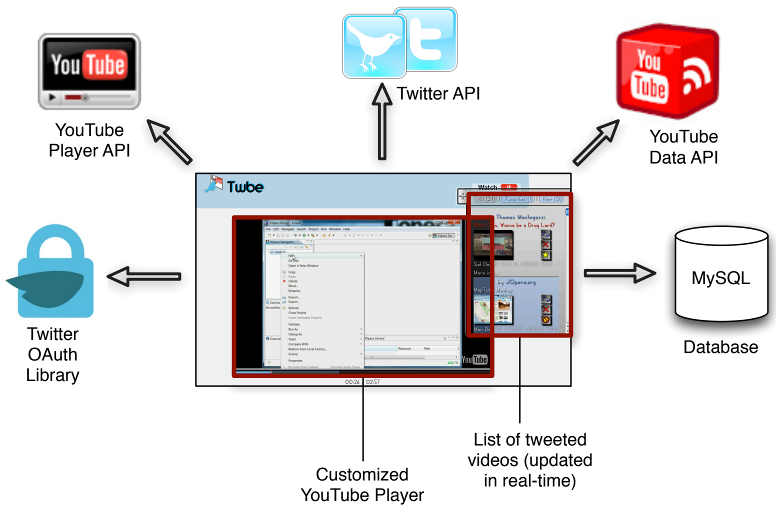

Fig. 1. TwBe Main Components

3.1 Yahoo Pipes

Yahoo Pipes is a popular tool for creating mashups by integrating data coming from various sources on the Web. It utilizes visual programming technique to hide the complexity of mashup development. The visual language is based on the wiring paradigm in which data sources, blocks, operators, and other tools are represented as parametrizable boxes which connect to each other. The result of connecting these boxes forms a pipe through which data flows and will be eventually visualized or syndicated to the user.

– **Discovery.** Yahoo Pipes supports component discovery, however, it only allows the discovery of mashups published in its local library. The discovery is based on matching keywords in user queries with the tags provided by the publishers.

– **Input/Output Data Format.** Primitive data types such as string and numerical values as well as frequently used MIME types like XML, RSS, Atom, and JSON are all defined in Yahoo Pipes. These are also the data types negotiated by the TwBe components.

– **Access Method.** The supported access methods include HTTP and RSS/Atom feeds. Thereby, YouTube videos and tweets can be easily retrieved as they are accessible via HTTP protocol. However, Twitter OAuth library and Youtube Player, which both use language dependent access method, as well as MySQL database cannot be utilized inside Yahoo Pipes.

– **Recursion.** Recursion is fully supported by Yahoo pipes. Mashups that are published in the tool library can be discovered and reused within a new mashup.

– **Output.** Components generating data and functionality are only supported by this tool. As a result it does not allow insertion of UI components such as YouTube Player.

– **Behavior.** This tool only supports task-based behavior of components and therefore, an event-based component cannot trigger a flow of control.

Overall, Yahoo Pipes can not be solely employed to develop TwBe as it does not support Twitter OAuth library, YouTube Player, and MySQL database.

3.2 Presto Cloud

Presto Cloud includes both a visual language and a powerful XML-based domain specific language called the Enterprise mashup Markup Language (EMML). It enables users to switch between the textual (EMML) [21] and visual mode depending upon their interests and background knowledge. Presto Cloud offers similar features as Yahoo Pipes for creating mashups integrating various data sources. It also adds support for integrating and designing mashup UI.

– **Discovery.** Component discovery is enabled and supported via keyword-oriented search.

– **Input/Output Data Format.** Components can declare both primitive and MIME (XML, RSS, Atom) types.

– **Access Method.** The two supported language-dependent techniques (JavaScript, and HTML) can be used to create *APPs*. APPs are similar to widgets that visualize data and can be recursively created through integration of other existing APPs. For instance, both the Twitter OAuth JavaScript library and the YouTube player that build the TwBe mashup can be wrapped as APPs. Moreover, all the frequently used protocol-based access methods (HTTP, SOAP, REST) are support by Presto Cloud.

– **Recursion.** Recursion can happen both in the back-end (data and functionality integration) and the UI (APPs)

– **Output.** Components with visual output are called APPs. *Blocks* abstract components with data and functional output.

– **Behavior.** Both task-based and event-based behavior of mashup components are handled by Presto Cloud. APPs can publish topics (events) to which other APPs can subscribe.
Presto Cloud can be used to create TwBe.

3.3 Serena Mashup Composer

Serena Mashup Composer is part of the Serena Mashup Suite. It decomposes mashups into orchestration, which defines the execution order of Web services, and application, which specifies the front-end of the mashup.

– **Discovery.** *Mashup Central* is a library containing templates and mashups shared by other users. However, it does not appear to support discovery of mashup components.

– **Input/Output Data Format.** Other than primitive data types, components can negotiate JSON and XML.

– **Access Method.** It supports protocol-based (REST, SOAP) and language-dependent (JavaScript, HTML) access methods. In the latter case, JavaScript and HTML is used to embed widgets. This can also be used to incorporate the Twitter JavaScript library for OAuth.

This tool does not support the use of databases, and therefore can not be used to create TwBe.

3.4 JOpera

JOpera is a rapid visual service composition tool. Service composition using JOpera is based on drawing a control flow graph that determines the sequence of service execution and one or more data flow graphs that indicate the flow of data between the services. JOpera allows abstraction of services of different types by concealing their internal mechanism (i.e., access method, input/output data types, etc.) behind a unified interface [22].

– **Discovery.** JOpera library stores both atomic and composite services and allows their discovery based on keyword-oriented search.

– **Input/Output Data Format.** JOpera data flow parameters can contain any data type.

– **Access Method.** It supports language-dependent (JavaScript, POJO, and HTML), protocol-based (HTTP, SOAP, REST), database (SQL, JDBC), and non-standard access methods (Web scraping) which cover all the access methods utilized by the TwBe components.

– **Recursion.** Recursion is supported by JOpera through the subprocess construct.

– **Output.** It handles all the three possible output of a mashup component (functional, data, visual).

– **Behavior.** JOpera not only supports task-based behavior of components but also allows handling exception events. Other types of event-based behavior such as data stream updates and UI events are not supported.

A very similar version of TwBe (in terms of UI), thanks to the high expressive power of JOpera in UI design (i.e., using *Echo* adapter that outputs DHTML code to browser), can be developed using JOpera.

3.5 Husky

Husky is a spreadsheet-based tool for service composition. Each cell of a Husky spreadsheet encapsulates a service. The sequence of service invocation is defined by placing service invocation events into adjacent cells.

– **Input/Output Data Format.** It only supports primitive data types.

– **Access Method.** The only supported access method is WSDL/SOAP Web services, which is not relevant in the case of TwBe example.

– **Output.** Since it only supports Web services, the only output types are functional and data.

This tool can not make any contribution to developing TwBe (none of TwBe components use SOAP Web service as their method of access).

4 Discussion

As the evaluation suggests, many of the selected tools do not provide adequate support for language-dependent mashup components. These are mostly exemplified by widgets accessible through JavaScript APIs (e.g., Google Maps). Tools like JOpera, Presto Cloud, and Serena Mashup Composer offer a JavaScript and HTML container to wrap such widgets. However, language-dependent components are not limited to JavaScript APIs and HTML, they may also be PHP libraries (for instance Twitter PHP library for OAuth).

A common limitation among the tools was the lack of support for event-based behavior of mashup components, which is usually the case in widgets. Even though JOpera and Serena Mashup Composer provides support for embedding widgets, they are unable to handle events fired by them through user interactions. The only tool of our selection that can manage UI events is indeed Presto Cloud. Furthermore, event-based behavior does not merely involve widgets, but also mashup components which generate data output can be used to subscribe to a source of streaming data. This also results in firing an event that should be handled by the mashup, e.g., when a new tweet containing a link to a YouTube video appears.

Interestingly, none of the tools thoroughly support component discovery. As a matter of fact, component discovery is one of the most important steps in the mashup development process. The majority of the selected tools (Yahoo Pipes, Presto Cloud, Serena Mashup Composer), except for JOpera, do not include atomic components in their library but only mashups that published by users. Moreover, the only discovery technique utilized by all of these tools was based on keyword-oriented search. Even though semantic discovery is not yet matured, its state-of-the-art [23] not only can contribute to streamlining mashup discovery but also can enable a higher degree of automation in mashup development.

Regarding the output types, the majority of the tools do not handle components maintaining their own UIs (i.e., visual output). Moreover, a tool that supports components with visual output may not necessarily support UI integration. JOpera is an example of this case, where the majority of UI components and widgets are supported but the required means of carrying on UI integration such as handling the communication of UI events [12] are not supported.

The level of support for recursion and input/output data types were satisfactory. In the latter case, however, the only supported MIME types were XML, HTML, RSS, and JSON. Though these are the most common media types for mashup components, they still need to broaden their support range to also cover less frequently used types such as YAML [24].

In general, supporting all types of mashup components is a challenging task. It gets even more challenging to keep the usability of a tool in a satisfactory level while increasing the expressive power of its component model. This can also be generalized to other aspects of mashup development such as composition and evolution. This is an important tradeoff that confronts the design of mashup tools, and thus needs to be addressed in future research.

5 Related Work

Previous efforts on proposing evaluation frameworks for mashup tools have been conducted along two directions. The first concerns the usability of mashup tools. For instance, the evaluation frameworks presented in [25,26] can both be considered towards this direction.

The second direction, within which this paper is to be considered, focuses on evaluating the expressive power of mashup tools. Previous evaluation frameworks target various aspects of mashup development that need to be expressed to end-users. For instance, [27] presents a benchmark for assessing mashup tools with respect to their data integration capabilities. The framework presented in [28] is also another example that classifies mashup tools and evaluates their expressive power concerning their support for process integration. We consider this paper as a complement to all the previous work done in this direction, the expressive power of component modeling for mashups.

6 Conclusion

The purpose of mashup tools is to lower the barriers of mashup development to the degree that even non-programmers can develop mashups. Even though the usability and ease-of-use are important factors for mashup tools, these should not compromise the expressive power offered by such tools and vise versa. One aspect of mashup development that determines this expressive power is the level of support for composing heterogeneous mashups components. In this paper, we have presented an evaluation framework that measures mashup tools based on to which extent they deal with the heterogeneity and diversity of mashup components. We defined the framework in terms of multiple dimensions and used it as the basis for undertaking an evaluation of a small group of mashup tools.

We believe the proposed evaluation framework can provide a roadmap towards an improved design for the next generation of mashup tools. To do so, heterogeneity can be addressed within a component model by means of adaptation and standardization. Adaptation is feasible in the short term and entails transforming heterogeneous mashup components into a common existing technology so that they conform with each other [2]. This method is harnessed by Mashape [29] by providing a programmable platform for converting various services and APIs into REST Web services. Standardization will require a more concerted effort and can provide a better solution in the long term, assuming that the resulting standard for mashup components become widely adopted. The Open Mashup Alliance (OMP) is now actively working on standardizing mashups, for example with EMML.

References

1. Ogrinz, M.: Mashup Patterns: Designs and Examples for the Modern Enterprise. Addison-Wesley (2009)

2. Assmann, U.: Invasive Software Composition. Springer (2003)
3. Zhao, Q., Huang, G., Huang, J., Liu, X., Mei, H.: A web-based mashup environment for on-the-fly service composition. In: Proc. of SOSE 2008. (2008)
4. Google Maps API. (Available at `http://code.google.com/apis/maps/documentation/javascript/`)
5. ProgrammableWeb. (Available at `http://www.programmableweb.com/`)
6. Schrenk, M.: Webbots, Spiders, and Screen Scrapers. No Starch Press (2007)
7. Smith, I.: Doing web clippings in under ten minutes. Technical report, Intranet Journal (2001)
8. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Computing **8** (2004) 51–59
9. Maximilien, E.M., Wilkinson, H., Desai, N., Tai, S.: A domain-specific language for web apis and services mashups. In: Proc. of ICSOC 2007. (2007)
10. Vrieze, P.d., Xu, L., Bouguettaya, A., Yang, J., Chen, J.: Process-oriented enterprise mashups. In: Proc. of GPC 2009. (2009)
11. Maximilien, E.M., Ranabahu, A., Gomadam, K.: An online platform for web apis and service mashups. IEEE Internet Computing **12** (2008) 32–43
12. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding ui integration: A survey of problems, technologies, and opportunities. IEEE Internet Computing **11** (2007) 59–66
13. Hoyer, V., Fischer, M.: Market overview of enterprise mashup tools. In: Proc. of ICSOC 2008. (2008)
14. Yahoo Pipes. (Available at `http://pipes.yahoo.com/pipes/`)
15. Presto Cloud. (Available at `http://www.jackbe.com/enterprise-mashup/`)
16. Serena Mashup Composer. (Available at `http://www2.serena.com/pages/mashups/campaigns/composer-download/index.html`)
17. JOpera. (Available at `http://www.jopera.org/`)
18. Husky. (Available at `http://www.husky.fer.hr/`)
19. Myers, B.A., Ko, A.J., Burnett, M.M.: Invited Research Overview: End-User Programming. In: Proc. of CHI 2006. (2006)
20. TwBe. (Available at `http://arc.inf.unisi.ch/twbe/twitter/`)
21. EMML. (Available at `http://www.openmashup.org/`)
22. Pautasso, C., Alonso, G.: From web service composition to megaprogramming. In: Technologies for E-Services. Springer Berlin / Heidelberg (2005)
23. Mohebbi, K., Ibrahim, S., Khezrian, M., Munusamy, K., Tabatabaei, S.G.H.: A comparative evaluation of semantic web service discovery approaches. In: Proc. of iiWAS 2010. (2010)
24. Yaml. (Available at `http://www.yaml.org/`)
25. Grammel, L., Storey, M.A.: An End User Perspective on Mashup Makers. Technical report, University of Victoria (2008)
26. Grammel, L., Storey, M.A.: A survey of mashup development environments. In: The Smart Internet. Springer Berlin / Heidelberg (2010)
27. Di Lorenzo, G., Hacid, H., Paik, H.y., Benatallah, B.: Data integration in mashups. SIGMOD Rec. **38** (2009) 59–66
28. Daniel, F., Koschmider, A., Nestler, T., Roy, M., Namoun, A.: Toward process mashups: key ingredients and open research challenges. In: Proc. of Mashups'10. (2010)
29. Mashape. (Available at `http://www.mashape.com/`)