# EnglishMash: Usability Design for a Natural Mashup Composition Environment

Saeed Aghaee and Cesare Pautasso

Faculty of Informatics, University of Lugano (USI), Switzerland
first.last@usi.ch

**Abstract.** The design of mashup tools combines elements from end-user development and software composition in the context of the Web. The challenge for mashup tool designers is to provide end-users with suitable abstractions, programming models and tool support for easily composing mashups out of existing Web services and Web data sources. In this paper we describe the design of a natural mashup composition environment based on the EnglishMash controlled natural language. The environment proactively supports users as they are learning the syntax of the EnglishMash language with features such as auto-completion, immediate feedback, live preview of the mashup execution and component discovery and selection based on natural language descriptions.

**Keywords:** Mashups, end-user development, natural language programming.

## 1 Introduction

Designing effective tools to facilitate mashup programming has become a key strategy to empower non-programmers to harness the potential of the programmable Web [1,2]. However, the main challenge that lies ahead in designing such tools consists of addressing the trade-off between expressive power against the assumed end-user skills [3,4]. In this paper, we present the usability and user interface design intended for the development environment supporting the EnglishMash mashup composition language, a tool that uses a restricted form of natural language (English) for mashup composition. To do so, we follow a use-case driven approach that starts by eliciting use cases from a case scenario and then maps each use case to a detailed model of the system's user interface [5].

One of the difficulties of applying natural language programming techniques lies in the need for end-users to discover and learn the constraints of the language syntax. Clearly, one cannot expect users to type arbitrary correct English sentences in the tool and effortlessly obtain a running mashup. Thus, the natural mashup composition language needs to be supported by the corresponding mashup composition environment, which is the primary focus of this paper. The highly interactive environment gives immediate feedback to users both in terms of correcting their mistakes but also showing them a live preview of the effect of their writing on the mashup output. Since basic sentences of the EnglishMash

M. Grossniklaus and M. Wimmer (Eds.): ICWE 2012 Workshops, LNCS 7703, pp. 109–120, 2012.

language are built out of component descriptions also expressed using natural language, we show how component discovery and selection can be seamlessly embedded into the lifecycle of the natural mashup composition tool.

The rest of this paper is composed as follows. In the next section, we give a brief introduction on the EnglishMash language. In section 3, we explain the barriers of EnglishMash in detail. In section 4, we extract and model the use cases of EnglishMash and provide its corresponding use case diagram. Next, we present the user interface modeling in Section 5. We discuss the related work in Section 6, followed by conclusion in Section 7.

## 2 EnglishMash: A Natural Language-Based Mashup Tool

EnglishMash is a mashup tool based on a controlled natural language—a subset of a natural language (e.g., English) restricted in terms of vocabulary and grammar. In terms of expressiveness, it supports various programming techniques such as conditional branches, event handling and iteration all expressed using very compact natural language grammar and syntax. For instance, the following Natural Language Mashup Description (NLMD) describes a mashup composing the Twitter search functionality (`https://dev.twitter.com/docs/api/1/get/search`) with the Google Maps widget (`https://developers.google.com/maps/`) as well as a HTML table widget.

```
''When the map is clicked, do as follows. Display a marker at
the location, and search for tweets at the location. Finally, show
the tweets on the table.''
```

Mashup Components in EnglishMash are also described in natural language. The Natural Language Component Descriptions (NLCDs) of abstract components allow to use the components in an NLMD by providing patterns of making clauses and sentences that together form an NLMD. For instance, the NLCD `''search tweets at [coordinate: longitude, latitude]''` associated with the Twitter search component is used to construct the clause `''search tweets for the location''`. The Twitter NLCD contains a placeholder for the required input parameters of the component (i.e., "longitude", "latitude") which, in the NLMD, is replaced with an object referring to the output parameter of the Google Maps widget `''the map is clicked''`.

The given NLMD along with a *component library and layout model* capturing the missing composition metadata including user interface design, and the list of the abstract components used by the NLMD, are passed to the EnglishMash compiler to generate its corresponding executable form. The runtime uses model transformation techniques to transform the input models to an executable form runnable by JOpera—a rapid visual service and mashup composition tool [6]. The detailed explanation of EnglishMash compiler and language, however, is out of the scope of this paper. For this paper, we only focus on the user interface and usability aspects of the EnglishMash natural mashup composition environment.

## 3 EnglishMash: Barriers and Required Skills

We divide EnglishMash users into two main groups: *NLMD authors*, who are those users interested in creating mashups without getting involved in programming tasks (e.g. non-programmers), and *component developers*, who are expert users (e.g., programmers) willing to develop useful components to be composed by NLMD authors. The focus of this paper is only on the NLMD authors, and therefore, the usability design proposed in this paper emphasizes the needs of users of this group.

Mashup programming is a challenging task that involves many advanced technical skills and knowledge, ranging from configuring the invocation mechanism of distributed mashup components to knowing how to program with Web scripting languages. On the one hand, this technical knowledge is abstracted from EnglishMash by hiding it inside its reusable component library. On the other hand, every tool or system requires certain skills to be mastered by its users and EnglishMash is no exception. Therefore, in order for NLMD authors to create mashups with EnglishMash, they will be required to acquire the following basic knowledge and skills:

**Components capabilities.** Before creating a mashup, the users must be aware of which components are subject to be mixed by the mashup, as well as what functionality is offered by each of these components. The required level of knowledge is remarkably shallow to the extent of being able to articulate the natural language-based description of the components. For instance, knowing that "**Google Maps** can **display** markers in a given **location**" is enough for a user to be able to compose the "Google Maps" component. This is made feasible by our component meta-model [7], which abstracts the complexity of the underlying invocation mechanisms of mashup components, including (but not limited to) their access methods (e.g., REST, JavaScript, SOAP, etc.), their input/output data types, and whether the components provide data, services, or user interface widgets.

**Components vs. Composition.** We assume that mashup components have been abstracted, described, and made available as a library to EnglishMash users by the component providers and not necessarily by the users themselves. The EnglishMash can thus be considered as an abstract composition language, which can be used to construct executable mashups once it is used in conjunction with the corresponding library of reusable components, which are described both at an abstract level with natural language and at a concrete level with executable code.

**Algorithmic thinking.** EnglishMash requires its users to have basic problem solving skills. These skills are needed for orchestrating the components of a mashup by describing how the mashup is supposed to work. Whereas this requires users to think algorithmically, as we are going to see, interesting non-trivial mashups can be already obtained with a small number of mashup components and simple descriptions.

**Syntax.** The biggest barrier imposed by EnglishMash is, indeed, the need to learn its core syntax rules. Even if the English language is used as a basis for

the Mashup composition language, i.e., every EnglishMash sentence is a correct English sentence, users must learn how to restrict their English sentences so that they can be executed by the EnglishMash tool. To do so, EnglishMash includes general composition syntax rules, which are used to define the structure of a mashup. Within this structure, users make references to component descriptions, which impose additional syntax rules contributing to increase the quantity of syntax rules, and consequently raise barriers to the learning process. However, the learning curve is a gentle slope, as the syntax associated with the component descriptions needs to be fully understood only if the components are selected to be included in the mashup.

As described by Nardi [8], end-users such as NLMD authors are not naive users, and they certainly have the ability, willingness and courage to learn, provided that the learning effort is worth the added value the mashup brings for them. Accordingly, the main requirement is to shorten the learning curve of EnglishMash as much as possible through the design of a usable user interface.

## 4 Use Case Modeling

As shown in the use case diagram of Figure 1, when starting to build a mashup using EnglishMash, a user should first have a goal in mind that reflects his/her situational needs. Let it be: ‘‘`I want a mashup to show tweets around a given location`’’. Having a goal in mind helps the user to elicit the needed mashup components, being, in this example, “Twitter” and “Google Maps”. Together with a powerful search engine provided by EnglishMash, the user then searches for the solicited components matching or approximating the given terms. For example, the user may search for the keyword “map” in the component library, which returns a number of mapping components which have been registered with the system. Afterward, the user chooses among the search results and adds the selected components to the stack, which indicates the list of components which are used in the mashup. If the search returns no results, then the user either creates the missing components, or ask other more expert users to do so.

Once the required components are available and selected, the user proceeds with the development of the target mashup. This use case is broken down into the following smaller use cases that should be supported simultaneously: (1) developing the logic of the mashup using NLMD, (2) designing the user interface of the mashup, (4) previewing the results of the execution of the mashup as it is being developed, (3) getting immediate feedback of syntax or runtime errors, (4) receiving NLMD writing aids in terms of auto-completion with drop-down menus containing suggestions.

The component discovery and composition use cases are clearly intertwined, since while developing a mashup, the user should be able to search for and add additional components to be composed within the target mashup, even if the mashup has already been partially described. For example, after adding the

"Google Maps" component to the stack, the user can start typing the mashup description, which should refer to the natural language description of the component. Typing the first few characters of `show` into the description will trigger another component lookup, based on the entered string. The tool will automatically proposed to complete the description with the `show a map.` text. Clicking on the auto-completion suggestion will 1) enter the completed natural language description of the component; 2) trigger a rebuild of the mashup, which will be executed and the results (i.e., the map widget centered around a default location) will be shown in the output live preview area.

The user may then proceed to define how to interact with the mashup widget. Typing a new sentence beginning with `When` will provide a list of auto-completion possibilities, including "When the map is clicked", "When the map is zoomed", "When the map center is moved". These correspond to events made available by the map widget component previouls added to the stack. After selecting the appropriate event, the user can continue typing to specify what should happen in the mashup when the event occurs.

Finally, the user should be able to deploy the mashup in production and share it with others. Even after a mashup has been published, it still remains modifiable and can be adjusted, redeployed and republished at any time.

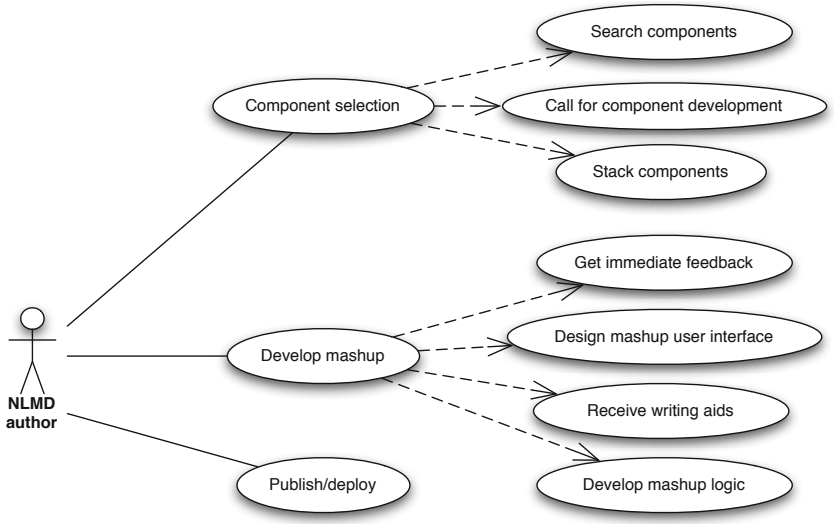


**Fig. 1.** Use case diagram for semi-automatic mashup platforms

## 5 User Interface Modeling

To model the user interface corresponding to the previous use cases, we used the UML*i* (`http://trust.utep.edu/umli/`) modeling language. UML*i* is a UML extension to support user interface modeling. To this end, it introduces user

interface diagram used to model the graphical elements of an interface, and extends the UML activity diagram to model the interaction between users and the target user interface.

The use cases elicited in the previous step (Figure 1) drive and inform the user interface modeling by providing various context-of-use scenarios. These scenarios, in turn, help to extract the target user interface elements and produce the user interface diagram as well as to model the user interaction with these graphical elements using the extended activity diagram. According to the EnglishMash uses cases, the following context-of-use scenarios can be identified: (1) searching components, (2) selecting components, (3) NLMD authoring, (4) mashup user interface design, (5) live mashup execution, and (6) publishing and deploying mashups.

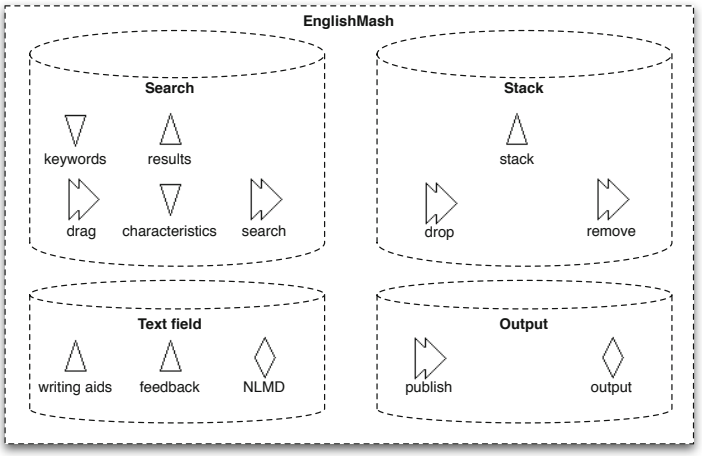


**Fig. 2.** UML*i* user interface diagram for EnglishMash

The high-level elements in a user interface diagram is a *FreeContainer* that corresponds to a window or a web page. Since there is no logical order between the scenarios 1-5, we incorporate a single user interface in which all the use cases are considered and integrated as a whole. Users should not have to navigate between different "screens" or switch between different "operational modes" in order to use the tool for searching components (by typing partial natural language sentences), composing components (by editing and refining the natural language description), and by observing the results which are immediately available. Likewise, we use a What-you-see-is-what-you-get (WYSIWYG) approach to design the mashup user interface and deal with widget placement and layout issues.

A FreeContainer is structured in *Containers*. As it is shown in the user interface diagram (Figure 2), the main FreeContainer ("EnglishMash") consists of four Containers associated to the context-of-use scenarios. These are "search", handling the component discovery scenario; "stack", supporting the component

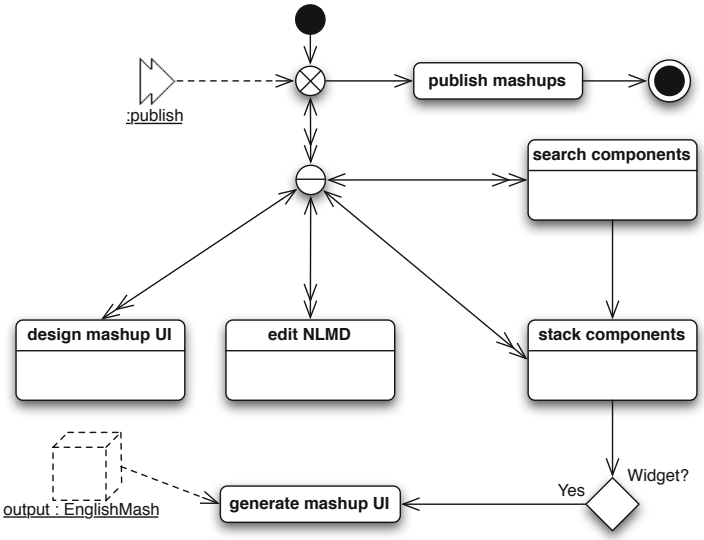


**Fig. 3.** UML*i* main activity diagram of the EnglishMash tool user interface

selection scenario; "text field", enabling the NLMD editing scenario; and "output", handling both the user interface design and the live execution preview scenarios. In the latter case, two scenarios are merged into a single container to simplify the design of the mashup user interface, which is tightly connected with the result of the mashup execution as described in the "input mashup text" container.

Within each Container, UML*i* allows to distinguish with specific graphical elements the user interface controls responsible for (1) sending visual feedback to users (e.g., "syntax checker"), (2) receiving information from users (e.g., "keywords"), (3) simultaneously sending and receiving information (e.g., "NLMD"), and (4) modeling user interface events (e.g., "drop").

To fully model the EnglishMash user interface requires also describing its interactions with users, we do so through UML*i* activity diagrams. Figure 3 illustrates the main activity diagram modeling the interaction with the EnglishMash user interface. It contains six activities, out of which four are composite ("design mashup UI", "edit NLMD", "stack components", and "search components"). The main activity diagram starts by a loop that executes one or none of these composite activities at a time. Inside the loop, the activities "design mashup UI" and "edit NLMD" are both followed by the immediate execution of the "generate mashup UI', which involves the regeneration and synchronization of the output mashup (live execution preview). Also, the live execution preview activity is activated ever time the "stack components" state results in removing or adding a widget (i.e., components with user interface) to the stack. The loop stops when the user publishes the mashup by triggering the "publish" graphical element (e.g., clicking a button).

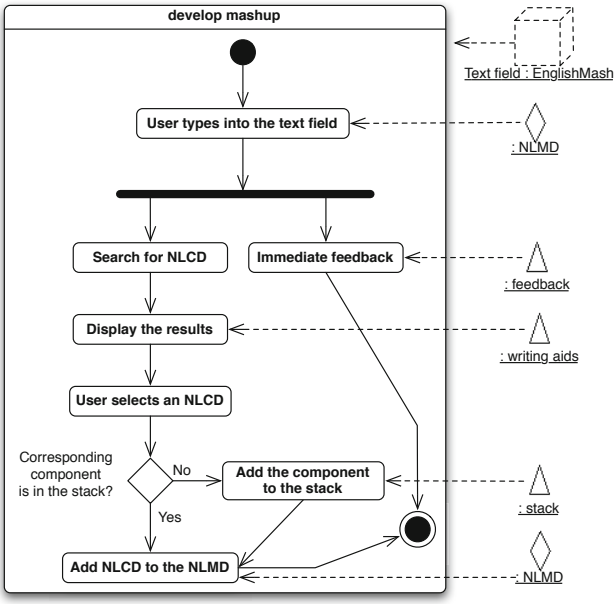


**Fig. 4.** The UML*i* activity diagram for the "edit NLMD" activity

The composite activities are depicted in Figures 4, 5, 6, and 7. According to the "search component" activity, user begin their search by entering some keywords. If the keywords produce no result, then the user can call for the development of his/her solicited components by describing their characteristics using natural language. The component description will be added to the library but until a matching component implementation is registered, the component will not be executable. In the "stack components" composite activity, in turn, users can either remove a component or add a new one by dropping a result from the search results to the stack.

In the "edit NLMD" activity diagram, as the user types into the "NLMD" graphical element, immediate feedback (syntax and runtime errors) as well as writing aids (auto-completion) will be provided. In the latter case, the partial text input by the user is used by EnglishMash to search the component library for components having a matching NLCD. The results are displayed to the users in a drop-down menu. After choosing among the results, if the selected NLCD does not already belong to the components in the stack, its corresponding component will be added to the stack. Finally, the "designing mashup UI" state involves resizing or relocating widgets in the mashup user interface.

A snapshot of the concrete EnglishMash user interface based on the mentioned models is illustrated in Figure 8. To implement the user interface, we used client side-technologies such as HTML5, CSS3, and JavaScript augmented with the JQuery user interface libraries (`http://jqueryui.com/`)

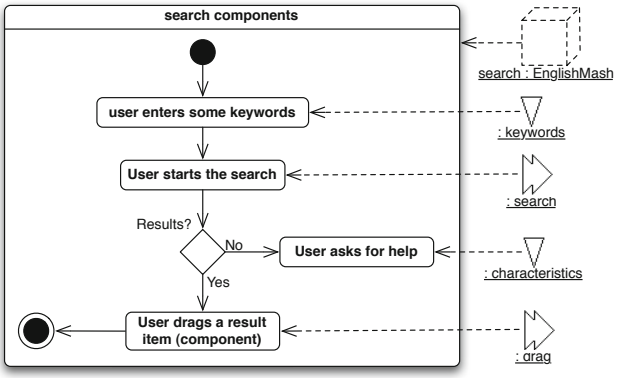


**Fig. 5.** The UML*i* activity diagram representing the "search components" activity

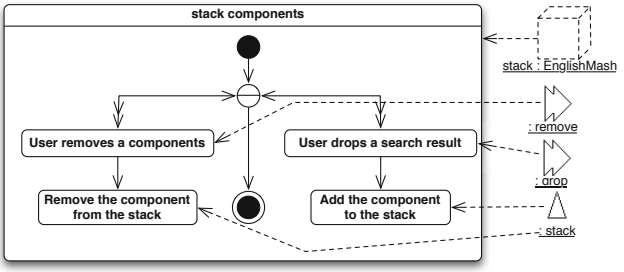


**Fig. 6.** The UML*i* activity diagram corresponding to the "stack components" activity

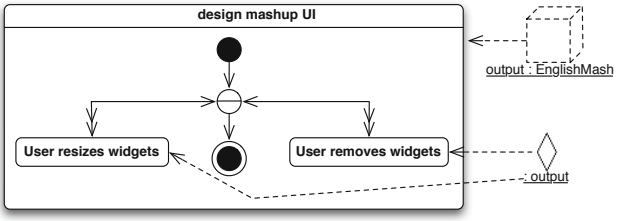


**Fig. 7.** The UML*i* activity diagram that illustrates the "design mashup UI" activity

## 6 Related Work

Mashup tools can be generally classified into automatic and semi-automatic [9]. Automatic tools do not necessitate the involvement of users, whereas semi-automatic tools aim at empowering users to quickly build their desired mashups

through providing utmost assistance and guidance. EnglishMash along with the majority of mashup tools like Yahoo! Pipes (`http://pipes.yahoo.com/`), DashMash [10], and JackBe Presto (`http://www.jackbe.com/`), are all categorized as semi-automatic. In fact, the users of semi-automatic tools are required to go through a learning process that, depending on the design of the tool, can be short or long. On the other hand, automatic tools do not require prior learning, but run the risk of deviating from user needs by producing irrelevant mashups. The process of validating and correcting the resulting mashups (if provided by the tool) can, in turn, become a time-consuming task [8].

The distinction of EnglishMash from other semi-automatic tools lie in its novel interaction technique, being an effective combination of natural language and WYSIWYG techniques. This, therefore, distinguishes EnglishMash from other mashup tools using either of the techniques. For instance, ServFace [11] is a tool relying on WYSIWYG technique. The shortcoming of the tool is in modeling all the required composition techniques (e.g., branches and loops) on the user ointerface level. Regarding natural language, Natural Mashup [12] is a tool incorporating a natural language-based interface for composing mashups which however does not support user interface integration and design which are integral part of mashup development [13]. Mashup auto-completion has been proposed in [14]. In our approach we rely on natural language descriptions of mashup components.

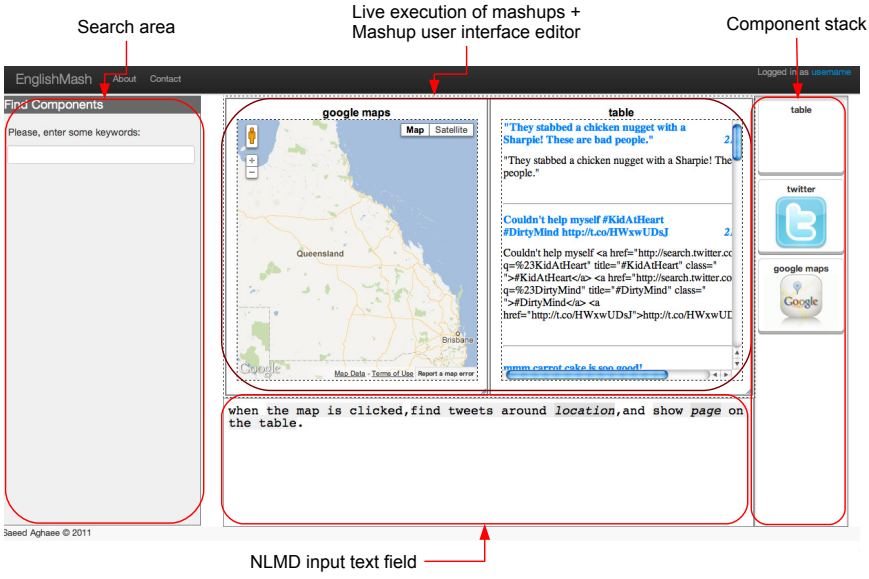


**Fig. 8.** The Web-based composition environment for EnglishMash

## 7 Conclusion

Designing a usable interface for a mashup tool plays an important role in addressing the trade-off between maximizing its expressive power and ensuring that it presents users with a gently-sloped learning curve. In this paper, we used a use-case driven approach to design a composition environment for EnglishMash, a mashup tool that relies on a novel approach that lies at the intersection of model-driven development and natural language processing. The tool makes use of the EnglishMash mashup composition language that conforms to a restricted form of natural language (English). The high level of abstraction offered by the language eliminates the need for expressing technical details, and consequently makes the executable description of the mashup very similar to its natural language description. The tool supports the users in learning the constrained syntax of the language by means of immediate feedback, both in terms of informing users about syntax and semantic errors, but also by providing a live preview of the mashup execution results. Users typying the description of the mashup are supported by auto-completion features which are closely tied to the component discovery and selection features of the tool.

The paper describes first a set of common use case scenarios for the tool and then presents a detailed model of the user interface of the EnglishMash environment. To do so, we used UML*i*, which is an extension to UML to support user interface modeling, and produced both a user interface diagram, specifying the constituent abstract graphical elements of the user interface, and its corresponding activity diagrams representing the interactions between the user interface and users. Finally, we implemented the user interface using client-side technologies (e.g., JavaScript, HTML5, and CSS3) after creating a mapping between the abstract graphical elements and the concrete elements corresponding to HTML tags, attributes, and events.

We are currently undergoing an internal evaluation of the tool with a small user community made of non-programmers (e.g., High School students). We plan to publish the tool on the Web after its preliminary evaluation has been concluded together with a library of example mashups and reusable component descriptions.

**Acknowledgements.** The work presented in this paper has been supported by the Swiss National Science Foundation with the SOSOA project (SINERGIA grant nr. CRSI22_127386).

## References

1. Benslimane, D., Dustdar, S., Sheth, A.: Services mashups: The new generation of web applications. IEEE Internet Computing 12, 13–15 (2008)
2. 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, 59–66 (2007)

3. Bozzon, A., Brambilla, M., Facca, F.M., Carughu, G.T.: A conceptual modeling approach to business service mashup development. In: Proc. of ICWS 2009, pp. 751–758. IEEE Computer Society (2009)
4. Cao, J., Riche, Y., Wiedenbeck, S., Burnett, M., Grigoreanu, V.: End-user mashup programming: through the design lens. In: Proc. of the CHI 2010, pp. 1009–1018 (2010)
5. Lif, M.: User-interface modelling: adding usability to use cases. Int. J. Hum.-Comput. Stud. 50, 243–262 (1999)
6. Pautasso, C., Alonso, G.: The JOpera visual composition language. Journal of Visual Languages and Computing 16, 119–152 (2005)
7. Aghaee, S., Pautasso, C.: The mashup component description language. In: Proc. of iiWAS 2011 (2011)
8. Nardi, B.A.: A small matter of programming: perspectives on end user computing. MIT Press, Cambridge (1993)
9. Fischer, T., Bakalov, F., Nauerz, A.: An overview of current approaches to mashup generation. In: Proc. of WM 2009, pp. 254–259 (2009)
10. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C.: DashMash: A Mashup Environment for End User Development. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 152–166. Springer, Heidelberg (2011)
11. Nestler, T., Feldmann, M., Hübsch, G., Preußner, A., Jugel, U.: The ServFace Builder - A WYSIWYG Approach for Building Service-Based Applications. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 498–501. Springer, Heidelberg (2010)
12. Belaunde, M., Hassen, S.B.: Service mashups using natural language and context awareness: A pragmatic architectural design. In: Proc. of EDOCW 2011 (2011)
13. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A framework for rapid integration of presentation components. In: Proc. of WWW 2007 (2007)
14. Abiteboul, S., Greenshpan, O., Milo, T., Polyzotis, N.: Matchup: Autocompletion for mashups. In: Proc. of ICDE 2009 (2009)