Creating Mashups by Direct Manipulation of Existing Web Applications

Giuseppe Ghiani, Fabio Paternò, and Lucio Davide Spano

CNR-ISTI, HIIS Laboratory, Via Moruzzi 1
56124 Pisa, Italy
{giuseppe.ghiani,fabio.paterno,lucio.davide.spano}@isti.cnr.it

Abstract. We present an environment to enable people without programming knowledge to create mashups composed of Web components selected directly from existing Web applications. The authoring environment allows the creation of communication among components originally belonging to different applications. We report on some example application, the underlying architecture of the environment, and a first user test.

Keywords: Mashups, Web applications, End User Development.

1 Introduction

The Web is still the most common user interface. Millions of Web sites are available and ever more people access them daily for their activities. However, each user has slightly different needs and activities to carry out. This has pushed interest in mashups, whose purpose is to compose user interfaces, contents, and functionalities from various sources. Indeed, Web mashups are Web applications generated by combining content, presentation, or application functionality from disparate Web sources. They aim to combine these sources to create new useful applications. Early studies [12] of Web developers and their experiences with building mashups found that the majority of developers created map mashups, with some creating photo mashups. This has changed over time and environments such as iGoogle have shown that mashups can be used for building a variety of composite applications. These studies also pointed out a number of issues with mashup environments: frustration when dealing with application programming interfaces, documentation, and coding details.

While a number of commercial and research environments to support the development and use of mashups exist, we note that they usually require specific tools and languages for their application or they have limited applicability. Even approaches such as Yahoo Pipes[1], which provides graphical mashup platforms aiming for a partially assisted Web development for less skilled programmers, still requires some effort and technical knowledge in order to be applied. Our goal is to overcome such limitations through an End User Development (EUD) [6] environment for Web mashups, which allows even people who have not software development as their primary

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

task to create their novel, composite Web applications. For this purpose we propose an environment that does not require the use of specific languages or the development of any specific model beforehand in order to enable the inclusion of parts of a Web application in the mashup. To this end, we simply require the use of a proxy/mashup server for the access to the Web applications that should be included in the mashup. The purpose of this server is to inject some scripts to enable the selection of the components that will be part of the newly created mashup. The authoring environment allows the creation of communication among components originally belonging to different applications. An example that we describe in detail in the paper is to take the search form from Amazon and connect it to multiple services in addition to those from Amazon (e.g. eBay, ...) so that one request can be sent simultaneously to all of them. Our environments aims to support the composition of the various types of components in Web applications: data, application logic, and user interface [11]. The user interface components maintain the original set of event handlers and are composed in terms of layout, data, and application logic (e.g. Web services). In order to obtain the new mashups no specific browser or plug-in is required, the involved applications have only to pass through a specific server. It is only needed the use of browsers able to support DOM serialization (e.g. Internet Explorer 9, Firefox 3.4 or higher, Chrome). The Mashup Enviroment is simply accessible through its Web address.

In the paper, after discussion of related work, we provide an example use scenario that shows the possibilities of our environment. Then, we describe the underlying architecture that makes such scenarios possible, we report on a first user test and, lastly, we draw some conclusions with indications for future work.

2 Related Work

d.mix [3] shares with our approach the use of a proxy server. In d.mix users select marked elements that they wish to copy. Through a site-to-service map d.mix composes Web service calls that yield results corresponding to the user's selection. This code is copied to a wiki for editing and hosting. Thus, d.mix still requires good programming knowledge for editing such code. A tool for rapid development of composite applications using annotated Web services [2] has been one of the tools developed in the ServFace project. This tool aims to support end user developers to build the front-end of existing services starting with the WSDL service operation descriptions, and exploiting Web services annotations providing suggestions for user interface development, when available. However, it is limited to create simple form-based user interfaces, and still requires some familiarity with Web services technology.

Collapse-to-zoom [1] is a technique for viewing Web pages on small screen devices by interactively removing irrelevant content selected through end user gestures performed through a pen on the mobile device screen. Differently, in our case we ask users to select the relevant content from different applications in order to build the mashup.

The existence of a tremendous amount of Web content, which is not always in a form that supports end users' needs has motivated Marmite [13], which allows users to select some operators, place them in a data flow representation, and view the current state of the data at a particular operator in a table, which shows what the data looks like after it has passed through an operator. However, Marmite is able to manage only a small set of pre-defined data types. In addition, it has been implemented as a Firefox plug-in, thus limiting its applicability to only this browser, while we propose a solution that is browser independent thanks to the use of an intermediate proxy/mashup server. Lin and others [5] have proposed a system using direct manipulation and programming-by-demonstration techniques to automatically populate tables with information collected from various Web sites. They have addressed a class of ad hoc mashups, where users want to quickly combine data from multiple Web sites in an incremental, exploratory fashion, often in support of a one-time or infrequently performed task. Their tool allow collecting a data table from anywhere and then running scripts that add columns to that table based on the data in each row while our tool allows users to combine functionalities from different Web sites and build new ones as well.

Some tools have been developed with the purpose of extracting code from Web pages but not with the goal of obtaining dynamic mashups. For example, Firecrow [7] aims to extract the code responsible for the desired behaviour of the selected Web UI control, but it is limited to extraction of the basic standard controls of Web applications. In the context of Web engineering, there also exist two related approaches and tools that facilitate the understanding of dynamic web page behaviour: Script InSight [4] and FireCrystal [8], which have a similar functionality of relating elements in the browser with the code responsible for them. We take this approach further since we enable including the components selected through their user interface directly in the mashup without having to deal with the underlying code.

3 An Example Application Scenario

In order to introduce our EUD mashup environment we describe an example application involving well-known applications.

In the scenario the user accesses Amazon through our proxy/mashup server and starts to search for books on funny t-shirts. After a while it occurs to the user that he may get better results through another application, such as eBay. Thus, he accesses eBay through the same intermediate server. Then, he notices that sometimes Amazon provides better results but in other cases eBay is more useful with the items shown. Thus, the user starts to think that it would be nice to have one single application able to show both results with a single query in order to speed-up the search. For this purpose he selects and sends the following components to the mashup editor: the Amazon search form, a result sample from Amazon and a result sample from eBay. The mashup editor asks the user to name and locate the widgets containing each of the three components selected and the display size associated with them in the mashup layout. Figure 1 shows the environment during this editing work. It is composed of three main elements: on the top-left part there is the mashup editor, which includes the parts selected and copied into it; on the right there is the eBay application highlighting in green the components that have been selected in the page presenting the search results; on the left-bottom there is the Amazon application in which the selected components (in this case the input form for the search) are highlighted.

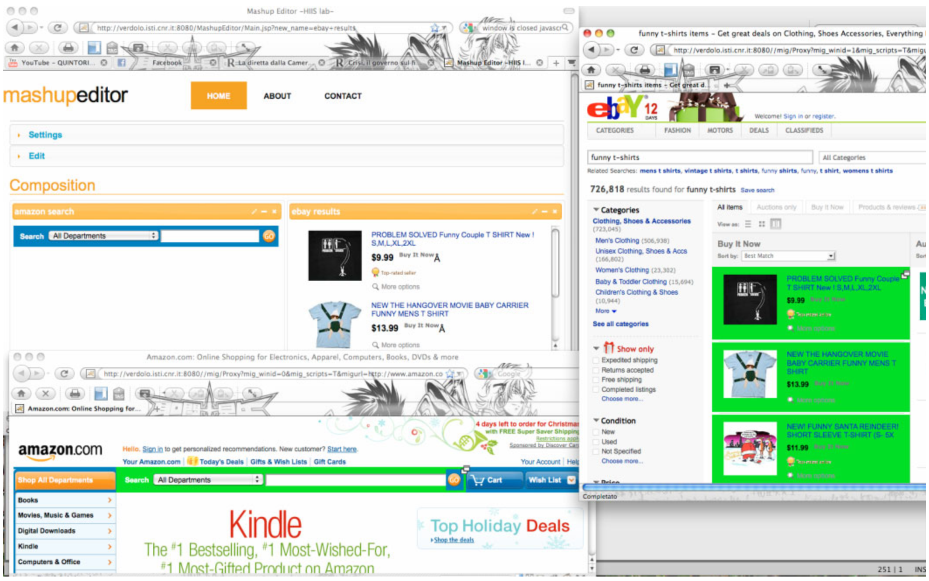
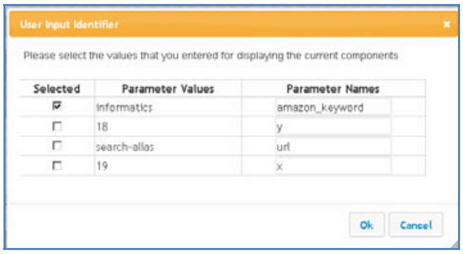

mashupeditor
HOME
ABOUT
CONTACT
Settings
Edit
Composition
amazon search
ebay results
PROBLEM SOLVED Funny Couple T SHIRT New ! S,M,L,XL,2XL
$9.99
NEW THE HANGOVER MOVIE BABY CARRIER FUNNY MENS T SHIRT
$13.99
amazon.com
Kindle
Top Holiday Deals
funny t-shirts
726,818 results found for funny t-shirts
Categories
Show only
Condition
Buy It Now
NEW! FUNNY SANTA REINDEER! SHORT SLEEVE T-SHIRT (S- 5X
$11.99

Fig. 1. The Mashup Editor and some Web components interactively selected

Afterwards, the user needs to create the connection between the input from the Amazon form and the eBay and Amazon results. To enable this connection the user has first to select each component presenting the results of a previous request to a Web server. For this purpose the mashup editor shows the list of parameters that have been initially used for generating the results (Figure 2 shows the case for Amazon).


User Input Identifier

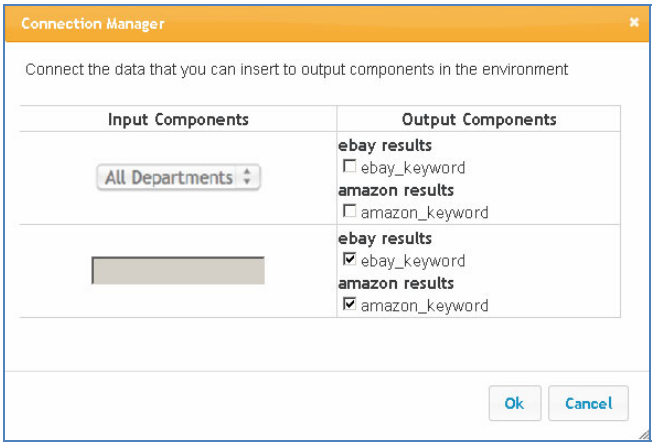
Please select the values that you entered for displaying the current components

Selected	Parameter Values	Parameter Names
☑	informatics	amazon_keyword
☐	18	y
☐	search-alias	url
☐	19	x

Fig. 2. The Dialog Box for selecting the parameter previously entered through an input field

The user only has to indicate which of them has been entered by himself through an input field. This information is useful for identifying a potential connection between this component and other components that provide user-generated input.

Then, the user activates the connections editor, which asks what input and output components to connect (see figure 3). In particular, the connection manager shows on the left-hand side the list of input elements that belong to the selected components, and, for each of them, on the right–hand side the output components that can be connected to a user input element. In our example the user selects the input text field of the Amazon form and then the Amazon and the eBay outputs resulting from a number of parameters.



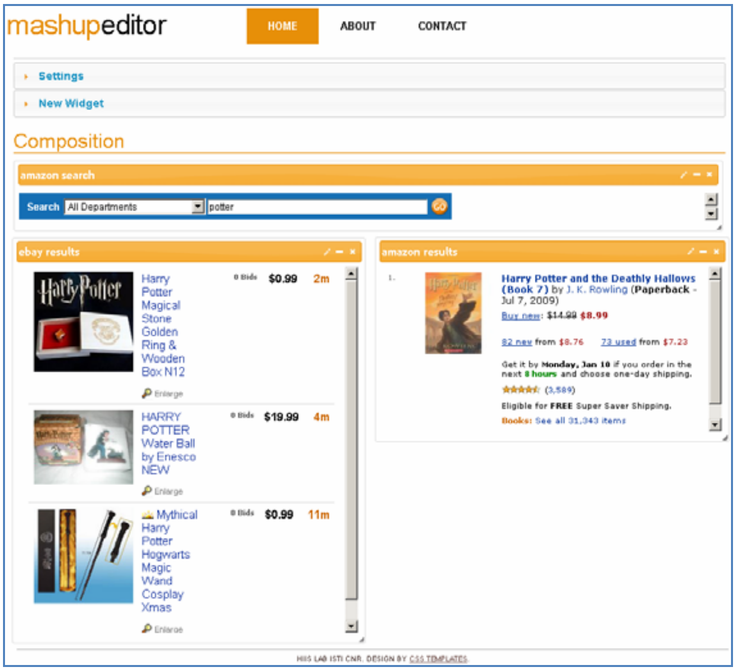
Connection Manager

Connect the data that you can insert to output components in the environment

Input Components	Output Components
All Departments	ebay results ☐ ebay_keyword amazon results ☐ amazon_keyword
	ebay results ☑ ebay_keyword amazon results ☑ amazon_keyword

Fig. 3. The Dialog Box allowing the user to associate input elements with output components

Figure 4 shows the resulting mashup working: in this case the user has entered the "potter" term and the results from both Amazon and eBay are shown. Please note that since only one result from Amazon and three from eBay were selected then the mashup shows only the first result from Amazon and the first three from eBay also in the further queries entered by the user.



Harry Potter Magical Stone Golden Ring & Wooden Box N12 — 0 Bids — $0.99 — 2m

HARRY POTTER Water Ball by Enesco NEW — 0 Bids — $19.99 — 4m

Mythical Harry Potter Hogwarts Magic Wand Cosplay Xmas — 0 Bids — $0.99 — 11m

1. Harry Potter and the Deathly Hallows (Book 7) by J. K. Rowling (Paperback - Jul 7, 2009)
Buy new: ~~$14.99~~ $8.99
82 new from $8.76 73 used from $7.23
Get it by Monday, Jan 10 if you order in the next 8 hours and choose one-day shipping.
(3,589)
Eligible for FREE Super Saver Shipping.
Books: See all 31,343 items

Fig. 4. The resulting mashup

4 The Architecture of the Environment

Our mashup environment is based on a server with twofold functionalities: it acts as a proxy and provides support for the authoring environment, including session management functionality for managing multiple users at the same time. Thus, the access to the Web applications should go through the proxy/mashup server, which parses the original document and adds the ids to those relevant elements that do not have one. The relevant elements are the ones that can be interactively selected (such as DIV, FORM, INPUT, TABLE, …). The server also includes a number of scripts, which are exploited by the mashup editor, to all the Web pages that are accessed through it. Such scripts allow the users to interactively select the elements to include in the newly created mashup. If the element is a container (e.g. a div) its inner components are recursively included into the selection. The elements selected in the Web page are highlighted by changing the background colour. To this end, the scripts that are automatically included are able to manage the on mouse over and on click events. If there are already some handlers for such events in the Web application, they are preserved and such behaviour is added to them. In particular, hovered components are gray-highlighted, selected components are green-highlighted. Highlighting of a component is done by changing its background and border colour. The original attributes value is saved on a JavaScript variable in order to be restored when the component is unselected or unhovered.

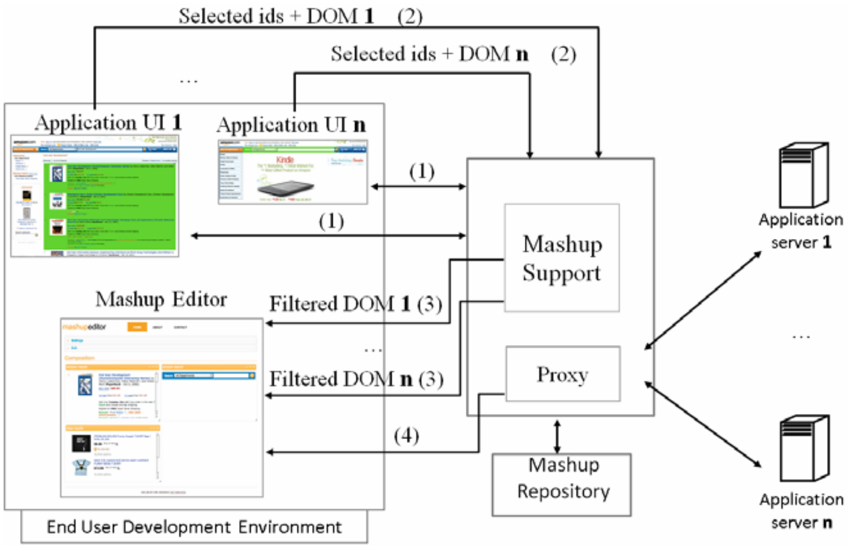


Fig. 5. The Underlying Architecture of the Mashup Environment

When the users are satisfied of the elements selected in a given application, they can send them to the mashup editor. In this case when the transmission is triggered the injected scripts send the ids of the selected elements along with the DOM description of the considered Web page to the mashup server (2 in figure 5). The mashup editor assumes that each DOM node has a unique id, and the proxy server supports this by first adding the missing ids to the original HTML page. The DOM sent to the mashup editor is state-persistent: this means that, for instance, it maintains the values previously entered by the user in a form. In the server the DOM is filtered removing all the elements that are not selected. The resulting filtered DOM is sent to the mashup editor (3 in figure 5), which allows the user to interactively determine the position and the dimensions of the set of new elements in the mashup layout. It also asks the user for a title to the newly added component. The list of the selected elements identified is also kept in the mashup server in order to allow the update of the components, as explained later on.

This type of selection of Web components and their inclusion in the mashup can be performed on various Web applications going through the same process.

The grid indicating the set of components in the mashup editor is stored in the associated session. In the stored information it is indicated where each component is located in the mashup layout; the path to the HTML file created in the mashup server, which is included in an internal frame; the list of parameters that the user has selected as possible values that can be provided by other components; the list of input parameters that the user can enter and that can be sent to other components.

In order to create communication among the groups of elements selected, the environment allows the identification of their communication needs. They are mainly classified into elements that require input from the user and can produce an output that can be sent to a Web server (input components), and elements that can receive input from the Web server and exploit such input to generate output towards the user (output components). It is also possible that one component has both input and output capabilities, but we will maintain the role distinction for the sake of clarity. The mashup environment is able to create connections between these two types of elements even if they originally belong to different applications. For this purpose, on user request, it is able to show the list of parameters that determine the content of output components showing query results and asks the users to select the elements that are the input that they entered to obtain them (the other parameters are generated by the application for its state management). Then, the user can activate a connection manager, which is able to allow the user to indicate what input components should be used to provide the parameters that determine the content of the output components.

The mashup environment is able to intercept the HTTP requests that in the case of a GET method use query strings and in the case of a POST method use post payloads. In both cases, the parameters are encoded according to a standard syntax (URL encoding), which enables its automatic analysis. Thus, by identifying all the parameters associated with an output component it is then possible to allow users to select any of them and indicate that the value entered in a given field in an input component should feed the selected input parameter. In order to perform such connection, the environment is able, on user demand, to show a list of the input fields that are contained into a mashup component (e.g. text fields, drop down lists, text areas etc.). For each field, the environment shows the list of input parameters for output components that can receive the value entered by the user through it. Figure 3 shows an example of such connection procedure: the input component contains two fields, a drop down list and a text field. The drop down list is not connected to any parameter, while the value entered in the text field is connected to the *amazon_keyword* of the *Amazon Result* component and to the *ebay_keyword* parameter of the *Ebay Result* component. This means that when the form containing the text field is submitted, the mashup editor updates the *Amazon Result* and the *Ebay Result* component, changing the value of the parameters in the query string associated to *amazon_keyword* and *ebay_keyword* respectively. For this purpose, through an analysis of the DOM, the action attributes of all forms are modified by the mashup server in order to submit to the mashup environment. The update procedure follows the following steps:

1. The mashup environment collects the value inserted by the user in the form;
2. For each value collected the components that should receive it are marked for update;
3. For each component that needs an update the new URL or POST content is calculated with the parameter value changed;
4. For each component that needs update, the mashup environment downloads in the server the full updated web page, filters the DOM using the id list created when the user selected the elements for creating the component, and uploads the result in the mashup editor.

The environment also allows the users to save the result of their mashup editing for further reuse by themselves or other users. This information is stored using the MARIA [9] logical language for interactive applications. The parameters are stored in the data model of the resulting MARIA specification. The layout is specified using a grid grouping composition operator of the language by defining the corresponding internal frames sources, width and height attributes.

One further advantage of saving the mashup specification in MARIA is that it is then possible to adapt it for access through mobile devices. For this purpose, it exists a tool for desktop-to-mobile adaptation [10], which performs an adaptation taking into account the screen size of the target device and other aspects.

5 Evaluation

A usability test has been carried out in order to evaluate the prototype of our mashup environment and, in general, to collect feedback from users.

12 subjects were involved (2 females, 10 males), with age ranging between 27 and 41 year old (average 30.3). The users were all recruited within the personnel of our research institute. However, among them there were not only researchers but also technical/administrative collaborators and university students. Furthermore, the participants were characterized by diverse educational level: 2 had PhD, 7 Master Degree and 3 Bachelor Degree.

With respect to the knowledge of programming languages, in a 1 to 5 scale (with 1 as most negative and 5 as most positive score), it varied between 2 and 5, (average 3.75, standard deviation 0.9).

It is worth noting that, as already mentioned, not all the users were equally experienced with programming. Four of them declared not to have any experience with the most common web-oriented languages (HTML / JavaScript). Furthermore, two users had only very little knowledge of C/C++ and some experience with Matlab. Only two users declared they had previously used a tool for creating web mashups (iGoogle).

The participants were required to interact with the mashup environment, through a desktop PC, for creating the mashup application introduced beforehand. In detail, they had to perform two searches: one in Amazon and one in eBay, respectively. Then, they had to select and move towards the mashup editor the Amazon search bar, a couple of Amazon results and one of the eBay results. Layout customization was then possible, and the users could resize and relocate the three widgets according to their preferences. At that point, the mashup consisted only in a presentation composition of Web components. One important part of the test was the binding between the input component (text field of Amazon search form) and the output components (eBay results and Amazon results). The binding is necessary to fully exploit the mashup environment by enabling the possibility to compose the functionalities of the components selected from the various Web applications.

Another relevant aspect of the test was to let the users notice the relationship between the originally selected component(s) and the ones resulting from the interaction with the mashup. At the end of the interaction, the participants were indeed requested to accurately observe the content of the widgets and to ensure about their correspondence with the parts extracted from the original pages. In detail, they had to look at the position of the search result items of Amazon and eBay with respect to the original page (this was possible by looking at the item index). Thus, besides the simplicity of the example, the users could realize the possibility of selecting additional and diverse parts of the original pages, such as the footer bar of Amazon or the left menu of eBay, to bring them to the mashup and to keep them while interacting with it.

The interaction took between 10 and 15 minutes per user, during which the number of errors was recorded together with their type, distinguishing between interaction errors and conceptual errors: general ones are related to errors in the interaction with the user interface (e.g., wrong selection in the source web page due to difficulties in moving the mouse pointer on the right element); conceptual ones are due to misunderstanding in the application of the mashup editing procedures (e.g., selecting only one variable in the binding table).

Nine users made at least one error. With respect to interaction mistakes, 5 users made 1 error each. With respect to conceptual mistakes, 3 users made 1 error each and other 3 users made 2 errors each.

Afterwards, the users filled in a questionnaire for rating several aspects of the mashup environment in a 1 to 5 scale (with 1 as most negative and 5 as most positive score). In the following, the rating for the intuitiveness/simplicity of the various tasks performed is reported ([min; max], average, standard deviation):

- Opening a new navigation window ([2; 5], 4.1, 0.9).
- Selecting the components on the navigated page and sending them towards the mashup tool ([3; 5], 4.3, 0.7).
- Instantiating and labeling the new widget ([3; 5], 4.7, 0.6).
- Rearranging (moving, resizing) the instantiated widgets in the mashup environment ([2; 5], 4.3, 0.9).

- Finding, in the connection panel of the mashup environment, the data previously filled in the form of the search page ([3; 5], 4.0, 0.8).
- Mapping the parameters of the two original applications in the connection panel ([2; 5], 4.2, 0.9).

Some further answers or indications were also provided in the questionnaire regarding particular aspects of the implemented prototype or with respect to recommended modifications/improvements. As weak point, regarding the interface for selecting and renaming the previously inserted parameters, one user stated that listing too many parameters could be confusing. He then suggested to automatically restrict, in some way, the parameters to be indicated in the connection table.

Another user declared that, using just the strings for identifying the parameters, can be difficult sometimes. Thus, she proposed the usage of visual techniques for recalling them.

Others suggested to further improve the support for the user, such as automatically pop-up the mashup editor window after one component is sent from a Web application to the mashup editor.

Even those users who initially had not well understood the actual meaning of the mashup editor, at the end of the interaction seemed to realize its capabilities and usefulness.

We investigated also the potential relationships between the programming experience of the users and the number/type of interaction mistakes. However, such relationships did not emerge from the test results. Indeed, we observed that general and conceptual mistakes occurred both among experts and non-experts users. At the same way we noticed that, among who did not make any error, there were users with high as well as with low programming experience. Relationships between experience and mistakes do not emerge even distinguishing the kind of user programming experience (i.e.: whether the user had familiarity with specific web-oriented programming languages). Thus, due to this apparent lack of relationships, we are prone to assert that the user attitude towards our mashup environment does not depend on the user programming experience.

6 Conclusions and Future Work

Despite the enormous number of Web applications often users need to create new composite applications that better suit their needs. In this paper we have presented the motivations and the design of a new solution that aims to allow even people who are not professional programmers to create their Web mashups. We have described an example application, the underlying architecture and the implementation of a prototype system, which has already been tested by a set of users. We also report on this first user test, which has provided some useful suggestions, for example to provide some graphical metaphor to make more intuitive the connection mechanism among components.

The interesting point of our approach is that it allows users to create the mashups through direct manipulation of existing Web applications without requiring the use of a specific browser and the knowledge of any programming language. This is obtained by automatically annotating the Web pages that are included in the environment by some scripts when they pass through our server. We are considering changing the technique used for identification of the various Web page elements that can be selected by the users in order not to relay on DOM nodes ids. A solution based on the XPath node should be more general.

We also plan to perform more systematic user validation of the proposed environment by involving a wider set of user groups, possibly without any development experience.

References

1. Baudisch, P., Xie, X., Wang, C., Ma, W.: Collapse-to-zoom: viewing web pages on small screen devices by interactively removing irrelevant content. In: UIST 2004, pp. 91–94 (2004)
2. Dannecker, L., Feldmann, M., Nestler, T., Hübsch, G., Jugel, U., Muthmann, K.: Rapid Development of Composite Applications Using Annotated Web Services. In: ICWE Workshops 2010, pp. 1–12 (2010)
3. Hartmann, B., Wu, L., Collins, K., Klemmer, S.R.: Programming by a sample: rapidly creating web applications with d.mix. In: UIST 2007, pp. 241–250 (2007)
4. Li, P., Wohlstadter, E.: Script insight: Using models to explore JavaScript code from the browser view. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 260–274. Springer, Heidelberg (2009)
5. Lin, J., Wong, J., Nichols, J., Cypher, A., Lau, T.A.: End-user programming of Mashups with Vegemite. In: IUI 2009, pp. 97–106 (2009)
6. Lieberman, H., Paternò, F., Wulf, V. (eds.): End User Development (2006) ISBN: 978-1-4020-4220-1
7. Maras, J., Štula, M., Carlson, J.: Extracting Client-Side Web User Interface Controls. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 502–505. Springer, Heidelberg (2010)
8. Oney, S., Myers, B.: Firecrystal: Understanding interactive behaviors in dynamic web pages. In: VLHCC 2009: Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 105–108. IEEE Computer Society, Los Alamitos (2009)
9. Paternò, F., Santoro, C., Spano, L.D.: MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16(4) (2009)
10. Paternò, F., Zichittella, G.: Desktop-to-Mobile Web Adaptation through Customizable Two-Dimensional Semantic Redesign. In: HCSE 2010, pp. 79–94 (2010)
11. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development. IEEE Internet Computing 12(5), 44–52 (2008)
12. Zang, N., Rosson, M.B., Nasser, V.: Mashups: who? what? why? In: CHI Extended Abstracts 2008, pp. 3171–3176 (2008)
13. Wong, J., Hong, J.I.: Making mashups with Marmite: towards end-user programming for the Web. In: CHI 2007, pp. 1435–1444 (2007)