DashMash: A Mashup Environment for End User Development

Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega, Donato Barbagallo, and Chiara Francalanci

Politecnico di Milano - DEI
P.zza Leonardo da Vinci, 32 - 20133 - Milano - Italy
{name.surname}@polimi.it

Abstract. Web mashups are a new generation of applications based on the "composition" of ready-to-use services. In different contexts, ranging from the consumer Web to Enterprise systems, the potential of this new technology is to make users evolve from passive receivers of applications to actors actively involved in the "creation of innovation". Enabling end users to self-define applications that satisfy their situational needs is emerging as an important new requirement. In this paper, we address the current lack of lightweight development processes and environments and discuss models, methods, and technologies that can make mashups a technology for end user development.

Keywords: mashups, service-based dashboards, end user development.

1 Introduction

Web mashups are a new generation of tools that support the "composition" of applications starting from services and contents oftentimes provided by third parties and made available on the Web. Mashups were initially conceived in the context of the consumer Web, as a means for users to create their own applications starting from public programmable APIs or contents taken from Web pages. However, the vision is towards the development of more critical applications, for example the so-called *enterprise mashups* [10], a porting of current mashup approaches to company intranets. The potential flexibility of mashup environments can help people help themselves [20], by enabling the on-demand composition of the functionalities that they need. Mashups are therefore emerging as a technology for the creation of innovative solutions, able to respond to the different problems that arise daily in the enterprise context, as well as in any other context where flexibility and task variability are dominant requirements.

Given the previous premises, the need arises to provide mashup environments where the end users (i.e., the main actors of this new development process) can easily and quickly self-construct their applications without necessarily mastering the technical features related to service invocation and integration. This is true in every context - not only in the Enterprise: a "culture of participation" [8], in which users evolve from passive consumers of applications to active co-creators of new ideas, knowledge, and products, is indeed more and more gaining momentum

S. Auer, O. Diaz, and G.A. Papadopoulos (Eds.): ICWE 2011, LNCS 6757, pp. 152–166, 2011.

1.1 Contributions and Paper Outline

What makes mashups different from plain Web service compositions is their potential as tools through which end users are empowered to develop their own applications. However, this potential is rarely exploited. So far the research on mashups has focused on enabling technologies and standards, with little attention on easing the mashup development process - in many cases mashup creation still involves the manual programming of the service integration. Some recent user-centric studies [14] also found that, although the most prominent mashup platforms (e.g., Yahoo!Pipes, Dapper or Intel Mash Maker) simplify the mashup development, they are still difficult to use by non technical users. In this paper, we try to respond to the need of easying the mashup development, and propose a Web platform, *DashMash*, that allows end users to develop their own mashups making use of an intelligible paradigm that abstracts from technical variables. In particular:

1. Through a case study, we provide a scenario in which general mashup composition principles can be applied to the construction of applications targeting the experts (e.g., analysts and decision makers) of a given domain (Section 2). Based on this scenario, we outline relevant factors supporting end user development. First of all, the importance of a *lightweight development process*, in which mashup composition paradigms are embedded in usable visual environments hiding the complexity of the composition languages actually managing the execution of the mashup.
2. We present a runtime architecture supporting the lightweight development processes, which increases the user's control over the mashup composition process (Section 3). This is possible thanks to (i) an instant execution support, based on the "on the fly" interpretation of the user composition actions and the immediate execution of the mashup composition in a WYSIWYG (What You See Is What You Get) manner (Section 3.2), and (ii) the automatic generation of descriptive models for the results of the users composition actions (Section 3.3), which then drive the execution of the mashup.
3. We promote the adoption of *recommendation mechanisms* that take into account quality variables to help end users select data sources and mashup components and composition patterns (Section 3.3). This is enabled by the enrichment of descriptive models with annotations specifying also non-functional user requirements.
4. Based on the results of a usability experiment, we discuss the effectiveness of our approach from an end user perspective (Section 4).

2 Case Study: Mashups for Sentiment Analysis

In the context of a project funded by the Comune di Milano (Milan Municipality), we have worked on the construction of a Web platform through which end users can construct their dashboards for *sentiment analysis*[1]. Sentiment analysis

[1] A demo is available at http://home.dei.polimi.it/cappiell/demo/DemoDashMash.mov

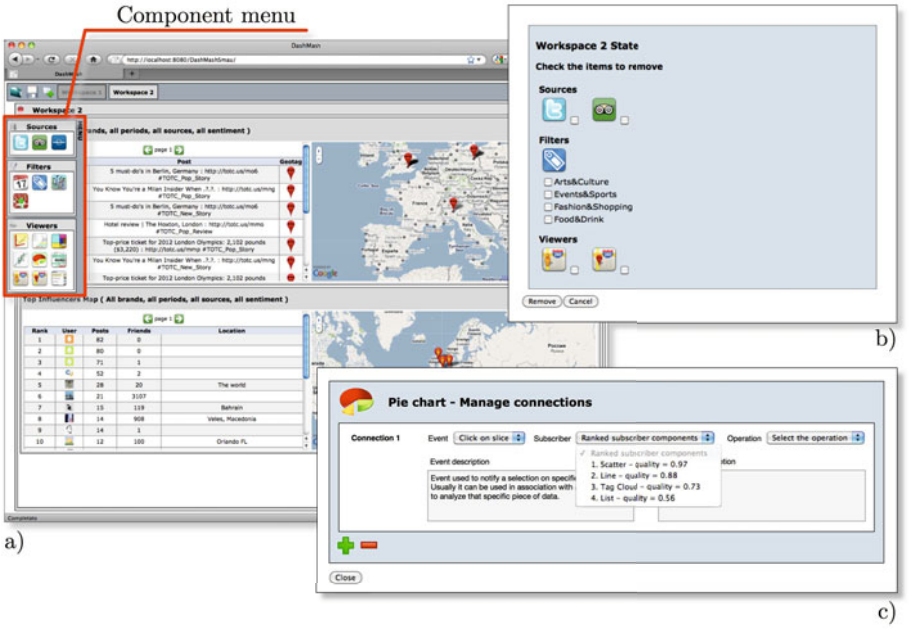

Fig. 1. Example of mashup composition for a sentiment analysis dashboard

focuses on understanding market trends starting from the unsolicited feedback provided by users comments published on the Web. Our project focuses on the design of an engine in charge of the automatic extraction of sentiment indicators summarizing the opinions contained in user generated contents [1], and on the provision of a Web environment where analysts can self-construct their analyses.

After preliminary attempts with a traditional static dashboard, we realized that end users could benefit from the ability to compose their analysis flexibly, playing in variable ways with sentiment indicators, and also complementing such indicators with "generic" external Web resources. This latter feature would indeed help them to improve their analyses, by interpreting sentiment indicators with a view on the events that cause trends and behaviors.

Figure 1 shows an example of use of the Web front-end of our platform, DashMash. A left hand menu presents the list of components, which for the sentiment analysis domain are *data sources* that materialize contents extracted from community sites, several types of *filters*, a multiplicity of *viewers* to visualize data, which are both open APIs, e.g, the Google APIs for maps and charts, and ad-hoc developed services[2], and utility open API/services, such as RSS feeds and calendars. Components can be mashed up by moving their corresponding icons into the so-called *workspaces*. Each workspace is associated with a data set

[2] Several charts offering advanced data visualizations have been developed using the Highcharts JS library (http://www.highcharts.com/).

resulting from the integration of data sources and filters, and renders this data set according to the visualizations offered by the selected viewers.

In the mashup shown in Figure 1a) the user has selected two data sources storing users comments extracted from two well-known social applications, Twitter and TripAdvisor. A filter is applied to select the only comments from users that are considered opinion leaders, so-called *influencers*. Influencers data are visualized through a *list viewer*, which is integrated with *Google Maps* to show the influencers locations. A further synchronization with another map and another list viewer allows one to see the original posts of each influencer, as well as the geo-localization of their posts, if available.

Users can iteratively modify the composition, by adding or dropping components. Changes are enacted at real time and the effect are immediately shown. They can also access a visual description of the status of the current composition (see Figure 1b), and easily modify sources, filters, viewers or even configuration properties of each single component.

Although the system automatically includes some default bindings to ensure basic inter-component synchronization, through simple dialog boxes the users can create new service combinations resulting into synchronized behaviors. For example, starting from the mashup shown in Figure 1a, the dialog box presented in Figure 1c allows the user to add a pie-chart viewer to show the distribution of influencers comments along different topics, and set a coupling so that a click on a pie slice contextualizes the analysis offered by the map viewer to that selected portion of data. Also, based on compatibility rules and quality criteria, the system provides suggestions about other candidate components to extend the mashup or replace existing components.

As shown by the previous example, the DashMash environment is characterized by factors that try to alleviate as much as possible the end users during the mashup composition task [14,8]:

- *Abstraction from technical details*: the representation of services as visual objects that abstract from technical details (e.g., their programmatic interface), the immediate feedback on composition action, and the immediate execution of the resulting mashup to reveal the service look&feel, help users realize the service functionality and the effect that the service has on the overall composition. In the end, users are asked to manipulate (e.g., add, remove or modify) visual objects focusing on the service visualization properties rather than technical details of service and composition logics. As also confirmed by a user-based experimentation (see Section 4), this increases user satisfaction and the user-perceived control over the composition process.
- *Composition support*: the composition task is guided in multiple ways. On the one hand, starting from components' descriptive models, the composition engine is able to infer and automatically create *default bindings* between the services that the users add to the composition. Compatibility rules and quality criteria are then adopted to "rank" available components and provide users with suggestions about *additional components* and *custom bindings*.

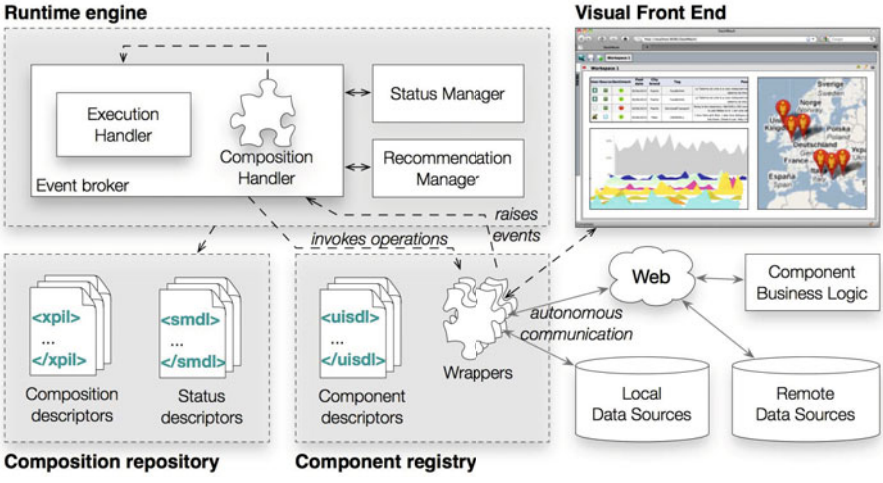

Fig. 2. Organization of the DashMash architecture

– *Continuous monitoring*: users are provided with mechanisms that allow them to understand the current state of the composition and to explore options about how to complete or extend the current composition.

The following sections are devoted to clarifying the modelling abstractions and the architectural features that allowed us to implement the previous requirements in DashMash.

3 The DashMash Platform

As represented in Figure 2, the organization of DashMash is centered around a lightweight paradigm in which the orchestration of registered services, the so called *Components*, is handled by an intermediary framework, in charge of managing both the *definition* of the mashup composition and the *execution* of the composition itself. Different from the majority of mashup platforms, where mashup design is separate from mashup execution, in DashMash the two phases strictly interweave. The result is that composition actions are automatically translated into models describing the composition; these models are immediately executed. Users are therefore able to *interactively* and *iteratively* define and try their composition, without being forced to manage complicated languages or even ad-hoc visual notations.

The current implementation of the DashMash runtime engine consists of a client-side (JavaScript) application that supports an *event-driven* execution paradigm. As represented in Figure 2, an *Event Broker* intercepts events, which can refer to users and system actions occurring during mashup execution (e.g., the click on a pie-chart slice which cause a change in a map), and to the dynamic definition of the composition (e.g., the drag&dop of a component icon into a workspace). The Event Broker then dispatches the events to the modules

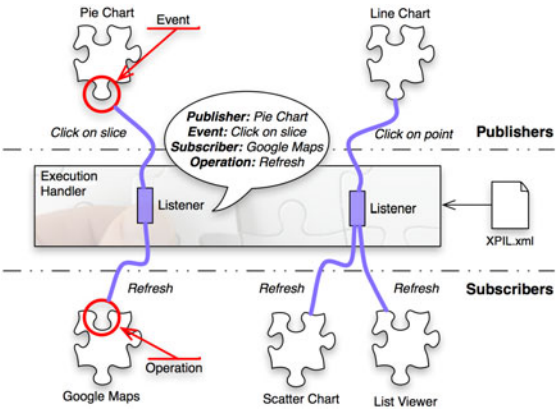

Fig. 3. Event-driven paradigm for service coupling definition and mashup execution

in charge of their handling, based on models and mechanisms that we explain in the rest of this section.

3.1 Event-Driven Execution

Events occurring during mashup execution are managed by an *Execution Handler* based on a *publish-subscribe model* addressing component integration at presentation level [21]: *events* generated from the user interaction with one mashup component (e.g., the selection of a slice in a pie chart) can be mapped to *operations* of one or more components that subscribe to such events (e.g., the visualization of details of the selected data in a scatter plot).

As represented in Figure 3, service couplings are expressed through the definition of the so-called *listeners* in a *composition model* expressed according to an XML-based language, *XPIL* [21]. During mashup execution, each component keeps running according to its own application logic, within the scope defined by an HTML `<div>`. As illustrated in Figure 3, as soon as events occur, the involved components publish them. Based on the defined listeners, the Execution Handler then notifies the subscribed components, if any, and triggers the execution of their corresponding operations.

This composition and execution logics requires each component to be characterized by a high-level model expressing the *events* that the component can generate, the *operations* that enable other components to modify its internal state, and the binding with the actual service/API, so that operations can be invoked. Therefore, as illustrated in Figure 2, for each registered component the component registry stores a *descriptor*, expressed according to the *UISDL* language [21], and a *wrapper*, in charge of rasing events and invoking operations. This component model provides a uniform paradigm to coordinate the mashup composition and execution, which obviates the heterogeneity of service standards and formats, also hiding the intrinsic complexity of services and composition.

3.2 Managing Composition

The *Composition Handler* manages composition events. In particular, as better explained in Section 3.3, it automatically translates the addition of a component into new *default listeners* and creates or updates (if already existing) the current composition model accordingly. The Composition Handler in turns dispatches composition events to the *Status Manager*, a module in charge of maintaining a *mashup state* representation. Combined with the composition model, the mashup state is useful to recover a previously defined mashup for a later execution, but especially to let users monitor their composition and modify it on the fly. State variables relate to default or specific parameter values (e.g., the value of a parameter for querying a data source), to layout properties (e.g., the colors used to show values on a chart) or to any other property that the user can set to control the component data and appearance. Composition events are also dispatched to the *Recommendation Manager*, a module in charge of providing suggestions about further components to select for extending/completing a composition (see Section 3.3). After terminating the handling of such events, the mashup composition is reloaded and immediately rendered into the workspace. The mashup is then executed according to the event-driven, publish-subscribe logic that characterizes the Execution Handler.

It is worth noting that the Composition Handler itself is a mashup component: any user composition action generates a Composition Handler's event, which is notified to and managed by the Execution Handler. An interesting side effect of this architectural choice is that the logic behind the automatic component coupling is "programmable" and, therefore, flexible: it depends on a set of predefined listener templates configured inside the Composition Handler which, being the Composition Handler a mashup component, can in turn be easily unplugged and/or replaced.

3.3 Definition of Listeners

DashMash supports the definition of *default* and *custom* listeners. The former are automatically defined by the Composition Handler when a composition action is intercepted. This ensures a minimum level of inter-component synchronization that does not require users to define service coupling. Custom bindings are instead user-defined. Nevertheless, the Composition Handler offers also support by generating compatibility– and quality– based recommendations. To this aim, it dispatches the composition events to the *Recommendation Manager* that is in charge of evaluating the quality of the current composition and provides suggestions about the selection of possible components to add or substitute to the existing ones in order to achieve or improve the mashup quality.

Default Listeners. To enable the automatic definition of default listeners, we start from a classification of components. For example, in order to facilitate the construction of dashboards, it is possible to identify the following classes of components (Figure 4):

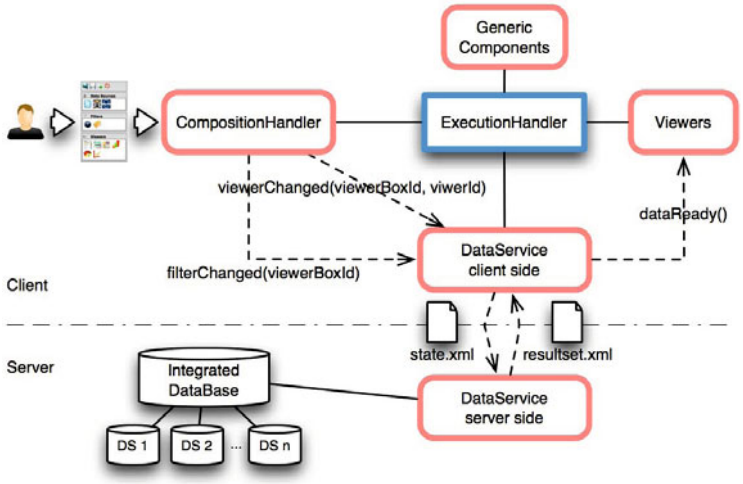

Fig. 4. Classification of components and their mutual interactions

- *Data services* are in charge of retrieving data from data sources. They are especially meant to provide access to internal (relational) data sources/ warehouses. For example, in the sentiment analysis domain internal data sources store data extracted from different social applications where users post their comments. A server-side module is in charge of accessing the data sources[3]. A client-side module provides the actual component that synchronizes with the other mashup components to intercept changes of the composition state and send pertinent information to the server-side module so that queries for data extraction can be constructed.
- *Filters* add selection conditions over the context defined by a workspace (e.g., an interesting keyword or a time interval specified through a calendar component), thus filtering the mashup result set.
- *Viewers* support the visualization of result sets, which can be extracted by data services from internal sources or from generic external resources. Since data visualization usually involves some form of aggregation, most viewers embed a transformation logic. Therefore viewers can be simple tables, any kind of graph (as for example, those provided by Google Charts), and any visualization/aggregation service that is useful for the specific domain (e.g., a tag cloud or a map).
- *Generic components* can be also integrated to make the analysis process more effective. They can provide a variety of functionalities, such as video or image retrieval through the most common APIs, or further data sources (e.g., RSS Feeds), which can complement the information extracted from the corporate data sources.

[3] When the integration of multiple sources is required, this module could embed the needed integration logics or alternatively make use of a dedicated integration layer.

The previous component classification can be adopted in several mashup contexts. For example JackBe Presto also exploits a similar classification of *mashables* [7]. What is new in our approach is that this classification of services allows us to codify default data flows that the platform exploits to automatically include ad-hoc listeners in the composition model. For example, viewers always "consume" data, as they elaborate and visualize data extracted by other components, for example data services. The addition of a viewer into the composition therefore requires at least the inclusion of a listener to subscribe the viewer to a *dataReady* event exposed by the component that produces data.

Figure 4 highlights the synchronization managed through the main default listeners. Any time a viewer is added into the workspace, the Composition Handler publishes an event that triggers a Data Service Client operation that sends a pertinent portion of the updated state to the Data Service Server. Based on the exchanged state information, the Data Service Server queries the workspace data set and sends the result set back to the Data Service Client. Based on another default listener, the Data Service Client raises an event so that subscribed viewers know about the new result set and, thus, refresh their state.

The "knowledge" about possible default mappings is coded inside the Composition Handler and can be easily configured, with the advantage that DashMash can be easily adapted to domains with possibly different services and service classifications. DashMash can also work without classification, allowing users to couple components by means of custom listeners definition.

Custom Listeners and Quality-based Recommendations. One peculiarity of DashMash with respect to other mashup platforms is the emphasis on quality aspects to support the users in the mashup construction. In particular, we have identified two phases where quality issues must be taken into account: (i) the registration into the platform of new services and (ii) the generation of recommendations during the mashup composition.

Registration of new services. As highlighted in Figure 2, the set of components C available in the DashMash platform is composed of services that can be created ad-hoc or can be public. Independently on the component nature, when a large amount of functionally equivalent data sources and services are available, quality can be one relevant driver for the selection of the most dependable components. We therefore adopt a quality evaluation approach for estimating the quality of components, which focuses on both the quality properties of mashup components and the reliability (i.e., reputation) of the Web information sources accessed to retrieve the mashup data:

– The *component quality* refers to the quality model proposed in [4], which specializes traditional quality dimensions (such as functional efficiency, reliability and usability of the APIs, and data and presentation quality criteria) to the peculiar context of mashup composition.

– The *reputation of data sources* refers to the model presented in [2], which specifically addresses the trustworthiness of Web 2.0 contents, and in particular those deriving from blogs and community sites.

When a new component is added to the DashMash component registry, the quality data needed to determine the quality indices prescribed by the two adopted quality models are documented as annotations to the component descriptors. Each component $c_i \in C$ is therefore associated with a *component descriptor* and a *quality data vector*. The component descriptor lists all the operations and the events, plus the technical details needed to compute quality indices. The quality data vector, $QD_i = [qd_{i1}, qd_{i2}, \cdots, qd_{in}]$, contains the list of measures for the component quality CQ_i and source quality SQ_i as aggregated quality indexes for the i-th component and its associated data source, respectively.

Component selection during mashup composition. When a component is added into a mashup under construction, recommendations about further components to be added to the composition are generated. In particular, the Composition Handler dispatches composition events to the *Recommendation Manager* that, starting from the quality indexes stored in the component registry, suggests actions to achieve or improve the quality of the mashup. To do so, we adopt the approach described in [16], in which components are ranked on the basis of their *mashability* [16]. Mashability is defined as the capability of a component (i) to be combined with previously selected components and (ii) to maximize the quality of the overall mashup. Thus, it can be seen as a combination of two dimensions: component compatibility and aggregated quality.

Component compatibility estimates whether a component can be coupled with those already included in a composition and distinguishes between *syntactic compatibility*, checking the compatibility between input/output parameters exposed by the components, and *semantic compatibility*, checking whether input/output parameters and operations belong to the same or similar semantic categories, assuming that syntactic compatibility is satisfied. The compatibility index $comp_i$ provides a preliminary measure of the compatibility of a service c_i to be added to a given mashup: a value equal to zero indicates the incompatibility from a syntactic point of view while a positive value provides a measure of semantic compatibility. This index is stored in form of a matrix, where events and operations of the available components are related to each other and their compatibility is scored. The matrix is updated every time a new component is registered to the component registry. During mashup composition, the Recommendation Manager accesses this matrix to determine the list of components that can be suggested to the user as they can be correctly coupled with the components already in the composition. For example, in Figure 1c, only the compatible components are shown in the drop-down list where the user can select a new component to add.

While compatibility ensures the construction of "correct" mashups, *aggregated quality* drives the user in the choice of components that can lead to quality mashups. It estimates the quality of the mashup under construction as a composition of the quality of individual components. The mashup quality QM_k cannot

be however quantified as a simple sum of the quality of individual components, CQ_i, but it is necessary to weigh quality indices by taking into account the role and the importance of each component [5]. Therefore, starting from the composition model, we take into account the in- and out-degree of each component in the composition graph, and use them to determine the component impact on the overall composition, so weighing the aggregated quality. As shown in Figure 1c, aggregated quality values are visualized for each compatible components suggested in the drop-down list.

4 User-Based Validation

In order to validate the composition paradigm of DashMash with respect to end user development requirements, we conducted a user based study. We observed domain experts and naive users completing a set of tasks through our platform. Our goal was to assess how easily the users would be able to develop a composite application. The experiment specifically focused on the efficiency and intuitiveness of the composition paradigm, trying to measure such factors in terms of user performance, ease of use and user satisfaction. In particular, we expected all users to be able to complete the experimental tasks. However, we expected a greater efficiency (e.g., reduced completion task times) and a more positive attitude (in terms of perceived usefulness, acceptability and confidence with the tool) by expert users. Their domain knowledge and background could indeed facilitate the comprehension of the experimental tasks, and improve the perception of the control over the composition method, and thus, their general satisfaction.

The study involved 35 participants. Six of them were real end users, i.e., analysts and decision makers that are supposed to actually use DashMash for their analyses in the sentiment analysis domain. In order to prove to which extent the tool was intuitive even for naive users, we also involved undergrad students of the Computer Engineering Programme at Politecnico di Milano, with a moderate knowledge about Web technologies, but never exposed neither to our tool nor to the sentiment analysis domain.

For novice users, the completion of the experimental tasks was preceded by a 5-minute explanation about the domain and about the basic composition actions supported by the tool. Expert users were instead introduced to the set of available components and the basic composition mechanisms. All users were first asked to fill in a pre-test questionnaire, to gather data on their knowledge about services and mashups. All users were then asked to perform two composition tasks. The two tasks were comparable in terms of number of components to be integrated and composition steps. Task 2, however, required a less trivial definition of filters, to sift the involved data sources, and a more articulated definition of bindings. Also, while the formulation of task 1 was more procedural, i.e., it explicitly illustrated the required steps, task 2 just described the final results to be achieved, without revealing any details about the procedure required.

After the completion of the two experimental tasks, users were then asked to fill in a satisfaction questionnaire.

4.1 Results

All the participants were able to complete both tasks without particular difficulties. No differences in task completion time were found between experts and novices. In particular, domain expertise was not discriminant for task 1 ($p = .085$) and for task 2 ($p = .165$). Similarly, technology expertise was not discriminant for task 1 ($p = .161$) and for task 2 ($p = .156$). The lack of significant differences between the two groups does not necessarily mean that expert users performed bad. However, it indicates that the tool enables even inexperienced users to complete a task in a limited time. The average time to complete task 1 was about 2.5 minutes, while for task 2 it was less then 2 minutes.

The difference in completion times for the two tasks can be also used as a measure of learning [9]. This difference is about half a minute ($t = 28.2, p = .017$), i.e., a reduction of about 15%. This result highlights the learnability of the tool: although the second task was more critical compared to the first one, subjects were able to accomplish it in a shorter time.

The ease of use was also confirmed by the data collected through four questions in the post-questionnaire, asking users to judge whether they found it easy to identify and include services in the composition, to define service bindings between services, and to monitor and modify the status of the mashups. We also asked users to score the general ease of use of the tool. Users could modulate their evaluation on a 7-point scale. The reliability of the ease of use questions is satisfying ($\alpha = .75$). The correlation between the four detailed questions and the global score is also satisfying ($\rho = .58, p < .001$). This highlights the high external reliability of the measures. On average, users gave the ease of use a mark of 1.77 (the scale was from 1 very positive to 7 very negative). The distribution ranged from 1 to 4 ($mean = 1.77, meanS.E. = .12$). We did not found differences between novice and expert users. This was especially true for perceived usefulness ($p = .51$).

The user satisfaction with the composition paradigm was assessed using two complementary techniques. A semantic-differential scale required users to judge the method on 12 items. Users could modulate their evaluation on a 7-point scale (1 very positive - 7 very negative). Moreover, a question asked users to globally score the method on a 10-point scale (1 very positive - 10 very negative). The reliability of the satisfaction scale is satisfying ($\alpha = 0.76$). Therefore, a user-satisfaction index was computed as the mean value of the score across all the 12 items. The average satisfaction value is very good ($min = 1.3, max = 3.5, mean = 2.2, meanS.E. = .09$). The correlation between the average satisfaction value and the global satisfaction score is satisfying ($\rho = .41, p < .015$). On average, users gave the composition method a mark of 2.9, with a distribution ranging from 2 to 4. We did not find differences between experts and novices. Despite our initial assumption, we found that the ease of use of the tool is perceived in the same way by novice and expert users, although the latter have greater domain knowledge. The moderate correlation between the satisfaction index and the ease of use index ($\rho = .55, p = .011$) also reveals that who

perceived the method as easy also tended to evaluate it as more satisfying. This confirms that ease of use is perceived.

The last two questions asked users to judge their performance as mashup developers and to indicate the percentage of requirements they believed to have satisfied with their composition. This metric can be considered as a proxy of confidence [9]. On average, users indicated to be able to cover the 91% of requirements specified by the two experimental tasks ($min = 60\%, max = 100\%, meanS.E. = 1.7\%$). They also felt very satisfied about their performance as composers ($mean = 1.8, meanS.E. = .13$; 1 - very positive, 4 - very negative).

5 Related Works

Service composition has been traditionally covered by powerful standards and technologies (such as BPEL, WSCDL, etc.), which however can be mastered by IT experts only [17]. According to the End User Development vision, enabling a larger class of users to create their own applications requires the availability of intuitive abstractions and easy development tools, and a high level of assistance [3,12]. Some projects (e.g., Dynvoker [19], SOA4All [11]) have focused on easying the creation of effective presentations on top of services, to provide a direct channel between the user and the service. However, very often such approaches do not allow the composition of multiple services into an integrated application.

Mashups are emerging as an alternative solution to service composition that can help realize the dream of a *programmable Web* [13], approachable even by non-programmer users. There is a considerable body of research on mashup tools, the so-called *mashup makers*, which provide graphical user interfaces for combining mashup services, without requiring users to write code. Among the most prominent platforms, Yahoo!Pipes (`http://pipes.yahoo.com`) focuses on data integration via RSS or Atom feeds, and offers a data-flow composition language. JackBe Presto (`http://jackbe.com`) also adopts a pipes-like approach for data mashups, and allows a portal-like aggregation of UI widgets (mashlets). IBM DAMIA [18] offers support to quickly assemble data feeds from the Internet and a variety of enterprise data sources. Mashart [6] focuses on the integration of heterogeneous components, offering a mashup design paradigm through which users create graph-based models representing the mashup composition. Marmite [20] is specifically tailored for accessing information sources: it offers a more intuitive paradigm to easily program and chain a set of operators for filtering sources. Operators provide an intelligible mechanism for extracting contents from data sources (especially from Web pages); however, the operator composition is still constrained to a parameter coupling logic based on manual type-matching.

With respect to manual programming, the previous platforms certainly alleviate the mashup composition tasks. However, to some extent they still require an understanding of the integration logic (e.g., parameter coupling). In some cases, building a complete Web application equipped with a user interface requires the adoption of additional tools or technologies. Even when they offer an easy composition paradigm, as it happens for example for Intel Mash Maker

(`http://mashmaker.intel.com`), they do not guide at all the composition process, as we do in DashMash through quality-based recommendations. A study about users' expectations and usability problems of a composition environment for the the ServFace tool [14] shows that there is evidence of a fundamental issue concerning conceptual understanding of service composition (i.e., end users do not think about connecting services). Also, in given contexts, the openness towards any kind of services and the full set of functions these tools are able to provide are not actually required. *Sandbox* environments, like the one presented in this paper, where ready-to-use services are composed according to a "constrained" integration logic, can help achieve a lightweight, easy to use composition [15], which is especially effective to serve well-defined situational purpose [10].

6 Conclusions

This paper has presented a mashup platform that leverages a composition paradigm aiding end-user development. Our platform is particularly effective for enterprise mashups, where the application domain and the component characterization are easier to identify and represent in form of rules for the automatic definition of service coupling. However, the approach remains valid for the composition of *generic* mashups. Of course, the lower the availability of domain descriptions, the greater the need for users to explicitly define service couplings. Nevertheless, as confirmed by the user-based study, the coupling definition mechanism is still intuitive and further facilitated by quality-based recommendations.

As future work, we aim at exploring different composition solutions, to address, for example, the cooperative definition of mashups (a feature that can greatly enhance team-based cooperation), also enabling mashup creation and fruition on mobile device, as well as an extension of the recommendations mechanisms based on the emergence of composition patterns from the community's mashups [16]. We also aim at easing the creation by users of DashMash components. First preliminary results concerning described services (i.e., WSDL and Linked Data) are encouraging. Our future will be also devoted to improving this feature.

References

1. Barbagallo, D., Cappiello, C., Francalanci, C., Matera, M.: A reputation-based dss: the interest approach. In: Proceedings of ENTER 2010 (2010)
2. Barbagallo, D., Cappiello, C., Francalanci, C., Matera, M.: Reputation-based selection of information sources. In: Proceedings of ICEIS 2010 (2010)
3. Burnett, M.M., Cook, C.R., Rothermel, G.: End-user software engineering. Commun. ACM 47(9), 53–58 (2004)
4. Cappiello, C., Daniel, F., Matera, M.: A quality model for mashup components. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 236–250. Springer, Heidelberg (2009)

5. Cappiello, C., Daniel, F., Matera, M., Pautasso, C.: Information quality in mashups. IEEE Internet Computing 14(4), 14–22 (2010)
6. Daniel, F., Casati, F., Benatallah, B., Shan, M.-C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 428–443. Springer, Heidelberg (2009)
7. Derechin, L., Perry, R.: Presto Enterprise Mashup Platform. Technical report, JackBe (2010)
8. Fischer, G.: End-User Development and Meta-design: Foundations for Cultures of Participation. In: Pipek, V., Rosson, M.B., de Ruyter, B., Wulf, V. (eds.) IS-EUD 2009. LNCS, vol. 5435, pp. 3–14. Springer, Heidelberg (2009)
9. Hornbk, K.: Current practice in measuring usability: Challenges to usability studies and research. Int. Journal of Human-Computer Studies 64(2), 79–102 (2006)
10. Jhingran, A.: Enterprise information mashups: Integrating information, simply. In: VLDB, pp. 3–4 (2006)
11. Krummenacher, R., Norton, B., Simperl, E.P.B., Pedrinaci, C.: Soa4all: Enabling web-scale service economies. In: Proceedings of ICSC 2009, pp. 535–542. IEEE Computer Society, Los Alamitos (2009)
12. Liu, X., Huang, G., Mei, H.: Towards end user service composition. In: COMPSAC, vol. (1), pp. 676–678. IEEE Computer Society, Los Alamitos (2007)
13. Maximilien, E.M., Wilkinson, H., Desai, N., Tai, S.: A domain-specific language for web aPIs and services mashups. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 13–26. Springer, Heidelberg (2007)
14. Namoun, A., Nestler, T., De Angeli, A.: Conceptual and usability issues in the composable web of software services. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 396–407. Springer, Heidelberg (2010)
15. Ogrinz, M.: Mashup Patterns: Designs and Examples for the Modern Enteprise. AddisonWesley, Reading (2009)
16. Picozzi, M., Rodolfi, M., Cappiello, C., Matera, M.: Quality-based recommendations for mashup composition. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 360–371. Springer, Heidelberg (2010)
17. Ro, A., Xia, L.S.-Y., Paik, H.-Y., Chon, C.H.: Bill organiser portal: A case study on end-user composition. In: Hartmann, S., Zhou, X., Kirchberg, M. (eds.) WISE 2008. LNCS, vol. 5176, pp. 152–161. Springer, Heidelberg (2008)
18. Simmen, D.E., Altinel, M., Markl, V., Padmanabhan, S., Singh, A.: Damia: data mashups for intranet applications. In: Wang, J.T.-L. (ed.) Proceedings of SIGMOD 2008, pp. 1171–1182. ACM, New York (2008)
19. Spillner, J., Feldmann, M., Braun, I., Springer, T., Schill, A.: Ad-hoc usage of web services with dynvoker. In: Mähönen, P., Pohl, K., Priol, T. (eds.) ServiceWave 2008. LNCS, vol. 5377, pp. 208–219. Springer, Heidelberg (2008)
20. Wong, J., Hong, J.I.: Making mashups with marmite: towards end-user programming for the web. In: Proceedings of CHI 2007, pp. 1435–1444 (2007)
21. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A framework for rapid integration of presentation components. In: Proceedings of WWW 2007, pp. 923–932 (2007)