Enabling End User Development through Mashups: Requirements, Abstractions and Innovation Toolkits

Cinzia Cappiello[1], Florian Daniel[2], Maristella Matera[1], Matteo Picozzi[1] and Michael Weiss[3]

[1] Politecnico di Milano, Dipartimento di Elettronica e Informazione
P. zza L. da Vinci, 32 - 20134 Milano, Italy
{Cinzia.Cappiello,Maristella.Matera,Matteo.Picozzi}@polimi.it

[2] University of Trento
Via Sommarive 14, 38123 Trento, Italy
daniel@disi.unitn.it

[3] Carleton University
1125 Colonel By Dr, Ottawa, ON K1S 5B6, Canada
weiss@sce.carleton.ca

Abstract. The development of modern Web 2.0 applications is increasingly characterized by the involvement of end users with typically limited programming skills. In particular, an emerging practice is the development of *web mashups*, i.e., applications based on the composition of contents and functions that are accessible via the Web. In this article, we try to explain the ingredients that are needed for end users to become mashup developers, namely adequate *mashup tools* and *lightweight development processes*, leveraging on the users' capability to *innovate*. We also describe our own solution, the *DashMash* platform, an example of end-user-oriented mashup platform that tries to fill the gaps that typically prevent end users from fully exploiting the mashup potential as innovation instruments. DashMash offers an intelligible, easy-to-use composition paradigm that enables even inexperienced users to compose own mashups. As confirmed by a user-centric experiment, its paradigm is effective and increases the satisfaction of the end users.

Keywords: Web Mashups, End User Development, User-driven Innovation

1 Introduction

The current trend in the development of modern web applications – and in particular of those applications commonly referred to as Web 2.0 applications – clearly points toward a high user involvement. One of the emerging "user-intensive" practices today is the development of online applications starting from contents and functionality that are available on the Web in form of open APIs or reusable services. A "classical" example is www.housingmaps.com, which interweaves housing offers taken from the Craigslist with Google Maps. The phenomenon is commonly known as ***web mashups***,

and it shows that web users are increasingly also taking part in the *development process* of web applications, in addition to taking part in the content *creation process* like in social web applications (e.g., Wikipedia).

The use of ***open services*** is a unique feature that distinguishes mashup development from other (component-oriented or service-based) development paradigms. Currently, the most popular mashups integrate public programmable APIs (like Google Maps and the Twitter API), but also RSS/Atom feeds (e.g., stock news), content wrapped from third party web sites (e.g., product prices), or any kind of available Web services providing computing support or acting as plain data sources [26]. However, the vision is that of so-called *enterprise mashups* [15], a porting of current mashup approaches to company intranets, enabling enterprise members to play with a company's internal services that give access to the enterprise information assets, and to mash them up in innovative, hopefully value-generating ways, for example, to automate a recurrent bureaucratic procedure.

Provided that suitable tools and methodologies for mashup composition are available, through these open services (both public and company-internal services) even less skilled end users could evolve from passive receivers of innovation to actors actively involved in the ***creation of innovation***. Aggregated over all users, this speeds up innovation (as users conduct parallel experiments with the same service), and covers a wider range of the design space than the service providers could have achieved on their own, had they not exposed their services to other parties. The effort that almost all of the big players of today's Internet economy (e.g., IBM, Intel, Yahoo!, SAP, etc.) are investing into research on mashups is indeed a clear indicator that there is something going on, which goes beyond the current "hacking" of mashups on the Web.

Despite this great potential, there is however a lack of adequate tools and methodologies that really empower the end user to compose services and innovate. In this article, we explore the mashup world, its potential as a tool to be offered to end users to create innovation and its current limits, and propose a new solution through which end users can easily create mashups. In Section 2, we shortly introduce the mashup world and explain why end users are interested in doing their own applications and who else benefits from this practice. Guided by our experience in the development of mashup tools, and by some experimental results, we then discuss the mashup development process and derive a set of requirements that mashups should meet, in order for end users to be able to use them profitably (Section 3). Next, we describe our tool, *DashMash*, that has been conceived to enable users to easily compose mashups supporting analytical processes and, hence, to innovate (Section 4), and in Section 5 we report on a user evaluation of DashMash. In Section 6 we discuss related work, and in Section 7 we finally draw our conclusions.

2 Rationale and Background

Web mashups 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, such

as Google Maps or the Twitter API, or contents taken from Web pages[1]. However, the vision is towards the development of more critical applications, for example the so-called enterprise mashups [15], through which enterprise users can compose by themselves and in a flexible way their dashboards for process and data analysis, using the plethora of available corporate services (e.g., for the access to a variety of enterprise information sources), Web resources and open services. Mashups are therefore gaining momentum as a technology for the creation of innovative solutions in any context where flexibility and task variability become a dominant requirement. A "culture of participation" [10], 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 [25].

2.1 User-based innovation and innovation toolkits

There is a specific driver at the heart of the mashup phenomenon and user participation: *user innovation*, i.e., the desire and capability of users to develop their own things, to realize their own ideas, and to express their own creativity. In a traditional design-build-evaluate cycle, feedback from the user is only collected once a product prototype has been developed. Thus feedback is collected late, and changes to the product that reflect an improved understanding of customer requirements are costly. In a user-driven innovation approach, a service provider offers users an ***innovation toolkit*** through which users can build their own products [23]. This toolkit provides a constrained interface on the capabilities of the company's product platform, but this ensures that the new products are properly constructed, adhering to a sort of *conservative invention* [12].

In general, the idea behind an innovation toolkit is that the iterative experimentation needed to develop a new product can now be entirely carried out by the user. Many users can work in parallel on the solution to a problem, by focusing on their own version of the problem. They can create a solution that closely meets their needs and can more quickly obtain feedback from their development experiments. At the same time, the toolkit provider does not carry the cost of failed experiments. Nonetheless, if an experiment turns out to add significant value, the company can integrate the user innovation back into its core product. On the Web, this is what happened when developers mashed up Flickr with maps. Subsequently, Flickr has incorporated a map function into both its platform and public service. Google also monitors the use of its public APIs (such as Google Maps and Google Search) to fine-tune the APIs and to learn from the best innovative uses [14].

2.2 End users involvement in the mashup development scenario

The way in which mashups are developed depends on the type of mashup. While current *consumer mashups* (for example, all the numerous mashups based on Google Maps) are mainly the results of some hacking activities by expert developers, *enterprise mashups* highlight different potential scenarios that might involve users at

[1] The Web site www.porgrammableweb.com manages a repository of consumer mashups.

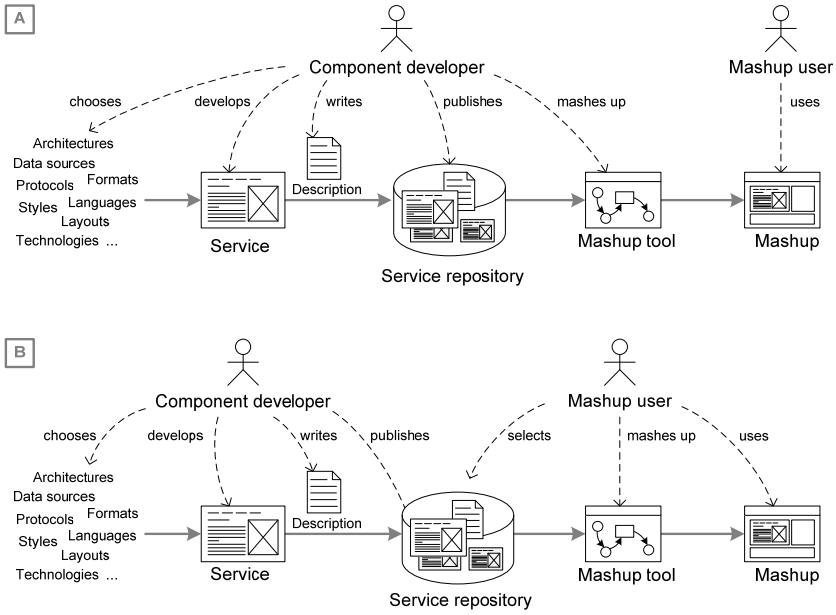

Fig. 1. The mashup development scenarios

different skill levels [18]. In the enterprise context it is indeed possible to recognize two main situations:

A) Mashup tools can be used by ***expert developers*** (for example implementers of an IT department or service providers) to deliver applications quickly. End users are not directly involved in the construction of such mashups but benefit from the shorter turn-around time for new applications. The resources for developing mashups are limited to the expert developers in the IT department. Given the limited resources of an IT department, only frequently requested applications will be developed.
B) Expert developers create services in a format that can be more easily consumed and combined into mashups by ***users who are not themselves developers***, for example requiring simple parameterizations of components; they also provide a tool where anyone creates their own mashups. This is analogous to how spreadsheets are used in organizations today: end users (e.g., business analysts) can create spreadsheets without involvement from an IT department. These mashups are often created for a single purpose and user (they are indeed also known as situational applications [1]), thus they potentially address a larger diversity of user needs.

Fig. 1 illustrates the previous scenarios. The two (extreme) corresponding solutions differ in terms of the heterogeneity of the services that can be combined, the diversity of user needs that can be met, and the level of sophistication of either the

user or the tools that support their work. A tool for the creation of mashups (scenario B) will, initially, be the most challenging scenario to implement. However, it also provides the biggest pay-off. Using the tool, users can combine services and data to create their own mashups. The tool constrains what users can do and, hence, ensures the composability of mashup components. In the sense of the earlier discussion on user innovation [23, 25], such a tool provides a toolkit that enables users to create their own applications. However, users are not limited in terms of the types of applications they can build: this scenario, therefore, supports the greatest diversity of user needs.

Another distinction between the two scenarios is the degree of control over the ***quality*** of mashups being created. In scenario A, the IT department fully controls what kind of mashup is being developed. Thus, the IT department ensures the quality of those mashups. However, not all mashups have stringent requirements in terms of security, performance, or reliability; they may only be used for a specific purpose, and a complex solution developed by the IT department would also be too costly. In scenario B, the IT department selects which components can be mashed up and provides an environment for safely executing those mashups. Users can create mashups from those components to meet needs unanticipated or not served by the IT department. Such mashups may subsequently serve as prototypes for hardened applications developed by the IT department, should there be a need for the mashup to be exposed to many users within the enterprise, or if the mashup has to be offered to outside users.

3 The need for lightweight development processes

Based on the previous observations, it derives that the ideal mashup development process should reflect the innovation potential of mashups: to *compose* an application, starting from given *contents and functionality* responding to personal needs, and to simply *run* it, without worrying about what happens behind the scenes. The *prototype-centric* and *iterative* approach that in the last years has characterized the development of modern Web applications is even more accentuated: the composer, i.e., the mashup end user, just mashes up some services and runs the result to check whether it works and responds to his needs. In case of unsatisfactory results, he fixes the problems and is immediately able to run the mashup again.

The following requirements, which also characterize the EUD domain [7,10], emerge as fundamental ingredients enabling the end user composition of mashups:

- ***Domain-specific focus and terminology***: In order to allow users to understand the possibilities offered by the mashup platform and to make sense of the services and components that are available for composition, it is important to restrict the platform to a well-defined domain the user is comfortable with. That is, we need to be able to develop a tool that *speaks the language of the user*, both in terms of functionalities and terminology known to the user. For instance, only unlikely an average user will understand what a “SOAP web service” is; yet, the user will immediately grasp the meaning of a “currency conversion service”.

- ***Abstraction from technical details**:* In order to help users understand the features provided by the available services and the effect that each service may have on the overall composition, we need to come up with representations of services as visual objects that abstract from technical details, e.g., their programmatic interface or communication protocol. Users should be asked to manipulate, e.g., add, remove, or modify, *visual objects* by operating service visualization properties rather than being required to configure technical details of services and the composition logic. As also confirmed by our user-centric experiment (see Section 5) this increases user satisfaction and, in particular, the user-perceived control over the composition process.
- ***Continuous feedback**:* In order to further enhance the users' perception of the effects that individual actions or services have on the final applications and to allow users to understand the current state and look&feel of the composition, it is highly desirable to provide *immediate visual feedback* on any composition action and to support the *immediate execution of the resulting mashup*. This requirement is backed by our observations that show that end users typically have difficulties in understanding the difference between design time and runtime.
- ***Composition support**:* In order to achieve a tool that speaks the language of the user, it is also important to aid those users that *don't speak the language of the tool*, that is, those users that do not have sufficient development knowledge. Composition can be *assisted or guided* in multiple ways, for instance, by providing recommendations of compatible services that can also increase the quality of the final mashup [20], of composition patterns that have been used successfully in the past [21], or also by pre-compiling or automatically connecting services on behalf of the user (see the next section).

While there are many mashup tools or platforms available today, none of these addresses all the above requirements, which we however regard as fundamental ingredients if we really want to enable users with average skills to develop own applications.

4 The DashMash Platform for Sentiment Analysis

The development of a mashup environment responding to the needs highlighted in the previous section is the object of our own research on the agile, lightweight development of mashups. The environment is called *DashMash*, it is an evolution of our prior work on mashup composition [28], and aims at an integration approach where a variety of different component types and technologies, ranging from simple RSS feeds to complex SOAP or RESTful Web services and UI components[2] can be combined, thanks to the adoption of some descriptive models for both component services and mashup composition.

[2] UI components are characterized by a presentation level (the User Interface) that is then reused "as is" within the final integrated mashup. Google Maps is an example of UI component: beside its application logics related to geo-localization, it also offers a UI for the map-based visualization of geo-localized data.

DashMash is a mashup tool, specifically conceived for the construction of dashboards exploiting both company-internal services extracting data from local data warehouses, and public APIs and web resources. Recently DashMash has been specialized for sentiment analysis (the ***domain***), an emerging business intelligence practice that aims at understanding market trends from the unsolicited feedback provided by user comments published on the Web through social applications. An ongoing project funded by the Municipality of Milan focuses on the design of an engine that is able to automatically extract sentiment indicators summarizing the opinions contained in user generated contents [2]. In this context, DashMash has been adopted to allow end users, i.e., analysts and decision makers interested in improving the quality of services offered by Milan city, to "compose" their analysis flexibly, playing in variable ways with sentiment indicators, and also complementing such indicators with interesting external Web resources, for example linking sentiment indicators to news, events, and opinions that cause trends and behaviors. The DashMash customization to the sentiment analysis domain has required the development of some ad-hoc services for the sentiment indicators computation and visualization, which are offered to the users as basic, still configurable, elements for their compositions.

As shown in Fig. 2, mashup creation is enabled through a web-based, visual environment; the visual composition paradigm has been specifically conceived to hide the complexity of the technical details and the composition languages actually managing the execution of the mashup (the ***abstraction***). As shown in Fig. 2(a), a visual menu at the left hand side presents the list of services: *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, ad-hoc developed services[3], and utility open APIs/services, such as RSS feeds and calendars. Each component is denoted through an icon and a label that shortly recall the offered functionality. Components can be mashed up by moving their corresponding icons into the so-called *workspaces*. As soon as a component is moved into a workspace, its UI is immediately rendered so that the users can easily check whether the component choice satisfies their needs.

Each workspace is associated with a data set, which results from the integration of data sources and filters that the users can select and configure depending on their needs. Some default rules also assume that in absence of user selections some data sources are automatically associated with the Workspace. In this way, the creation of meaningful mashups is preserved. Each workspace visualizes its data set according to the visualizations offered by selected *viewers*. For example, Fig. 2(a) shows a mashup in which the user has selected two data sources, storing contents extracted from two social applications, Twitter and TripAdvisor, and has filtered them by using a keyword-based filter, with key = "Milan". Contents are then presented through a pie chart viewer, visualizing the percentage of comments related to categories of interest in the tourism domain (e.g., food, entertainment, art, and other relevant entities), and a scatter plot visualizing the average value of sentiment for the same set of categories.

---

[3] Several viewers offering graphic visualizations have been developed using the Highcharts JS library (http://www.highcharts.com/), to offer advanced presentations specific for sentiment indicators.

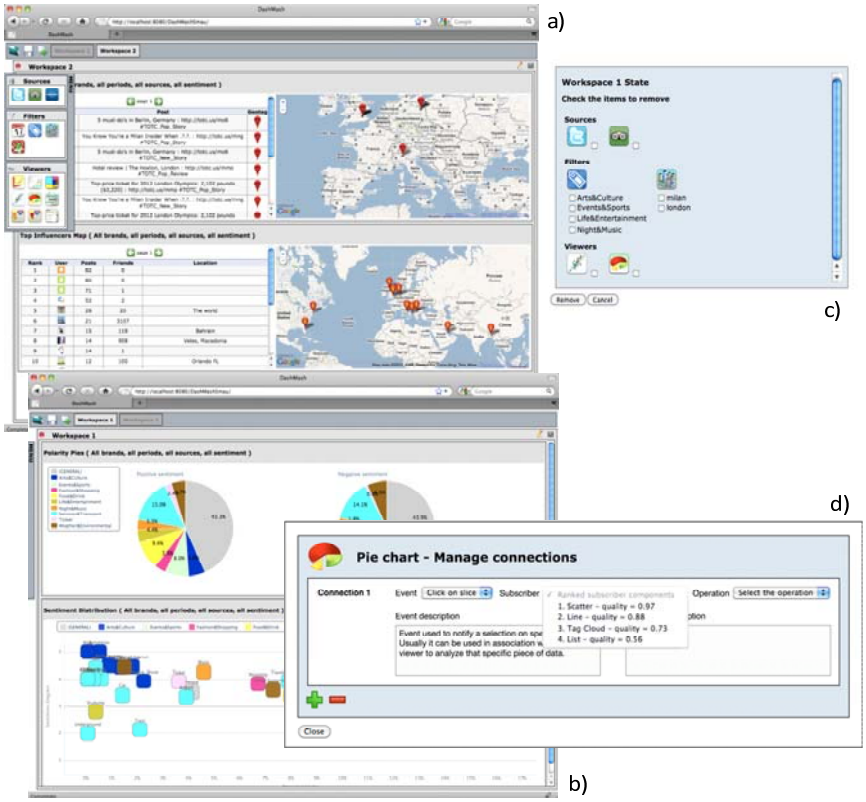

Fig. 2. The DashMash editor for drag-and-drop composition of mashup services and immediate execution of the resulting mashup. The two screenshots show the mashup of sentiment analysis dashboards [2, 6].

Fig. 2(b) shows a second mashup defined on top of the same data sources as the previous one. In this case, the filters select comments from users that are considered opinion leaders, the so-called *influencers*, who are visualized through a *list viewer* integrated with Google Maps to show the influencers' location. This is an example of integration between an internal service (the one providing information about influencers) and an external, public API, this latter providing an added value to the overall analysis.

Users can iteratively modify the composition, by adding or dropping components through some visual actions. Changes are enacted in real time, i.e., the mashup visualization changes accordingly, so that users can immediately see the effect of their composition actions in their workspace (the ***continuous feedback***). They can also access a detailed description of the status of the current composition (see Fig. 2(c)), summarizing the main elements, their configuration and synchronization behavior, and easily modify sources, filters, viewers or even configuration properties of single filters or viewers.

Once a new component is added, the system automatically binds it to the pre-included components - if possible (the ***composition support***). The platform can automatically generate service bindings, based on a service classification and on corresponding parameter-operation couplings. For example, when a new viewer is added into a workspace, its visualization logic is automatically mashed up with the corresponding data sources and filters associated with the workspace. Users can then introduce further synchronization behaviors. Simple dialog boxes, abstracting from technical details, allow them to create new service combinations resulting in synchronized behaviors. For example, starting from the mashup shown in Fig. 2(a), the dialog box presented in Fig. 2(d) allows the user to set a coupling so that a click on a pie slice contextualizes the analysis offered by the map viewer to that selected label. Based on descriptive models of components, the dialog box presents possible connection points, namely the component events (see next section), exposed by the components selected by the user, plus a short description of the resulting synchronization behavior. The system provides suggestions about other candidate components based on compatibility rules and quality criteria [20].

The rest of this section is devoted to illustrate the architectural elements and the mechanisms that implement the previous functions and behavior in DashMash.

4.1 DashMash Architecture

The overall organization of the DashMash platform is illustrated in Fig. 3. The mashup execution is centered on 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, and 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.

Mashup Execution. DashMash capitalizes on the mashup paradigm defined in [28], which is based on an event-driven model operating at the presentation level: *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 subscribed to such events (e.g., the visualization of details of the selected data in a scatter plot). The occurrence of events, intercepted by an *Event Broker* module, causes a state change in the subscribed components. Each component therefore keeps running according to its own application logic, within the scope defined by an HTML <*div*>. As soon as events occur, the involved components publish them. Based on the definition of service binding, the so-called *listeners,* an *Execution Handler* then notifies the subscribed components, if any, and triggers the execution of their corresponding operations.

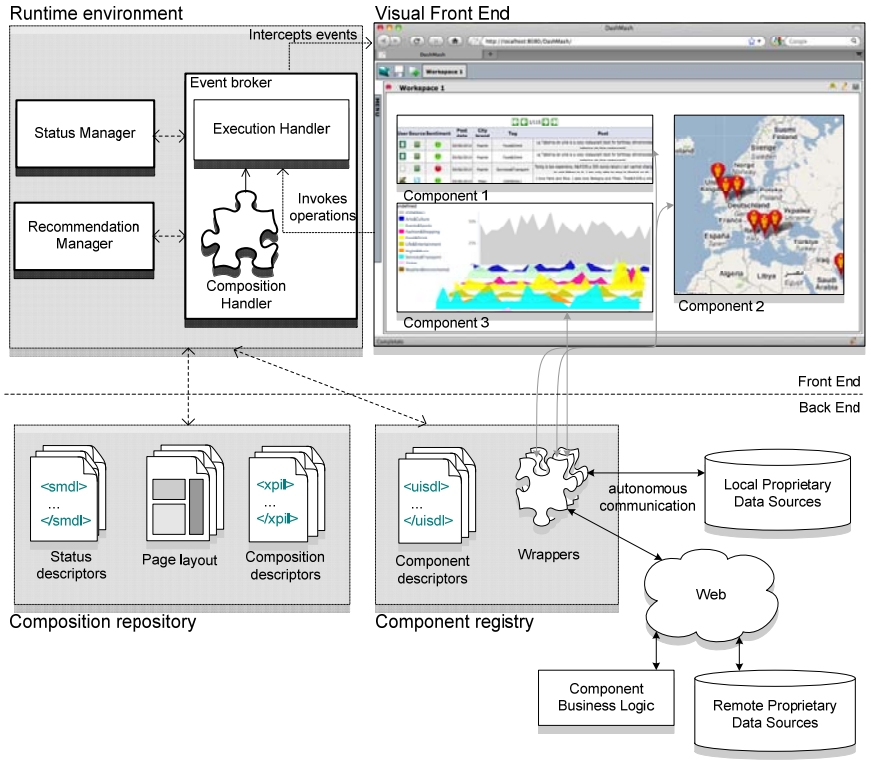

Fig. 3. Overall organization of the DashMash platform

Listeners are specified in a *composition model*, expressed by means of the XPIL (eXtensible Presentation Integration Language) XML-based language [28]. This composition logic also requires each component to be characterized by a model expressing the binding with the actual service/API, the events that the component can generate, and the operations that enable other components to modify its internal state. This component description, expressed by means of the UISDL (UI Service Description Language) XML-based language [28], provides a uniform model to coordinate the mashup composition and execution, which obviates the heterogeneity of service standards and formats by embedding only the information needed for synchronizing services at the presentation level. The adoption of such a component model is an important ingredient toward the provision of an environment where technical details are hidden to the user.

Component and composition models are stored in dedicated repositories:

– The *Composition Repository* maintains the XPIL-based specifications of the compositions as created by the users, the HTML templates for the mashup

layout management, and a *state model* that maintains information about the configuration of a mashup instance (i.e., values instantiating parameters and specific configuration of the involved components), to support saving and restoring functions, history management, and also the easy, "on-the-fly" modification of the composition (as shown in Fig. 2(c)).

- The *Component Repository* stores the component descriptive models plus wrappers through which the platform invokes service operations. The creation of component wrappers is the only "technical" activity that is required to register services into the DashMash platform, and, as such, it is up to the expert developer, not the end user. However, once the component registration is performed, the end user can transparently use and integrate any service through the visual paradigm illustrated above.

Mashup composition and automatic model generation. Due to the intermixing between mashup composition and execution, in DashMash events captured by the event broker can be related not only to users and system actions occurring during the mashup execution (those ones managed by the execution handler, which causes a change to some other component's state), but also to the dynamic definition of the composition (e.g., the drag&drop of a component icon into the composition area). The Event Broker intercepts events and dispatches them to the modules in charge of their handling.

The *Composition Handler* manages composition events. In particular, it automatically translates the addition of a component into a set of listeners, based on default couplings between the involved services. Based on such listeners, it creates or updates (if already existing) the current composition model. It also dispatches the composition events to the *Status Manager* in charge of maintaining the description of the mashup instance status. As soon as the composition and the status update are complete, the mashup composition is reloaded and immediately rendered through the visual front-end. The mashup is then executed according to the event-driven, publish-subscribe logic that characterizes the Execution Handler.

Service binding definition. DashMash supports the definition of *default* and *custom* bindings:

- *Default bindings* are automatically defined by the Composition Handler when a composition action is intercepted and ensure a minimum level of inter-component synchronization that does not require end users to explicitly define service coupling. To enable the automatic definition of default bindings, we start from a classification of components. For example, in order to facilitate the construction of a dashboard, it is possible to identify four component classes, namely *data services*, retrieving data from corporate/relational data sources, *filters*, expressing selection conditions over the context defined by a workspace, *viewers*, supporting the visualization of result sets also offering data aggregation and transformation functions, and *generic components*, i.e., any kind of open service (local or remote) offering functionality that can make the analysis process more effective. Service classification is domain-specific, and needs to be revised for any DashMash customization. Classification changes, however,

only imply a new configuration of the Composition Handler, while no other changes are required to other architectural elements.

- *Custom bindings* are user-defined. Nevertheless, the Composition Handler supports the user in the choice of components and component bindings, since it generates compatibility– and quality– based recommendations. To this aim, it dispatches the composition events to the *Recommendation Manager*, an additional module of the runtime environment that is in charge of evaluating the quality of the current composition and providing suggestions about the selection of possible components to add to or of compatible components that can substitute the existing ones in order to achieve or improve the mashup quality [5, 20].

5 Validation

In order to validate the composition paradigm of DashMash with respect to user needs, we conducted a study involving 35 participants. Six of them were real end users of the DashMash sentiment analysis customization, i.e., analysts and decision makers that are supposed to actually use DashMash for their analyses, with a medium technical expertise. Other users were master students of the Computer Engineering program at Politecnico di Milano, featuring different levels of technical background: 12 of them were already acquainted with concepts related to service composition and mashups. The others were familiar with Web application development but not with service composition and mashups.

We observed users completing two tasks through DashMash, which consisted in the composition of mashups extracting and visualizing data related to two specific sentiment indicators: the percentage of volume for the positive and negative sentiment along different brand categories, and the volume distribution in time for the positive sentiment. In both the mashups, multiple components needed to be synchronized among each other. Our goal was to assess how easily the users would be able to develop a composite application. The experiment specifically focused on the effectiveness and intuitiveness of the composition paradigm, trying to measure such factors in terms of user performance, ease of use and satisfaction.

We expected all users to be able to complete some experimental tasks, with however 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. However, surprisingly no significant differences in task completion time were found between experts and novices. In particular, domain expertise was not discriminating for task 1 ($p = .085$) and for task 2 ($p = .165$). Similarly, technology expertise was not discriminating 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 badly. However, it indicates that the tool enables even inexperienced users to complete a task in a limited time and that the expertise needed to properly understand the necessary concepts and to operate the tool is relatively low.

Another interesting result is that the difference in completion times for the two tasks 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 [13]: 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 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. 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 the perceived usefulness ($p = .51$).

The post-experiment questionnaire also allowed us to assess the user satisfaction by means of a semantic differential scale requiring users to judge the method on 12 items. We did not find significant differences between experts and novices. Despite our initial assumption, we therefore 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. Moreover, 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.

6 Related Works

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. 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://www.jackbe.com/) also adopts a pipes-like approach for data mashups, and allows a portal-like aggregation of UI widgets (mashlets). IBM DAMIA [23] offers support to quickly assemble data feeds from the Internet and a variety of enterprise data sources. MashArt [8] focuses on the integration of heterogeneous components (not only data or RSS feeds), offering a mashup design paradigm through which composers create graph-based models representing the mashup composition.

With respect to manual programming, all the previous platforms certainly alleviate the mashup composition tasks. However, to some extent they still require the user to deeply understand the application logic behind services and the integration logic. In some cases, building a complete Web application also equipped with a user interface requires the adoption of additional tools or technologies. A recent user-centric study [9] 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.

Marmite [27] is a tool specifically tailored for integrating and accessing information sources. With respect to other platforms, it offers a more intuitive composition paradigm, which has been devised by means of a user-centered process: it allows users to easily program a set of source filtering operators that can then be connected into a data flow. In line with our approach, Marmite goes in the direction of easing mashup development, for example ensuring continuous feedback through an immediate visualization of the included services and the overall resulting mashup. This works however is still centered on a dataflow paradigm, which in our opinion does not abstract enough from the technical background, requiring for example the users to define operator chaining by means of parameter coupling.

Our work tries to overcome the previous limitations, allowing end users to develop their own mashups through an intelligible paradigm that abstracts from technical variables. The aim is to maximize some well-known principles that characterize End User Development [3,4,7,10]. In particular, our approach provides a composition environment that can facilitate the creation of successful applications accommodating the diversity of the needs, interests and activities that end users want to perform through computer systems. DashMash is indeed a general-purpose mashup environment in which however the risk of becoming too general, thus in some cases ineffective, is limited by the possibility to be customized through the development of ad-hoc components and the registration into the platform of out-of-shelf resources that are of interest to the domain-specific activities that the users need to tackle. In other words, our platform tries to provide the right trade-off between extremely general systems and highly specialized, domain-specific applications that on the other hand cannot be generalized, adapted or evolved [7, 11].

7 Conclusions

In this article, we have proposed our perspective on mashups, mashup tools, and lightweight mashup development processes, arguing that enabling web users (in the consumer context) or employees (in the business context) to develop own applications demands for a high degree of assistance and intelligible concepts. Our proposed approach is a first attempt towards the realization of this objective. However, some more efforts are needed on the following ingredients:

- *Easy-to-use APIs*: Expressive models and description languages for data, application logic, and user interface components are needed to facilitate the component integration within mashups. Suitable discovery and selection facilities (e.g., registries and protocols) are needed as well.
- Design aimed at *interoperability*: Services and mashups should be interoperable, meaning that they must feature cross-platform reusability. Although some proposals exist for mashup-specific standards [19], any mashup platform keeps using its own models and description languages.
- *Dependable mashups*: Although the current efforts are mainly devoted to the improvement of the previous aspects, it is unquestionable that mashups also need to address issues like reliability, transactions, and security – especially if used in business contexts.

DashMash addresses the currently still low ease-of-use of APIs (by definition, APIs are still oriented toward programmers, not end users) and their generally low interoperability (e.g., in terms of supported communication protocols or data formats) by wrapping them and transforming their data into an internal, canonical format that can be understood by other wrappers. This task, however, requires the intervention of expert developers, and cannot be accomplished by the users themselves. As for the dependability of mashups, DashMash does not provide any specific solution, as so far we support non-critical application scenarios only. We have however planned some extensions to address these features.

Finally, while lightweight development processes are needed to alleviate the effort of mashup developers and especially end-users, the development of services to be integrated into mashups is a demanding activity, to be performed according to traditional development processes by professional programmers. After all, if on the one hand the success of a mashup is influenced by the added value that the final combination of services is able to provide, on the other hand it is self-evident that the quality of the final combination is strongly influenced by the quality of each single service. Defining models and techniques for developing "good" services and for assessing their quality is therefore another promising direction of our current research, which can give a fundamental contribution towards the development of quality mashups and to aid user innovation [5].

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 in the enterprise context), as well as an extension of the recommendations mechanisms based on the emergence of composition patterns from the community's mashups [20]. We also aim at investigating mashup interoperability, for example making DashMash mashups compatible with emergent standards, such as Enterprise Mashup Markup Language (EMML) [19].

References

1. S. Balasubramaniam, G. A. Lewis, S. Simanta, D. B. Smith. Situated Software: Concepts, Motivation, Technology, and the Future. *IEEE Software*, Nov-Dec, 2008, pp. 50-55.
2. D. Barbagallo, C. Cappiello, C. Francalanci, M. Matera. A reputation-based DSS: the INTEREST approach. *Proceedings of ENTER'10*.
3. P. Bottoni, M. F. Costabile, S. Levialdi, M. Matera, P. Mussio: Principled Design of Visual Languages for Interaction. *Proceedings of VL 2000*, IEEE Computer Society, pp. 145-155.
4. M. Burnett, C. Cook, and G. Rothermel. End-User Software Engineering. *Communications of the ACM*, 47 (9), 2004, pp53-58.
5. C. Cappiello, F. Daniel, M. Matera. A Quality Model for Mashup Components. *Proceedings of ICWE'09*, Springer LNCS, pp. 235-249.
6. C. Cappiello, M. Matera. M. Picozzi, G. Sprega, D. Barbagallo, C. Francalanci. DashMash: a Mashup Environment for End User Development. Submitted for publication, October 2010.
7. M. F. Costabile, P. Mussio, L. P. Provenza, A. Piccinno. Supporting End Users to Be Co-designers of Their Tools. *Proceedings of IS-EUD'09*, pp. 70-85.

8. F. Daniel, F. Casati, B. Benatallah, and M.-C. Shan. Hosted Universal Composition: Models, Languages and Infrastructure in MashArt. *Proceedings of ER 2009*, LNCS 5829, pp 428–443.
9. A. De Angeli, A. Namoun, T. Nestler. End user requirements for the composable web. *Proceedings of ComposableWeb'10*, LNCS 6385, Springer Verlag, pp. 396-407, 2010.
10. G. Fischer. End-user Development and Meta-Design: Foundations for Cultures of Participation. *Proceedings of IS-EUD 2009*, pp. 3–14, 2009.
11. G. Fischer. Beyond Binary Choices: Understanding and Exploiting Trade-Offs to Enhance Creativity. *First Monday*, 11 (2006)
12. T. P. Hughes. The evolution of large technological systems. *The social construction of technology systems: New directions in the sociology and history of technology*. W. E. Bijker, T. P. Hughes and T. J. Pinch, eds. Cambridge, Mass., MIT Press, pp. 51-82, 1987.
13. K. Hornbæk. Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies*, 64(2):79–102, 2006.
14. B. Iyer and T.H. Davenport. Reverse Engineering Google's Innovation Machine. *Harvard Busines Review*, 86(4), pp. 58-69.
15. A. Jhingran. Enterprise information mashups: integrating information, simply. *Proceedings of VLDB'06*, pp. 3-4.
16. M. Maula, T. Keil, J.-P. Salmenkaita. Open Innovation in System Innovation Contexts. *Open Innovation: Researching a New Paradigm*, Chapter 12, 2006, pp. 249-257.
17. Z. Obrenovic, D. Gasevic. Mashing Up Oil and Water: Combining Heterogeneous Service for Diverse Users. *IEEE Internet Computing*, Nov/Dec, 2009, pp. 56-64.
18. M. Ogrinz. *Mashup Patterns: Designs and Examples for the Modern Enterprise*. Addison-Wesley, 2009.
19. OMA. *EMML Documentation*. Technical report, Open Mashup Alliance, http://www.openmashup.org/omadocs/v1.0/index.html, December 2010.
20. M. Picozzi, M. Rodolfi, C. Cappiello, M. Matera. Quality-based Recommendations for Mashup Composition. *Proceedings of ComposableWeb'10*, LNCS 6385, pp. 360-371, 2010.
21. S. Roy Chowdhury, C. Rodríguez, F. Daniel, F. Casati. Wisdom-Aware Computing: On the Interactive Recommendation of Composition Knowledge. *Proceedings of WESOA 2010*, Springer, December 2010.
22. M. Sabbouh, J. Higginson, S. Semy, D. Gagne. Web mashup scripting language. *Proceedings of WWW'07*, pp. 1305-1306.
23. D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh. Damia: Data Mashups for Intranet Applications. *Proceedings of SIGMOD 2008*, pp. 1171–1182. ACM.
24. S. Thomke, E. von Hippel. Customers as Innovators: A New Way to Create Value. *Harvard Business Review*, 80(4), 2002, pp. 74-81.
25. E. von Hippel. *Democratizing Innovation*, MIT Press, 2005.
26. M. Weiss, G.R. Gangadharan. Modeling the Mashup Ecosystem: Structure and Growth. *R&D Management*, 2009 (accepted for publication).
27. J. Wong, J. I. Hong. Making Mashups with Marmite: towards end-user Programming for the Web. *Proceedings of CHI'07*, pp. 1435–1444, 2007.
28. J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, M. Matera. A Framework for Rapid Integration of Presentation Components. *Proceedings of WWW'07*, pp. 923 - 932.