Quality-Aware Mashup Composition: Issues, Techniques and Tools

C. Cappiello*, M. Matera*, M. Picozzi*, F. Daniel†, and A. Fernandez‡
*DEI - Polytechnic of Milan, Italy, 20134
Email: [cappiell, matera, picozzi]@elet.polimi.it
†DISI - University of Trento, Italy, 38123
Email: daniel@disi.unitn.it
‡DSIC - Universitat Politècnica de València, Valencia, Spain, 46022
Email: afernandez@dsic.upv.es

***Abstract*—Web mashups are a new generation of applications based on the composition of ready-to-use, heterogeneous components. 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 their artifacts, thus accommodating the inherent variability of the users' needs. Current advances in mashup technologies are good candidates to satisfy this requirement. However, some issues are still largely unexplored. In particular, quality issues specific for this class of applications, and the way they can guide the users in the identification of adequate components and composition patterns, are neglected. This paper discusses quality dimensions that can capture the intrinsic quality of mashup components, as well as the components' capacity to maximize the quality and the user-perceived value of the overall composition. It also proposes an assisted composition process in which quality becomes the driver for recommending to the users how to complete mashups, based on the integration of quality assessment and recommendation techniques within a tool for mashup development.**

I. Introduction

Mashups are new-generation Web applications that integrate heterogeneous "components", such as RSS/Atom feeds, Web services, content wrapped from third party Web sites, or public APIs (e.g., Google Maps). Such components may have their own user interface, reused to build the interface of the composite application, may provide computing or visualization support, or may just act as plain data sources. Independently of the nature of the components, the goal of mashups is to provide novel features by integrating resources in a value-adding manner, similar to service composition, which however only focuses on the application logic layer.

The interest in mashups has grown constantly over the last years. Mashups and their "lightweight" composition approach represent indeed a new opportunity for companies to leverage on past investments in service-oriented software architectures and Web services, as well as on the huge amount of public APIs available on the Web. In fact, the possibility to integrate Web services with UIs greatly supports the development of complete applications. Furthermore, the emergence of mashup tools, which aim to support mashup development without the need for programming skills, has moved the focus from developers to end users, and from product-oriented software development to consumer-oriented composition [1].

So far, research on Web mashups has focused on the definition of composition technologies and tools, while limited efforts have been devoted to quality concerns [2]. Research on Web Engineering has proposed several quality models addressing Web applications. However, as also proved by a study that we conducted on a large collection of mashups from the programmableWeb.com repository [3], the application of traditional and generic models not always captures the real value of Web mashups. Although the majority of such applications are characterized by a simple one-page structure, specific concerns, related to the component-based development and to the dynamics that characterize the mashup ecosystem, require specific attention.

In this paper we discuss the quality of mashups in the light of the activities that characterize their development process. In this context, the quality and the role of the constituent components become relevant ingredients for quality assessment [4], [5], [3]; the added value introduced by the integration logic is however also relevant. We thus propose evaluation techniques taking into account these features, and show how they can be integrated into the mashup life cycle to enable a *quality-aware* mashup development process. We also show how this process can be supported by a mashup-maker tool [6] that we have developed as a result of our experience on the definition of models and methods for mashup development.

The paper is organized as follows. In Section II we illustrate the rationale of our research on assisted mashup composition, describing the composition paradigm that we have defined for supporting mashup composition by end-users and highlighting the need for quality-based mechanisms guiding users in the selection of components and composition patterns. In Section III we illustrate the typical scenario for mashup development and the different quality issues related to component creation and mashup composition. Section IV introduces a set of techniques to assess mashup quality, also illustrating how they can be used for the generation of quality-aware recommendations assisting mashup composition. Section V illustrates how the assessment techniques have been integrated into the architecture of our own mashup tool. Section VI discusses related works. Finally

Section VII concludes the paper and discusses our future work.

II. Motivation and Background

Mashups are Web applications that are strongly characterized by the reuse of ready-to-use components. They preserve the data, the logic and the UI (if any) of the original components, and introduce an additional integration logic to synchronize the behavior of the different components. In the last years we have assisted to an evolution of the way mashups are created. Initially, mashups were merely developed manually by skilled programmers, who took advantage of reusable components and mainly devoted their efforts to programming the composition logics. Soon, mashup development emerged as a paradigm to enable end-users, not necessarily expert programmers, to compose their applications. This phenomenon therefore started inspiring a new generation of tools for the "assisted composition of mashups", the so-called *mashup-makers*, providing visual editors where intuitive notations allow the users to select components and synchronize them by defining data or control flows.

The aim of mashup-makers is to ease mashup development, empowering also unskilled users to compose their own applications [7]. However, they still require the users to identify the "right" components, i.e., components that i) best fit the current composition from the point view of the syntactic and semantic compatibility and that (ii) increase the mashup quality, also from the point of view of the added value that they can provide in terms of additional data and functions. As found out in recent user-centric studies [8], this task could not be trivial for the average users. The need therefore emerges for design-time assistance mechanisms, guiding the users in the selection of components.

In the last years we have worked on the design of mashup tools fostering end-user development [10], [11], [6], also trying to identify mechanisms to assist the users in the composition activity. Figure 1 illustrates the visual editor of one of such tools, PEUDOM (Platform for End User Development) [9]. A visual menu shows the available components, previously registered into the platform by defining appropriate descriptions and wrappers for the services they relate to[1]. The end-users can add components into their mashups by simply moving the corresponding icons into the central *workspace*. The effect of this action is the immediate visualization of the component's UI. For example, the mashup illustrated in Figure 1, which is supposed to retrieve and visualize data about music events, is built by "moving" into the workspace a number of heterogeneous components that refer to public APIs, namely Facebook and LastFM, both used to retrieve music events (e.g., concerts), Flickr, to retrieve images, Youtube to retrieve videos, Google Maps, for geo-localizing events, and TimelineJS, to visualize the retrieved events on a timeline.

The visual composition paradigm also allows composers to add *synchronization bindings* among the different components, by defining "parameter-operation" couplings, through which parameters propagates from one component to another, thus synchronizing the state of the different components involved in a binding chain. The selection of an icon in the left-hand upper corner of a component's window opens a pop-up where users can choose to activate the available bindings. The user's choices of components and component bindings are then translated into a *composition model*, in which *listeners* specify the way in which operations exposed by some components subscribe to events raised by other components, according to an event-driven publish-subscribe paradigm [10]. The so-achieved composition model is then used at runtime to manage the execution of the mashup, by invoking and synchronizing the services involved in the composition.

In the rest of this paper we will show how we have extended our composition paradigm with quality assessment techniques able to assist the users in the selection of components for the construction of quality mashups. In particular, as highlighted in Figure 1, the tool has been extended with functions that, based on the assessment of quality properties, suggest the use of *alternative components* that can improve the quality of the current composition, and of *additional components* that can extend the composition by improving its added value.

III. Quality issues in mashup development

Mashup applications generally consist of a single page, with a simplistic presentation layer, usually deriving from the combination of the layout of each individual component, and an application logic mainly deriving from the operations exposed by the involved components [3]. The additional integration logic has a limited complexity.

One could assume that, being mashups "simple" Web applications, their quality could be addressed by the methods so far proposed for traditional Web applications. This is partially true: traditional principles must not be neglected; however, models need to be repurposed to capture the salient characteristics of these applications. This is the conclusion we reached by analyzing about 100 mashups available on programmableWeb.com, by applying criteria and metrics related to traditional dimensions of the perceived quality of Web applications, e.g., accessibility and usability [3]. We compared the results achieved through such traditional metrics with the results of a heuristic evaluation conducted by a pool of five independent evaluators, PhD students and researcher acquainted with Web Technologies and Web mashups. The study revealed a discrepancy between the two assessments, highlighting that understanding the quality of mashups requires models that takes into account the specifics of such applications. For example, several applications were ranked as good on the basis of Web quality metrics, but the expert inspection revealed that they just "embedded" some APIs without any attempt to define an integration logic, which is instead a typical aspect of mashups. It is indeed fundamental to ground assessment techniques on the typical activities in the mashup life-cycle, which spans from *component development* (this task is generally accomplished by expert developers, outside of mashup-

[1]The component registration is a technical activity, performed by the platform administrators, that is necessary to provide the end-users with ready-to-use components.

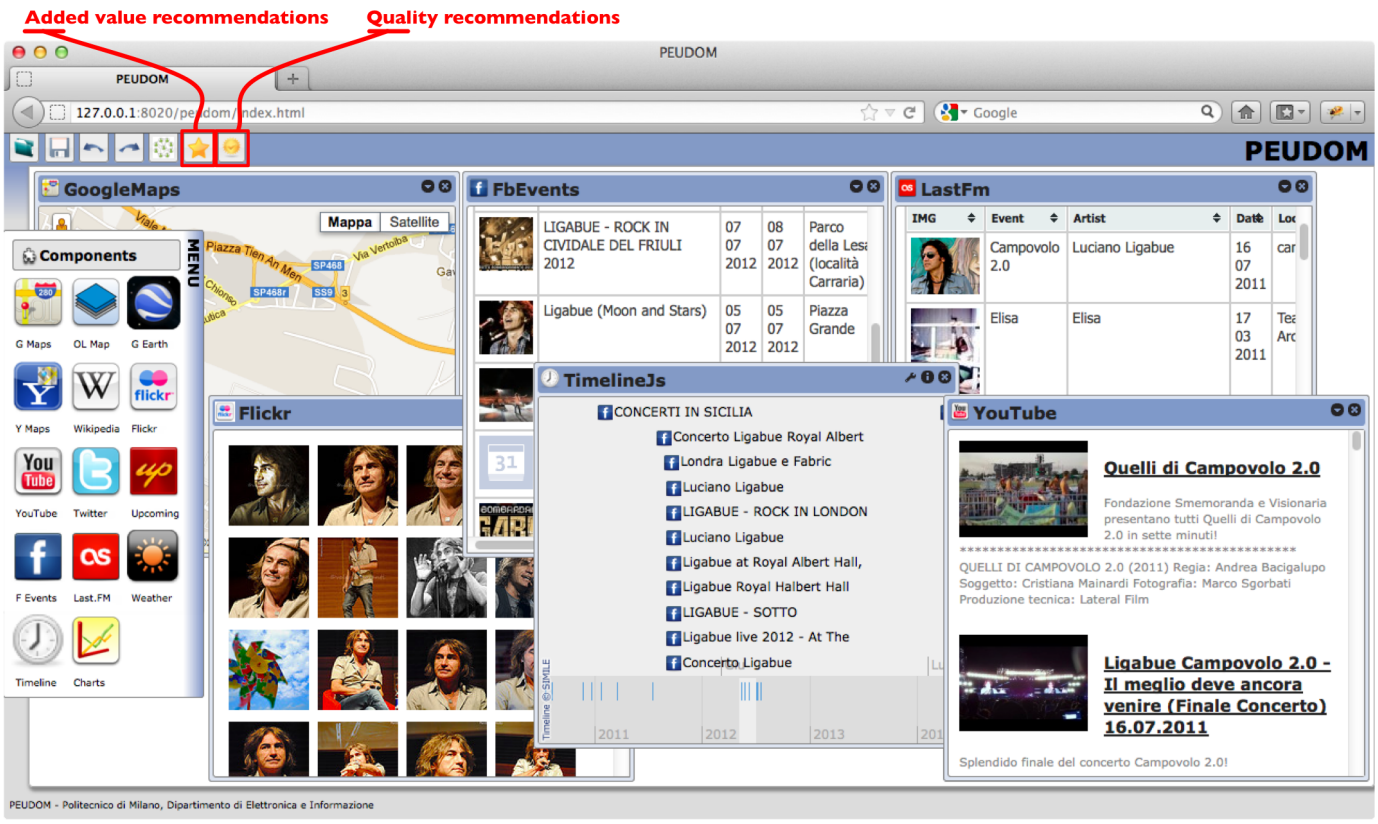


Fig. 1: The visual editor of the PEUDOM tool [9], [6], with an example of mashup composition for searching information on music events.

maker tools) to the identification and integration by the end-users (not necessarily experts, possibly using a tool like the one described in the previous section) of components into a final *mashup composition*.

A. Component Development

When used in a mashup composition, components are selected by considering some external properties. Indeed, publishing mashup components as APIs or services hides their internal details and gives more importance to external properties [12]. In line with this black-box view, in [4] we proposed a quality model for mashup components that privileges properties of the component APIs. This is indeed the perspective that is most relevant to the mashup composer and to the mashup user, as also emerged by our own experience with the development of components and mashups and on the experimental evidence gathered by analyzing data from *programmableweb.com* [3]. We organized the model along three main dimensions:

- *Data quality*, focusing on the suitability of the data provided by the component in terms of *accuracy*, *completeness*, *timeliness*, and *availability*;
- *API quality*, referring to software characteristics that can be evaluated directly on the component API and that relate to the offered *functionality*, *reliability*, and *API usability*;
- *Presentation quality*, addressing the user experience, with attributes such as *presentation usability*, *accessibility*, and *reputation*.

Independently of the adopted quality model, we assume that the component developer tries to maximize quality properties, but especially that within the adopted mashup-maker tool such properties are visible through ad-hoc descriptions on which the choice of components by mashup composers can be based. For example, in our mashup tool components are made available in a repository where *component descriptions* specify both the functional properties of the component (e.g., exposed operations, I/O parameters, their syntactic and semantic categories to assess component compatibility and similarity), and also quality annotations. Figure 2 reports a simplified example of such a descriptor. For a given component - Google Maps in the example - the descriptor specifies the exposed *operations* and *events*, i.e., the main ingredients at the basis of our event-driven publish-subscribe composition paradigm, adequately annotated to express syntactic and semantic categories. The descriptor also specifies quality annotations, expressing quality properties such as the complexity of the component's technological properties (e.g., languages and data formats enhancing operability and interoperability), the richness and completeness of the provided data, the UI usability. These properties (e.g., the available languages and formats) are partly derived from the documentation of services and APIs, disclosed by

```
<component id="googlemaps" name="GoogleMaps"
    description="Geolocalization service.">

    <operation name="centerMap" description="..."
        similarity-meaning="CenterMap"
        similarity-verb="center" similarity-object="map">

        <param name="latitude" description="Latitude"
            direction="input" type="xsd:decimal"
            similarity-meaning="Geography.owl#Latitude"/>
        <param name="longitude" description="Longitude"
            direction="input" type="xsd:decimal"
            similarity-meaning="Geography.owl#Longitude"/>
    </operation>

    ...

    <event name="positionOnClick" description="..."
        similarity-meaning="ProvidePosition"
        similarity-verb="provide" similarity-object="position">

        <param name="latitude" description="Latitude"
            direction="output" type="xsd:decimal"
            similarity-meaning="Geography.owl#Latitude"/>
        <param name="longitude" description="Longitude"
            direction="output" type="xsd:decimal"
            similarity-meaning="Geography.owl#Longitude"/>
    </event>

    ...

    <qualityAttributes>
        <reputation>1</reputation>
        <languages>
            <language>javascript</language>
        </languages>
        <dataFormats>
            <dataFormat>JSON</dataFormat>
            <dataFormat>XML</dataFormat>
            <dataFormat>VML</dataFormat>
            <dataFormat>KML</dataFormat>
        </dataFormats>
        <security>no authentication</security>
        <timeliness>0.9</timeliness>
        <accuracy>0.9</accuracy>
        <completeness>0.8</completeness>
        <availability>1</availability>
        ...
    </qualityAttributes>

</component>
```

Fig. 2: Excerpt of a descriptor adopted in PEUDOM for the specification of functional and quality properties of a component.

the component developers - if any. Some other properties may also derive from evaluations that the administrator of the mashup platform performs at the component registration time. Also, quality and popularity data disclosed by public ranking services (e.g., Alexa (http://www.alexa.com)) can be taken into account.

B. Mashup Composition

When composing a mashup, the composer first needs to identify the "right" components. The selection can be based on the syntactic and semantic fitness of each component within the mashup under construction, but also on quality measures. As soon as components are added into the composition, the assessment of the quality of the overall composition can indeed take place, and recommendations can be provided to the user accordingly.

Figures 3a and 3b shows examples of the recommendations generated by our tool, to guide the user in the selection of alternative or additional components. The starting point for computing the quality indexes on which the recommendations are based is the quality of each single component. However, the composition model is also taken into account, to identify the *role*, i.e., the importance, that components play within the composition [5]. The quality of the composition can thus be evaluated as an aggregation of the quality of the single components, weighed on the basis of the components' role. The analysis of the composition model can also provide indications about the richness of the integration logic, revealing whether the composition introduces information spaces, functionality sets and visualizations that are richer than what would be achieved by accessing separately individual components. Recommendations can thus be generated, by ranking the components available in platform on the basis of their attitude to increase the quality and the value of the mashup.

IV. Quality-aware assisted composition

Let us assume that, using a mashup-maker tool, the mashup composer can access a component registry C in which each component c_i is associated with a *component descriptor* specifying functional properties [6], [10], and a *quality vector*, $QV_i = [qa_{i1}, qa_{i2}, \ldots, qa_{in}]$, storing the values computed through the metrics associated with a set of component quality attributes. It is thus possible to define the value of a *quality index* for the i-th component (QI_i) as an aggregation of the different qa_i, possibly weighed to privilege some quality attributes over others. The QI computed for each single component is the basis for the generation of quality-aware recommendations. When the user starts the composition, and the workspace is empty, components are first ranked based on the value of their QIs. Each time the user adds a new component, all the other components in C are classified and ranked according to the criteria that we describe in the following.

A. Component Compatibility and Similarity

Inconsistencies at the composition logic level can cause a low quality of the mashup. When the user extends the current composition with new components, the *compatibility* of such components with the current status of the composition is an important factor; components in C can be therefore analyzed and scored accordingly. In particular, compatibility can be estimated as the combination of syntactic and semantic compatibility:

- *Syntactic compatibility* checks for *type compatibility* among the operation parameters exposed by one candidate component and the parameters of all the other components already in the composition.
- *Semantic compatibility* subsumes the syntactic compatibility, and checks whether the operation parameters of the candidate component belong to the same (or a similar) semantic category of at least one of the operations exposed by the other components already in place.

Component similarity can also help determine in which measure some components in C can functionally substitute the ones already in the composition. This property can be useful

to recommend alternative components that can improve the composition quality.

Compatibility and similarity can be verified on the basis of annotations that enrich the components' descriptor with semantic categories (based on ontological entities) for both operations and parameters. Examples of such annotations in a component descriptor are represented in Figure 2, where the specification of events and operations are enriched with "similarity" tags expressing semantic meanings.

B. Aggregated Quality

Compatibility and similarity can ensure that a more consistent composition is produced. An estimation of the mashup quality can be then achieved by aggregating the QIs of the individual components. In particular, as soon as new components are added into the composition workspace, the compatible components in C can be ranked based on their capacity to increase the quality of the overall mashup.

The quality of the overall composition cannot be simply quantified as a plain aggregation of the individual QIs; rather the aggregation must take into account the *role* that each component plays in the composite logics. By analyzing the most popular mashups published on programmableWeb.com we identified a number of composition patterns, in which two component roles emerge [5]:

- In most cases one component assumes a central role in the composition, being the service the user interacts with the most. We call this component *master*. The master is the starting point of the user interaction causing the other connected components to react and synchronize accordingly, in practice implementing a "star" composition pattern. For example, in the mashup reported in Figure 1, the Facebook component is master with respect to Flickr and GMaps, since the selection of one of its displayed concerts propagates parameters that trigger the visualization of related images in Flickr, and the display of the concert locations in GMaps.
- A *slave* is then a component whose behavior depends on another component; its state is mainly modified by events originating in a master component. Many mashups also allow the user to interact with slave components. However, the filtering of the content displayed by slave components depends on the user's interaction with the master component and occurs by automatically propagating synchronization information from the master to the slaves. In Figure 1, Flickr and Gmaps are examples of slave components.

It emerges that master components, being central points of synchronization, have a major influence on the mashup quality - a master could even degrade the quality of the other components that depend on it. Therefore, the aggregation of the different QIs must be adequately weighted.

In order to identify master-slave dependencies during mashup composition, we model a mashup as an directed weighted graph $G = (V, E)$, where each vertex $v_i \in V$ represents a component and each arc $e_{ij} \in E$ represents that one binding is defined between the two connected components, and therefore that v_i is master with respect to v_j.

Based on the analysis of all the paths in the composition graph, which reflect the defined bindings, for each component v_i we then define the *centrality* of a component v_i, $Centrality_i$, as a variant of the *betweenness centrality measure* [13], weighting all the shortest paths inversely proportional to their length [14].

This measure, applied to each component in the composition and normalized with respect to the maximum centrality, provides the weights to be used to aggregate the different QIs. The quality of the overall composition is therefore defined as:

$$QC = \sum_i Centrality_i * QI_i$$

QC is computed every time a new component c_i is added to the composition to identify in which measure c_i and all its similar components increase the quality of the composition. Figure 3a shows the window that in our tool visualizes such recommendations. The window displays on the left the components currently included in the mashup together with an evaluation of their quality expressed in form of stars. Internally, the tool exploits the graph-based representation of such a composition, to identify the existing inter-component dependencies and scoring the composition quality according to the QC metric. If one component is selected, for example Google Maps in figure, the right panel shows a list of components that are similar to Google Maps and compatible with the other components in the composition. These components are ranked by taking into account their capacity to increase the composition quality if replaced to Google Maps. Further details about their quality and their similarity with Google Maps are also given. Quality measures are normalized with respect to a selected scale. For example, in Figures 3a we express measures in a scale from 1 to 5, and visualize them in form of stars.

C. Added Value

In our analysis of mashups, we also tried to capture the concept of *added value* of a composition, conceived as the set of additional features (data, functions, visualizations) that the composition logics introduces with respect to accessing single services separately [3].

Mashups are developed in order to offer a set of functions that we call MFS (Mashup Function Set), and consequently retrieve and give access to a data set, MDS (Mashup Data Set), exploiting a set of visualization mechanisms, MVS (Mashup Visualization Set). Each single component c_i is also characterized by its own data set DS_i, a function set, FS_i, and a set of visualization mechanisms, VS_i [2]. When used within a mashup, smaller, *situational* portions of such sets, SDS_i

[2]Depending on their nature, components may or may not provide all the three layers. For example, map APIs expose data, functions and multiple visualizations. On the other hand, other components may just offer one of these layers. For example, an RSS Feed is a plain data source for which functionality, e.g., feed filtering, and visualizations have to be provided by other components.

(the component's Situational Data Set), SFS_i (the component's Situational Function Set) and SVS_i (the component's Situational Visualization Set) are considered, depending on the specific needs that the mashup is supposed to satisfy [5]. The composition thus provides an added value if the number of features that it offers, at least at one of the three layer (data, function, visualization), is greater than the amount of those offered by the single components, i.e., if

$$\bigcup_i SFS_i \subset MFS \vee \bigcup_i SDS_i \subset MDS \vee \bigcup_i SVS_i \subset MVS.$$

Operatively, to evaluate the added value of the composition at the *data layer*, it is possible to consider the *richness of data formats*, e.g., whether the mashup provides plain, multimedia or social data, which can be assessed on the basis of a classification of components reflecting the nature of their data sets. Also, it is possible to consider the *richness of additional information* deriving from the join queries enabled by the synchronization among the different components. This measure can be quantified as the ratio between the data bindings actually defined among components, and the number of all the possible data bindings. The former are evaluated by analyzing the composition model, where the data bindings actually defined are specified; the latter are instead evaluated by defining and analyzing a compatibility matrix, derived from the component descriptors, to identify all the possible join queries that could be achieved combining output parameters produced by a component with compatible data filtering operations exposed by other components. The added value at the data layer thus aggregates the richness of data formats with the richness of additional information.

The *functionality layer* addresses the *richness of functionality*. Similarly to what we propose at the data layer, this measure can be quantified as the ratio between the number of function bindings actually defined and the number of all the possible function bindings, which can be derived respectively from the components and the composition descriptions.

At the *presentation layer*, we then relate the multiplicity of visualizations with the multiplicity of data sources. In particular, we classify components as *data sources* when they have an own data set, and *viewers*, when they only provide visualizations on top of any external data set[3]. The added value can be assessed by considering the *richness of visualizations*, i.e., the number of different viewers associated with the involved data sources. In fact, in many cases, the analysis of the same phenomenon from the different perspectives that different visualizations can offer can better support decisions. Symmetrically, in other situations the analysis can be improved by aggregating heterogeneous data into a unified visualization. Hence, we also take into account the *cohesiveness of visualizations*, i.e., the capacity of a viewer to convey integrated data.

The aggregated measure of the composition added value can then be achieved by averaging the three distinct values, possibly assigning different weighs to the three layers. Such a measure is especially useful to understand whether the value of the current composition can be increased by adding new components. Figure 3b shows an example of completion recommendation that our tool produces on the basis of the added-value measure. Assuming that the current composition consists of the components listed in the left-hand side of the window, our technique identifies candidate components (reported in the right-hand panel) that can increase the composition value. In the example, the suggested components would provide content of different nature, i.e., multimedia data (YouTube) and user-generated contents (Twitter), and an additional timeline visualization (TimelineJS) that in the example mashup would support the temporal characterization of the music events. In the rest of this section we will show how such recommendations can be better contextualized by taking into account the most frequent composition practices emerging from the analysis of mashup repositories collecting the contributions of communities of mashup composers.

D. Community-Perceived Quality

The notion of recommendations can be further extended to take into account the best practices adopted by communities of mashup developers, so exploiting collaborative filtering mechanisms. If large repositories of components are available, then the most frequent associations among (categories of) components can be mined and exploited to suggest typical composition patterns that get consensus in a given community, and that therefore reflect the users' perception of the quality of the created mashups.

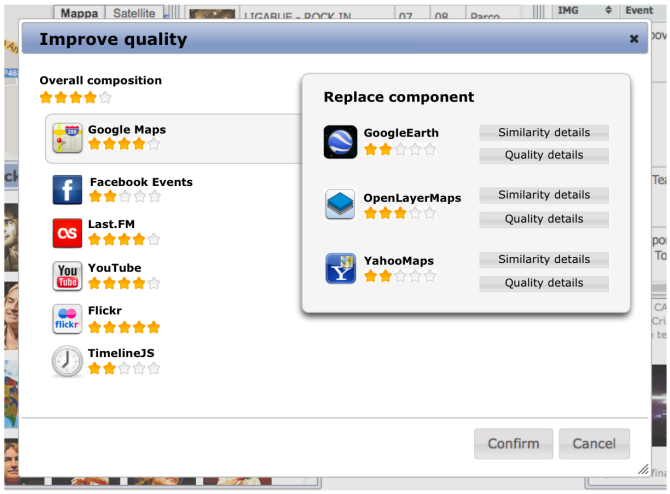
With the exception of few mashup-maker tools available online (e.g., Yahoo! Pipes), it is uncommon for a mashup platform to have large repositories with a number of components and compositions adequate for the mining tasks. In order to perform experiments on this quality dimension, we therefore mined recurrent composition patterns from a data set crawled from programmableweb.com, the largest collection of mashups and mashup components currently available on the Web. From the so-achieved database, we selected mashups with at least two components; then we mined the most recurrent combinations of components. Given the huge number of distinct components (more than 6000 at the time of our experiment), and the difficulty to reproduce such a large variety in a local repository, we identified the most recurrent categories; the extracted association rules thus reflect recurrent combinations at the category level. We thus defined a technique that exploits such extracted knowledge to guide the production of recommendations.

We assume all the components available in the local repository be classified according to the same categories used for the mining tasks. As soon as the state of the composition evolves, the set of categories of the involved components guide the filtering of pertinent association rules, i.e., of those rules where the set of categories in the antecedent part corresponds to the set of categories in the composition. Thus the categories appearing in the consequent part of the rule with higher

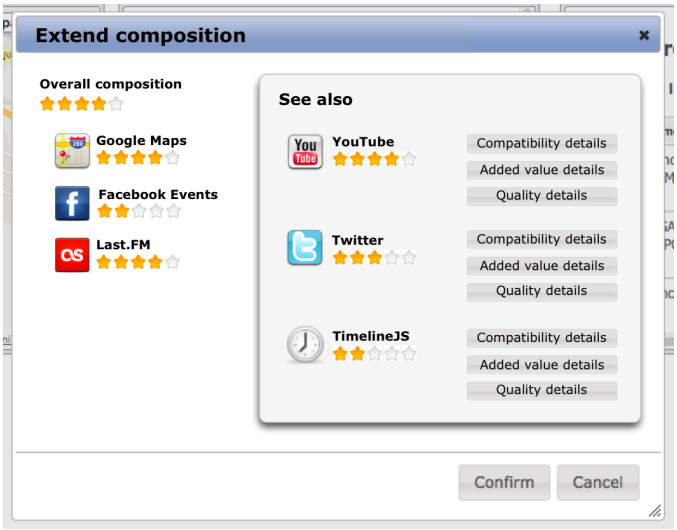
[3]Some components, e.g., GMaps, can fall in both categories.

Iteration no.	Categories already in the composition	Selected Association Rule	Suggested Component
1	Reference	Reference → Photo	Flickr
2	Reference, Photo	Reference, Photo → Social	Twitter
3	Reference, Photo, Social	Reference, Social → Search	Google Ajax Search

TABLE I: Selection of association rules in the construction for the Shahi sample mashup.



(a) Example of quality-based recommendations for *alternative components*.



(b) Example of recommendations for *additional components* based on the assessment of the perceived quality and added-value dimensions.

Fig. 3: Recommendation windows in the PEUDOM visual editor [3].

confidence are projected over the local repository, to identify the components falling into those categories. For example, the recommended components shown in Figure 3b belong to categories frequently associated with the components already included in the composition. At this point, the quality and added-value metrics described above are taken into account to rank the identified components based on their capacity to improve the overall composition.

E. Recommendation procedure: an example

In order to test the effectiveness of our approach, we performed a set of experiments in which we simulated the creation of a number of mashups based on our assisted composition method. We selected a set of well-designed and popular mashups, top-ranked in the programmalbleWeb repository and we used such mashups as benchmarks. More specifically, we used such mashups for the identification of our goal mashups. We wanted indeed to understand in which measure, given a properly selected set of components including the ones in the selected goal mashups, our mechanism would have led to the composition of quality mashups.

In order to identify the set of benchmark mashups, we considered the popularity ranking provided by programmableWeb and we mediated it with the judgment of five evaluators, achieved in a previous study [3] through heuristic evaluation sessions that did not take into account our quality metrics. For each goal mashup, we simulated our assisted composition assuming as starting point the inclusion of one of the components in the considered goal mashup. Our recommendation technique was able to suggests different alternative compositions, that in 80% of the cases were very close to our goal mashups, with a distance of maximum 2 components. In any final composition, the quality was not degraded with respect to the goal mashups.

We here describe one of the performed composition experiments, in which the goal was to create a mashup providing a visual encyclopedia: the idea is to enrich the textual content extracted by an online encyclopedia or dictionary with visual content. The benchmark mashup selected to verify the resulting composition is Shahi[4], a popular mashup which is also one of the top-ranked mashups among those that we have analyzed. In particular, Shahi is a visual dictionary that combines Wiktionary content with Flickr images, and offers search facilities by using Google Ajax Search and Yahoo Image Search.

As described in the previous sections, the assisted composition procedure mainly relies on the component registry, and on the association rule extracted from the analysis of mashup repositories. The component registry contains all the available components together with their descriptors in which operational and quality features are described. The association rules repository instead suggests valuable patterns for combining different components' categories. Once the user selects the first component to be included in the composition, the recommendation procedure starts, and proceeds according to two fundamental steps: i) the selection of the category of the component to adopt in order to enrich the current

[4]http://blachan.com/shahi/

composition and ii) the selection of a specific component, within the selected category, to add. In the second step, the selection of the most suitable component is driven by the compatibility with the components already in place and the potential mashup quality. The two steps are repeated until the user reaches the desired goal, or the algorithm does not find any other components to include in the composition.

For example, in order to build the visual encyclopedia mashup, we consider Wikipedia as the first component; applying iteratively the procedure for the assisted composition we obtain the results summarized in Table I. Here, the second column refers to the categories to which the components already involved in the mashup belong. The third column specifies the association rule that drives the selection of the category of the new component that can extend the composition. When more rules are identified, support and confidence are considered to identify a single rule. In case of more rules with the same value for such parameters, the different possible expansions of the mashups are ranked based on the quality and added value of the components falling in the involved categories. Finally, the fourth column contains the name of the component selected because it results to be the top-ranked, based on the assessment of quality measures.

After the third steps, the recommendation procedure did not find any other association rules to apply for expanding the current composition. Comparing the obtained mashup with our benchmark, Shahi, we noticed that both mashups address the same situational need, but our quality-aware assisted composition also suggested to extend the composition with a "social" component (i.e., the Twitter API), not included in the original mashup. This in a sense proves that the aggregation of different quality dimensions lead to the consideration of different composition solutions that can improve the value of the final composition.

V. Architecture and Implementation

The techniques for quality assessment described in the previous section have been integrated in our tool for mashup development, PEUDOM [6], [9]. Figure 4 illustrates the main architectural components, with particular emphasis on those in charge of executing the quality-based ranking algorithms.

In the platform back-end, the *component registry* stores the component wrappers that enable the platform to invoke the services the components relate to. The registry also includes *component descriptors* specifying teh functional and quality properties of the component that are exploited by the recommendation algorithms. In particular, every time a new component is added in the component registry C:

- The functional properties augmented with semantic annotations are exploited to compute the *Compatibility and Similarity matrices*. More specifically, the `type` and `similarity` attributes in the descriptors are respectively exploited to assess the compatibility and the similarity among all the components in the repository. A semantic reasoner is used for this purpose [5]. The two XML-based matrices are computed at the first use of the platform, and updated every time a new component is added into or dropped from the component registry C.
- The quality annotations specified in its descriptor are used to compute the *quality vector*. A quality vector stores the quality measures achieved by computing metrics, such as those defined in our quality model for mashup components [4], starting from the quality annotations.

The association rules reflecting community-based composition practices are also computed off-line periodically, starting from the data crawled from mashup repositoris, publicly available (such as programmableWeb.com) or local to the adopted mashup platform.

Based on the data described above, the recommendations algorithms are executed every time a component is added into the composition. The *event handler* module intercepts the component addition in the front-end visual environment, and triggers corresponding actions to *i)* update the composition model, and to *ii)* activate the *component recommender*. The component recommender generates the component ranking. It analyzes the association rules, to discover the component categories to recommend for mashup completion, and identifies the components in those categories that are compatible and similar with the components already in the composition. It then exploits a *quality broker* to compute the aggregated quality and the added value indexes, based on the analysis of the quality vectors and of the composition model. The result is a ranking of components, based on the quality and the added-value increment that components can give to the composition under constructions.

All the modules related to the generation of recommendations run on the server. In particular, the component recommender is implemented as a REST service. The event handler is instead a client-side AJAX-based module, which sends requests to the component recommender service. The component ranking, represented in JSON, is then sent back to the client AJAX front-end that finally manages the visualization of the recommendations window.

VI. Related Works

Several works have proposed quality models for Web applications (see, among others, [15], [16], [17], [18]). Few proposals also concentrate on modern Web 2.0 applications. For example, in [19] the authors extend the ISO 9126-1 standard, and discuss the internal quality, external quality, and quality in use of Web 2.0 applications. Our model for component quality [4] is also derived from the quality attributes defined by the ISO standard. We however add a specific perspective that concentrates on the external quality of components, i.e., on the set of properties that affect the component's quality as perceived by the mashup composer. Other works focused on API quality in the more general SOA (Service-Oriented

[5]Our current implementation uses the Pellet reasoner (http://clarkparsia.com/pellet/).

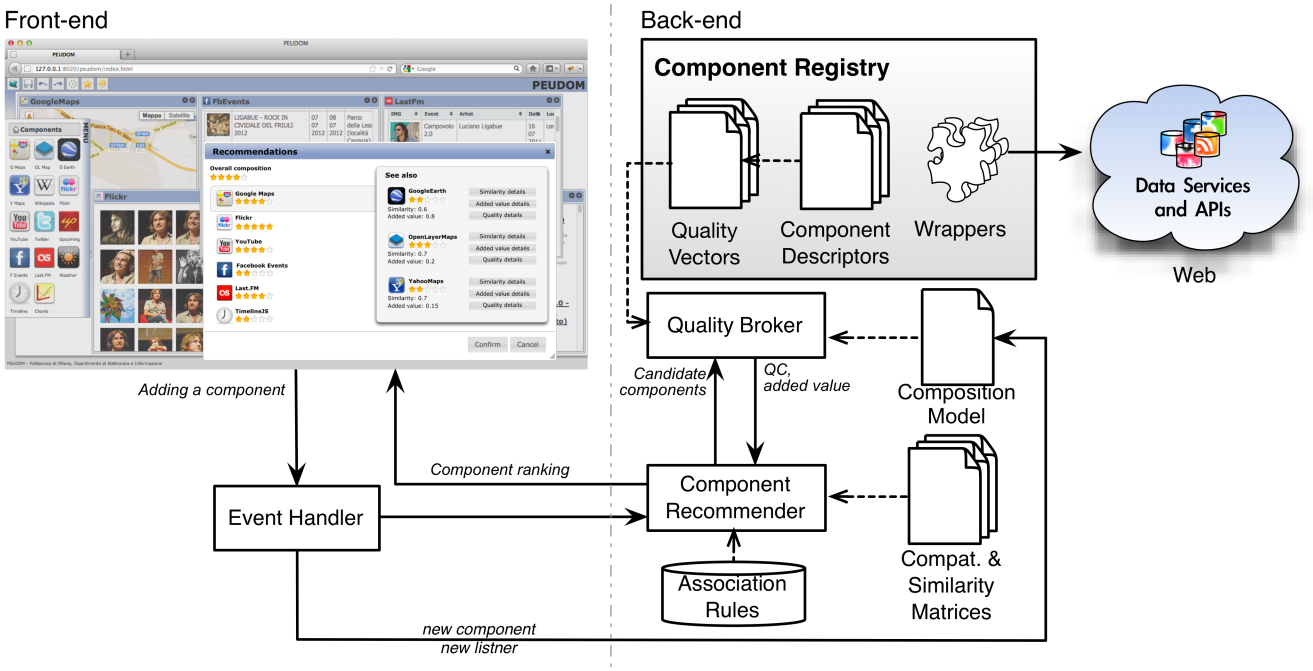


Fig. 4: Modules for quality-aware recommendations in the PEUDOM architecture.

Architecture) domain, by specifically addressing the API ease of use (the so-called *API usability*) [20], [21], [22]. Our approach capitalizes on these contributions but tries to go beyond, since it considers a broader set of external quality factors – not only usability, all having impact on the success of mashup components. In [23] we also proposed a method to select trustworthy, relevant data sources and compose them into a mashup. The proposed model specifically focuses on the quality of sources providing user-generated contents. In this paper we instead try to capture several other issues related to generic services and their composition into mashups.

In relation to service-based applications the literature provides some approaches in which quality is the main driver for service selection and composition (e.g., [24], [25]). In particular, similar to our approach, in [24] the authors assess the overall quality of the final applications by aggregating the quality of the composing services. The importance of considering quality in service composition is also discussed in a recent paper describing the current approaches and the open challenges in quality-aware, service-oriented data integration [2]. However, this paper also highlights the lack of proposals for mashup composition, recognizing the need of specialized approaches. In fact, some works have proposed design-time assistance mechanisms. For example, MashupAdvisor [26] generates recommendations through a probabilistic model that ranks candidate mashup compositions retrieved in a community repository based on their popularity and on AI metrics that identify a maximum utility plan. Also, some of the authors of this paper defined algorithms to query mashup repositories and discover relevant composition knowledge, i.e., reusable *composition patterns*, based on syntactic similarity [27]. None of these works however addresses quality. The assisted composition method described in this paper, being based on quality models that are specific for mashups [5], [3], tries to fill this gap.

VII. Conclusion

This paper presented our perspective on the quality of Web mashups, based on peculiar features of this class of applications. It also illustrated the integration of quality-based recommendations in the mashup development process, to aid (inexperienced) end-users to complete and improve their mashups. We showed the application of our method within our tool for mashup composition. However, we believe that the proposed technique can easily be replicated within other frameworks for mashup composition, provided that adequate descriptions for specifying the component quality and the graph-based composition structure are available.

The selection of the dimensions along which quality is measured and recommendations are generated is based on our experience gained during the last years in the development of mashups and mashup-maker tools. The emphasis on component quality also derives from research conducted in the field of Web service composition [24], [25]. The focus on the mashup integration logic, and especially on the role of components within the composition, is what we consider peculiar for mashups. Therefore we invested several efforts to assess this dimension. Finally, the consideration of the added value and user-perceived quality dimensions is related to the nature of mashups as artifacts that can enable innovation [7]. The composition of different resources should be indeed aimed at introducing innovative uses of the resources themselves. With this respect, the added-value dimension can help guiding

the users to introduce new value. The perceived quality can then help composing applications along directions that are "generally" considered useful by other users too.

Our current work is devoted to validate our quality-aware composition paradigm through experiments with users. In a context of a previous study focusing on the usability of our mashup tool [6], we collected some data about the satisfaction of the users with a preliminary version of our quality-aware recommendation mechanism. The users showed a good level of satisfaction and found the mechanism useful to help them select components to complete the mashup. We are planning more formal validation experiments, also focusing on our last extensions towards the added value and the perceived quality. We will also compare our quality-based approach with other recommendation mechanisms. In this respect, we already investigated the potential of generic, quality-agnostic recommender systems [28]. We will therefore observe the users when exposed to the two kinds of approaches, quality-aware vs. quality-agnostic, and compare their performance to assess the effect of taking into account quality.

ACKNOWLEDGMENT

The authors would like to thank all the students that took part to the implementation of the quality-based recommendation framework for mashup composition. This research work is supported by the Search Computing ERC IDEAS project (SeCo), the MULTIPLE project (TIN2009-13838) and the FPU program (AP2007-03731).

REFERENCES

[1] T. Nestler, "Towards a mashup-driven end-user programming of soa-based applications," in *iiWAS*, G. Kotsis, D. Taniar, E. Pardede, and I. K. Ibrahim, Eds. ACM, 2008, pp. 551–554.

[2] S. Dustdar, R. Pichler, V. Savenkov, and H.-L. Truong, "Quality-aware Service-Oriented Data Integration: Requirements, State of the Art and Open Challenges," *SIGMOD Record*, vol. 41, no. 1, pp. 11–19, 2012.

[3] C. Cappiello, F. Daniel, A. Koschmider, M. Matera, and M. Picozzi, "A Quality Model for Mashups," in *ICWE*, ser. LNCS, vol. 6757, 2011, pp. 137–151.

[4] C. Cappiello, F. Daniel, and M. Matera, "A quality model for mashup components," in *ICWE*, ser. LNCS, vol. 5648, 2009, pp. 236–250.

[5] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso, "Information quality in mashups," *IEEE Internet Computing*, vol. 14, no. 4, pp. 14–22, 2010.

[6] C. Cappiello, F. Daniel, M. Matera, M. Picozzi, and M. Weiss, "Enabling End User Development through Mashups: Requirements, Abstractions and Innovation Toolkits," in *IS-EUD*, ser. LNCS, vol. 6654, 2011, pp. 9–24.

[7] F. Daniel, M. Matera, and M. Weiss, "Next in mashup development: User-created apps on the web," *IT Professional*, vol. 13, no. 5, pp. 22–29, 2011.

[8] A. Namoun, T. Nestler, and A. D. Angeli, "Conceptual and usability issues in the composable web of software services," in *ICWE Workshops*, ser. Lecture Notes in Computer Science, vol. 6385. Springer, 2010, pp. 396–407.

[9] C. Cappiello, M. Matera, M. Picozzi, G. Sprega, D. Barbagallo, and C. Francalanci, "Dashmash: A mashup environment for end user development," in *ICWE*, ser. LNCS, vol. 6757, 2011, pp. 152–166.

[10] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera, "A framework for rapid integration of presentation components," in *Proc. of WWW*. ACM, 2007, pp. 923–932.

[11] F. Daniel, F. Casati, B. Benatallah, and M.-C. Shan, "Hosted universal composition: Models, languages and infrastructure in mashart," in *ER*, ser. Lecture Notes in Computer Science, A. H. F. Laender, S. Castano, U. Dayal, F. Casati, and J. P. M. de Oliveira, Eds., vol. 5829. Springer, 2009, pp. 428–443.

[12] S. Yu and C. J. Woodard, "Innovation in the programmable web: Characterizing the mashup ecosystem," in *ICSOC Workshops*, ser. LNCS, vol. 5472, 2008, pp. 136–147.

[13] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, pp. 35–41, 1977.

[14] U. Brandes, "On variants of Shortest-Path Betweenness Centrality and their Generic Computation," *Social Networks*, vol. 30, no. 2, p. 136145, 2008.

[15] C. Calero, J. Ruiz, and M. Piattini, "A Web Metrics Survey Using WQM." in *ICWE*, ser. LNCS, vol. 3140, 2004, pp. 147–160.

[16] L. Olsina, P. Lew, A. Dieser, and B. Rivera, "Using Web Quality Models and a Strategy for Purpose-Oriented Evaluations," *J. Web Eng.*, vol. 10, no. 4, pp. 316–352, 2011.

[17] M. Matera, F. Rizzo, and G. T. Carughi, "Web Usability: Principles and Evaluation Methods." in *Web Engineering*. Springer, 2005, pp. 109–142.

[18] A. Fernandez, S. Abrahão, and E. Insfran, "A Web Usability Evaluation Process for Model-Driven Web Development," in *Proc. of CAISE 2011*, ser. LNCS, vol. 6741, 2011, pp. 108–122.

[19] P. Lew and L. Olsina, "Instantiating Web Quality Models in a Purposeful Way," in *Proc. of ICWE 2011*, ser. LNCS, vol. 6757, 2011, pp. 214–227.

[20] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *Proc of VL/HCC*. IEEE Computer Society, 2004, pp. 199–206.

[21] B. Ellis, J. Stylos, and B. Myers, "The factory pattern in api design: A usability evaluation," in *Proc. of ICSE 2007*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 302–312. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2007.85

[22] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Improving documentation for esoa apis through user studies," in *IS-EUD*, ser. LNCS, vol. 5435, 2009, pp. 86–105.

[23] D. Barbagallo, C. Cappiello, C. Francalanci, M. Matera, and M. Picozzi, "Informing Observers: Quality-driven Filtering and Composition of Web 2.0 Sources," in *Proc. of BEWEB*. ACM, 2012, p. In print.

[24] M. C. Jaeger, G. Rojec-Goldmann, and G. Mühl, "Qos aggregation in web service compositions," in *Proc. of EEE 2005*. IEEE Computer Society, 2005, pp. 181–185.

[25] Q. A. Liang, X. Wu, and H. C. Lau, "Optimizing Service Systems Based on Application-Level QoS," *IEEE T. Services Computing*, vol. 2, no. 2, pp. 108–121, 2009.

[26] H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin, "Mashup advisor: A recommendation tool for mashup development," in *ICWS*. IEEE Computer Society, 2008, pp. 337–344.

[27] S. R. Chowdhury, F. Daniel, and F. Casati, "Efficient, interactive recommendation of mashup composition knowledge," in *ICSOC*, ser. Lecture Notes in Computer Science, G. Kappel, Z. Maamar, and H. R. M. Nezhad, Eds., vol. 7084. Springer, 2011, pp. 374–388.

[28] P. Cremonesi, M. Picozzi, and M. Matera, "A Comparison of Recommender Systems for Mashup Composition," in *Proc. of RSSE 2012*. IEEE Press, 2012, p. In print.