Hybrid Techniques for Web APIs Recommendation

Michele Melchiori
Dept. of Information Engineering
Università degli Studi di Brescia
Brescia, Italy
melchior@ing.unibs.it

ABSTRACT
Currently on the Web there is a proliferation of software components with public APIs that made available ready-to-use functionalities and access to contents. These APIs can be exploited to build applications, called mashups, that are gaining a lot of interest as an opportunity to integrate contents and application logics from independent sources in short-living situational applications. Although existing mashup frameworks, like Google’s Mashup Editor or Yahoo Pipes, provide valid support for development of new applications from given content sources, designers often have to deal with large component repositories in which APIs for components are usually featured by semantic and technological heterogeneities. This situation creates disorientation in a designer that has to select manually components and finding the better way to compose them in a mashup. As a contribution to solve this problem, we introduce hybrid techniques that combine semantics and collective knowledge to organize component descriptors in a registry according to coupling links. Then, we discuss how this organization can be exploited to implement proactive component recommendation for the design of mashup applications.

Categories and Subject Descriptors
H.1 [Information Systems]: Miscellaneous; D.2.13 [Reusable Software]: Reusable Software—Reusable libraries

Keywords
component registry, semantic description, API tagging, API linking, mashup

1. INTRODUCTION
Recently, developers has started to produce Web-based application that combines existing content, data or application functionality made available on the Web from third parties. These applications, called mashups [11], usually meet the need to create situational applications, also short living, to implement a solution with a low development effort. Moreover, they make leverage on the quality of available sources of data or functionalities, to create new, added value, applications by combining these sources.

Various systems and environments to build mashup solutions have been made available, like Yahoo’s pipes or IBM Lotus Mashup. Although these frameworks provide valid solutions to make it easier to compose new applications, designers often have to deal with very extensive repositories of heterogeneous components from which select and integrate components.

On the one hand, an important requirement in mashup development is the availability of techniques and tools for easy selection and composition of available components. On the other hand, the interest in mashups has raised the availability of a huge quantity of components and, currently, their number is more and more growing. As remarkable examples, in ProgrammableWeb.com, an on-line registry of mashups and APIs, nearly 3000 mashup components are listed till now, but the number is growing, while Yahoo! Pipes contains over 20,000 user-defined components.

Currently, mashup components are manually selected and combined, and a support for their effective selection and composition is therefore strongly demanded. Many efforts, in the last years, have been devoted to semantic service discovery and composition, and several ideas could derive from that experience (see for example [9]). However, the composition of Web APIs presents specific peculiarities [20], that distinguish mashup development from service composition and must be taken into account. For example, components can be of different types (not just Web services) and can generate/react to events that must be modeled and handled. Another well known issue is the semantic heterogeneity of component descriptions (in terms of I/O variables, operations, events). It is therefore crucial to abstract from underlying heterogeneities [1, 5, 12] that hinder the discovery of components as well as their composition. In this context, approaches and techniques to proactively recommend mashup components to be combined in an application and
to support their selection, constitute important issues.

Requirements and techniques for semantic-based mashup design have been presented in our previous work [4]. In that paper, a framework for mashup conceptual design, SMASHAKER (Semantic MASHup shaKEr), has been introduced. This framework provides: (i) a conceptual model for mashup component that abstract from the implementation aspects and, specifically, include semantic annotation with respect to domain ontologies; (ii) a Mashup Component Repository (MCR) in which components are registered, semantically annotated with reference to ontological concepts and described as abstraction of the underlying APIs.

In this paper, we focus on the organization of components in the MCR for recommendation purposes. In SMASHAKER, APIs are firstly modeled as component descriptors, then they are automatically organized in the MCR according to semantic relationship. To this purpose, in this paper we firstly recalls the definition of component descriptor and the main features of the SMASHAKER approach. Then, we introduce new hybrid techniques that combine semantics and collective knowledge to organize the components in the registry according to similarity links and coupling links, to suggest possible patterns of composition. Components in the MCR can be searched and navigated through the links. Finally, we discuss how these links can be exploited for interactive, computer-aided design of mashup applications. With respect to our previous work, this paper mainly proposes the hybrid techniques to evaluate semantic links and organizing component descriptors in the MCR.

The paper outline is the following. In Section 2, some relevant literature and approaches about component modeling and API composition are discusses; Section 3 presents the main features of the SMASHAKER approach; component descriptors based on semantic annotation are presented in Section 3.1; Section 4 describes the hybrid techniques to automatically discover semantic links between components to the in the MCR; finally, in Section 6 we give concluding remarks and future research directions.

Motivating example. Consider a simple scenario, where a software designer has to build a Web site for promoting social events in a given city. Various mashup components could be required to build this application such as: (i) to obtain information about events and locations where they are organized, such as Boomloopapi (http://code.google.com/p/boomloopapi/); (ii) to search for available hotels in a city, such as Booking.com API; (iii) to display on a map the position of the hotels to allow participants to choose the accommodation according to their preferences and as closer as possible to the concert location, such as Google Maps (http://www.google.com/maps). However, a mashup registry may contain several components providing similar functionalities, such as Eventful (http://eventful.com), to display event related information, or YahooMap, for map visualization. Different components could have slightly different I/O variables and tools that are able to quantify how much the components are similar between them or wit respect to a designer request can be very useful for her/him. On the other hand, once the designer selects the Boomloopapi component to be inserted in the mashup, the system could suggest further components that can be combined with the Boomloopapi component on the basis of their semantics. For this purpose, metrics to quantify the level of coupling between components are introduced in the following.

2. RELATED WORK
With reference to the considered problem, literature and approaches on models for APIs, their selection and composition are relevant and discussed in the following.

Modeling of APIs and software components.

The importance of adopting semantic models for describing unstructured and heterogeneous APIs and recommending the composition of mashups has been highlighted in [1, 5, 9, 12]. In [1] a formal model based on datalog rules is proposed to capture all the aspects of a mashup component. Mashups are combined into patterns and a notion of inheritance relationship between components is introduced. With respect to the proposal given in our paper, this work do not provide environments with on-the-fly support for mashup composition. In [5] an abstract component model and a composition model are proposed, expressed by means of an XML-based language, and mashup is built according to a publish/subscribe mechanism. A composition model is also proposed, where components are coupled according to a publish/subscribe mechanism, where events raised from a component are associated to operations of other components in the final mashup application. We agree with this vision and we better focused on the design of a tool environment for interactive and proactive suggestion of semantic-enhanced mashup components. The SA-REST language [12] allows semantic annotation of RESTful service descriptions and it is proposed in the context of semantic mashups. Data mediation between heterogeneous services is addressed by means of a mechanisms called lowering and lifting. In SA-REST, semantic annotation of RESTful services should be supplied by their providers. Mashup composition is manual and no component selection strategies or metrics to support it are proposed.

A model-driven approach for collaborative service mashup design has been discussed in [19], where integration of RESTful and WS-* Web Services is realized by means of a service mashup meta-model. RESTful services are often compared against WS-* Services and related standards (WSDL, SOAP, WSBPEL) [15] that are used to deeply specify service behavior to perform a more in-depth integration of application logics at enterprise level [2]. No semantic-based metrics are proposed to support the designer in the composition.

Mashup building support.

Recent efforts to provide mashup platforms underline the importance of mitigating the burden of mashup composition that requires programming skills, the in-depth knowledge of inputs/outputs formats for manually implementing data mediation in the final composition, the integration of different kinds of components (e.g., components with their own UI, SOAP web services, RESTful web services,
RSS/Atom feeds) through the definition of an abstract component model. A new perspective, known as Enterprise Mashup, allows business users to be empowered in a collaborative environment to quickly develop rich internet applications combining and reusing resources from internal and external information sources [17]. However, building mashup applications starting from multiple available registered components also requires proper support for finding the right ones before composing them.

In [9], a faceted classification of unstructured Web APIs and a ranking algorithm to improve their retrieval are proposed. The classification and searching solution is based on IR techniques. This proposal provides a quite coarse-grain discovery mechanism and adopts components descriptions as published on a specific web site (www.ProgrammableWeb.com), but does not requires further semantic characterization of components. The approach described in [10] addresses the problem of proactive suggestion of components to be combined in a mashup. The MatchUp system is proposed to support a rapid, on-demand and intuitive mashup development. This approach is based on the notion of autocompletion: when the designer selects a component (called mashlet), the system suggests other components to be connected on the basis of recurrent patterns of components in the repository. This is an interesting perspective, different from the one we adopted here since we do not constrain to maintain the history of previous choices. Moreover, we stress the importance of interactive involvement of designer in the development of the mashup application.

Other works provide tools that consider complementary aspects of mashup building. A tag-based navigation technique for composing data mashups to integrate data sources is proposed in [16]. MashMaker [7] suggests widgets that might assist in handling data currently managed by the mashup designer. For example, the tool might suggest adding "map" location or "distance" widgets if the designer currently views a list of addresses. However, it does not provide semantic-based metrics to support mashup composition. The MashupAdvisor [6] system recommends a set of possible outputs for a specific mashup. Each specific output corresponds to some transformation of the data being manipulated by the mashup. After the user has selected one output, MashupAdvisor computes an extension of the mashup in order to achieve the selected output.

**Contribution.** Firstly, SMASHAKER gives a methodological contribution: in fact, it proposes a conceptual framework for the design of mashups according to different phases and, specifically, the first ones concentrate on building an abstract conceptual model of the mashup. Secondly, it introduces techniques for semantic-based search of components and recommendation. In this paper, we extend the SMASHAKER approach by introducing hybrid techniques for organizing components based on both component semantics and collective knowledge constituted by the tagging associated with components by the provider and by the way these components has been associated in mashups by the community of the developers.

### 3. OUTLINE OF THE APPROACH

In this section we recall the main ideas of our SMASHAKER approach. Generally speaking, developing a mashup based on composing APIs is a process based of the following phases: (a) suitable API have to be selected from a registry/repository or from the Web; (b) I/O mappings among components and pairing events with operations of different APIs have to be established; (c) programming code to actually glue the selected APIs has to be written to get the final application. SMASHAKER aims to support a designer about the phases (a) and (b), given that it it devoted to APIs selection and to the preliminary steps of composition.

A recommendation system to support mashup development should allow the developer to choose in an effective way the APIs to be used according to different kind of searches.

The APIs we consider here, are published on the Web and are associated with tags and categories, like the ones made available on web in the cited registry ProgrammableWeb.com. In fact, tags are often available from the API Authors or by the community of the developers that use it and therefore can be exploited to characterize single APIs and mashups.

![Figure 1: The SMASHAKER approach for mashup construction.](image-url)
Two different roles are involved in the mashup developing phases: (i) the provider of the mashup component and (ii) the mashup designer, who selects and combines the components to build a mashup.

In our vision, the component APIs are semantically annotated, classified and made available to be assembled in the conceptual mashup through the following steps, that are shown in Figure 1.

- **Semantic annotation.** Each available component is semantically described by annotation of its API. In this phase, the meanings of APIs are expressed by associating API elements (inputs/outputs/operations) to concepts defined in domain ontologies. The result of this phase is a collection of semantic descriptors that abstract from underlying concrete implementations of the component APIs. Each semantic descriptor has a reference to the URI of the original API. Moreover, the tags associated with the APIs

- **Matching and linking of semantic descriptors.** Semantic-based matching techniques are applied to the semantic descriptors previously defined to establish automatically coupling links between component descriptors. The links, as result of this phase, are stored in a Mashup Component Repository (MCR) to be available for the following phase.

- **Component recommendation.** Similarity and coupling links are exploited to obtain proactive recommendation of MCR components for supporting: (i) proactive suggestion of component descriptors ranked with respect to their similarity with the mashup designer’s requirements; (ii) interactive support to mashup designer for component composition, according to an exploratory perspective. The result of this phase is a conceptual mashup, where component descriptors are properly connected establishing relationships between: (a) one event and one operation, (b) pairs of I/O parameters, between a pair of descriptors.

### 3.1 Component modeling

In this section we introduce the descriptor of a component, required to: (a) abstract from technological details of the component like the invocation protocol and the message format; (b) to semantically characterize the component by annotating it.

**Component Descriptor.** A component is published on the Web by supplying its API and is often endorsed with a GUI. A well known example of component is Google Map, mainly used to visualize information with a geographical component on maps. An API exposes its signature, composed of the programmatic inputs, outputs and operations to permit to invoke the component functionalities. The interactions of a user with the component GUI change the component properties (e.g., the zoom level of a map). Interactions are modeled through events, whose outputs represent the properties changed during the interaction with the API.

Starting from the API description given by the provider, a semantic annotation scheme must be provided to enable semantic characterization of the functional aspects of the component, in an independent way from the specific terminology used in the API signature. In annotated APIs, the names of operations, operation I/Os and event outputs are associated with concepts defined in from reference domain web ontologies, that are ontologies available on the Web for a particular domain. If a suitable ontology is not available, it must be created first, for example using common editors (such as Protégé OWL). Semantic annotation of Web APIs is performed with the support of proper tools. Moreover, components are usually published with additional information to further characterize them: a set of tags and a set of one or more categories used to classify components according to standard taxonomies. A component descriptor includes tags and categories, and is formally defined as follows.

A component descriptor $SD_i$ is formally defined as a tuple:  

$$\langle CAT(SD_i), OP(SD_i), EV(SD_i), TAG(SD_i) \rangle$$  

where $CAT(SD_i)$ is a set of categories, $OP(SD_i)$ is a set of operations, $EV(SD_i)$ is a set of events and $TAG(SD_i)$ is a set of tags. Each operation $op_k \in OP(SD_i)$ is described by an operation name $op_k$, operation inputs $IN(op_k)$ and operation outputs $OUT(op_k)$, that are annotated with concepts taken from the reference domain ontologies. Each event $ev_k \in EV(SD_i)$ is described by the set of ontological concepts which annotate the event outputs $OUT_{ev}(ev_k)$ as well.

**Example 1.** A descriptor can be encoded as a XML document according to a format inspired by the SAWSDL proposal [8] as shown in Figure 2, that shows a semantically annotated descriptor for a component called MapViewer for map visualization. On line 3 the namespaces of domain ontologies used for the semantic annotations are imported. In this example, Travel.owl is an OWL-based ontology available on the Web. Operation names, operation I/Os and event outputs are associated through the semanticRef tag, with URIs that identify ontology concepts. For instance, the first input parameter of the show operation (line 13) is annotated with the concept whose URI is: "Travel.owl#address". The address attribute in the operation (resp., event) element (see, for example, again line 13) refers to the operation name (resp., the event name) in the mashup component API.

### 4. SEMANTIC MATCHING TECHNIQUES FOR SIMILARITY AND COUPLING EVALUATION

In our approach, component descriptors are organized in the MCR, to support component recommendation and registry navigation by following the links established among descriptors. In particular, descriptors are related in two ways:

2. http://sites.google.com/site/ontotravelguides/Home/ontologies
We introduce hybrid matching techniques that use two sources of evidence to establish automatically links. Firstly, semantic matching techniques and algorithms are used. Then a tag-based validation phase is applied as explained in Section 4.1 to validate the result obtained by the previous step.

In particular, a functional similarity degree coefficient and a coupling degree coefficient have been defined for semantic matching (summarized in Table 1 and Table 2). Similarity evaluation is performed in two steps: (i) two descriptors, $SD_T$ and $SD_C$, are compared on the basis of compatibility of their categories; in particular, at least one category in $SD_C$ has to be the same or more specific than a category in $SD_T$; (ii) if compatibility of categories is verified, then a matching model based on the computation of the CAff function is used to quantify the degree of similarity. The CAff $\in [0..1]$ uses standard techniques to evaluate the similarity between two ontological concepts. Functional similarity $Sim_{FD}(SD_T, SD_C)$ between $SD_T$ and $SD_C$ descriptors is a linear combination of the similarity of their operations $OpSim_{TO}(SD_T, SD_C)$ and the similarity of their events $EvSim_{TO}(SD_T, SD_C)$. Formulas to evaluate such similarities are summarized in Table 1. The building block of these expressions is the Dice coefficient [18] used in Information Retrieval. In the formulas, we used the notation $op_X \in OP_X$ (resp., $ev_X \in EV_X$) to denote the operations (respectively, the events) of component $SD_X$, $IN(op_X)$ (respectively, $OUT(op_X)$, $OUT_{in}(ev_X)$) to denote the set of inputs of the $op_X$ operation (resp., the set of outputs of their categories).
The coupling degree between two descriptors SD\textsubscript{T} and SD\textsubscript{C} is computed to quantify how much events raised from SD\textsubscript{T} can be caught by operations in SD\textsubscript{C}. This is measured through the \textit{Coupling} coefficient shown in Table 2. Actually, in the mashup composition we are not usually interested in having an operation for every single event that is raised by a component. Often, we are only interested in a subset of the events, while another one reacts to the rest. Therefore, in the evaluation of coupling between descriptors we consider a subset of the events of SD\textsubscript{T} (denoted with E\textsubscript{T}). This coefficient is asymmetric as the functional similarity coefficient. As for the \textit{Sim\textsubscript{IO}} coefficient, this formula is based on the Dice coefficient.

A similarity link from SD\textsubscript{T} to SD\textsubscript{C} is established, and stored in the MCR, if \textit{Sim\textsubscript{IO}}(SD\textsubscript{T}, SD\textsubscript{C}) \geq \theta_{\text{sim}}. A coupling between the event ev\textsubscript{T} of SD\textsubscript{T} and the operation op\textsubscript{C} of SD\textsubscript{C} is established if \textit{Coupl\textsubscript{IO}}(ev\textsubscript{T}, op\textsubscript{C}) \geq \theta_{\text{coupl}}. The thresholds \theta_{\text{sim}} and \theta_{\text{coupl}} are experimentally set during a training phase.

In the MCR, descriptors and links can be seen as forming a graph, where nodes are component descriptors and directed edges represent similarity or coupling between descriptors. Directed edges are due to the asymmetry of coupling and similarity coefficients. A similarity link is labeled with its \textit{Sim\textsubscript{IO}} value and a coupling link is labeled with its \textit{Coupl\textsubscript{IO}} value and with a validation flag, as explained in the following.

4.1 Tag-based validation of Coupling links

Coupling links established between SD\textsubscript{T} and SD\textsubscript{C} are then validated with reference to mashups actually built and described in public mashup repositories. In particular, the co-occurrence in some mashups of pairs of components tagged with tags associated with SD\textsubscript{T} and SD\textsubscript{C} is evaluated as a further source of evidence for validating the composition pattern represented by the coupling link between SD\textsubscript{T} and SD\textsubscript{C}. Intuitively, the semantic matching techniques establish a link on the basis of the similarity of portions of the interfaces of SD\textsubscript{T} and SD\textsubscript{C}. The meaningfulness of the link is then checked with respect to the existent mashups. In this way the available knowledge on existing components actually combined in mashups is used to validate the result of semantic matching techniques.

Each coupling link from SD\textsubscript{T} to SD\textsubscript{C} is set as a \textit{strong} if exists at least a pair of tags, \textsubscript{t}T\textsubscript{i} \in \textit{TAG}(SD\textsubscript{T}) and \textsubscript{t}C\textsubscript{j} \in \textit{TAG}(SD\textsubscript{C}) such that the following conditions hold:

- \textit{Support}(\textsubscript{t}T\textsubscript{i}, \textsubscript{t}C\textsubscript{j}) \geq \gamma_{\text{supp}};
- \textit{Confidence}(\textsubscript{t}T\textsubscript{i}, \textsubscript{t}C\textsubscript{j}) \geq \gamma_{\text{conf}};

Otherwise, the link is set as \textit{weak}. Values \gamma_{\text{supp}}, \gamma_{\text{conf}} are thresholds that set the minimal acceptable values of confidence and support. The coefficients \textit{Support}(\textsubscript{t}T\textsubscript{i}, \textsubscript{t}C\textsubscript{j}) and \textit{Confidence}(\textsubscript{t}T\textsubscript{i}, \textsubscript{t}C\textsubscript{j}), defined in Table 3, are the usual coefficients used in data mining to discover association rules between data items. Specifically, the first one quantifies the frequency of co-occurrence of \textsubscript{t}T\textsubscript{i} and \textsubscript{t}C\textsubscript{j} in the mashup repository; the second one, how strong is the following implication: the presence of \textsubscript{t}T\textsubscript{i} in a mashup implies the presence of \textsubscript{t}C\textsubscript{j} in the same mashup. The higher the values of support and confidence, the stronger is the evidence that developers combine in the same mashup components having some tags featuring SD\textsubscript{T} and SD\textsubscript{C}.

5. PROACTIVE COMPONENT RECOMMENDATION

In this section we discuss how the links among descriptors, that are stored in the MCR, can be exploited for searching, finding and suggesting suitable components to be used for building mashup applications. The presented approach is different and innovative with respect to component search by keywords or by tags since it proposes proactively, and stepwise, to the designer those components that are the most relevant for composition.

The support to mashup designer for component searching and integration is given according to an exploratory perspective, where the designer has not exactly in mind the set of components to use, but new components are suggested on the basis of their links with already selected ones.

- **Initial components selection.** The mashup designer can initially search a component by specifying a request SD\textsubscript{R} in terms of desired categories, operations and I/Os according to a query-by-example modality. Components registered in the MCR are considered. A set of components \{SD\textsubscript{1} \cdots SD\textsubscript{n}\} that present a high similarity with the request and such that at least a category in SD\textsubscript{R} is equivalent or subsumed by a category in each SD\textsubscript{k} are proposed. These components are ranked with respect to the values of \textit{Sim\textsubscript{IO}} with the request.

- **Recommendation of components.** Once the designer selects one of the proposed components, say SD\textsubscript{i}, to include it in the mashup, additional components can be automatically recommended. Recommendation is performed according to similarity and coupling criteria to return two sets of components: (i) components that are similar to the selected one. The rationale is that the designer can consider to substitute the selected component with a similar one, after she/he has compared their features; (ii) components that are coupled with the selected one, to implement additional functionalities in the mashup. Each time the designer selects a component, the MCR content is exploited to suggest the two sets of components related to the selected one.

In particular, the recommended similar components are ranked according to the similarity values labeling the similarity links, and the recommended coupled components are ranked according to the properties of the coupling links: (a) the value of the \textit{strong}/\textit{weak} property; (b) the coupling values labeling the links.
The proposed recommendation model is based on semantic descriptions of mashup components and their organization in the MCR according to similarity and coupling criteria that exploit in a combined way semantic techniques whose results are checked with respect to component tagging. Recommendation techniques that exploit this organization have been proposed.

The SMASHER tool is already under development and future work will be devoted to performing experimentation on sets of real APIs available in Web and to include in the tool the tag-based validation of coupling links. Moreover, we plan to investigate how to extend our hybrid techniques by including semantic tagging of components as discussed in the work [14].

6. CONCLUSIONS AND FUTURE WORK

In this paper we described the extension of a recommendation model for interactive computer-aided mashup design by introducing hybrid techniques, semantic- and tag-based.

Our more general purpose is to support interactive and proactive mashup construction according to an exploratory perspective, where the mashup designer does not know in advance what are the available Web APIs and how they can be combined and for these tasks is supported by a recommender system implementing the techniques presented in the following.

The recommendation model is based on semantic description of mashup components and their organization in the MCR according to similarity and coupling criteria that exploit in a combined way semantic techniques whose results are checked with respect to component tagging. Recommendation techniques that exploit this organization have been proposed.

The SMASHER tool is already under development and future work will be devoted to performing experimentation on sets of real APIs available in Web and to include in the tool the tag-based validation of coupling links. Moreover, we plan to investigate how to extend our hybrid techniques by including semantic tagging of components as discussed in the work [14].

7. REFERENCES

[14] Roberto Mirizzi, Azzurra Ragone, Tommaso Di Noia,


