Web APIs selection for Mashup Interoperability

Devis Bianchini, Valeria De Antonellis, Michele Melchiori

Università degli Studi di Brescia – Dip. di Ing. dell'Informazione
Via Branze 38
25123 Brescia, Italy
{bianchin|deantone|melchior}@ing.unibs.it

Abstract. Mashup is new development style adopted in enterprises for implementing non-mission-critical Web applications, which are created to satisfy a business need and that often are used only for short periods of time, while the need exists. However, the development of mashup requires retrieving, understanding and composing heterogeneous software components often made available as Web APIs. Mashup interoperability[1] is defined as the set of conditions, including technological and organizational ones, to permit designers/developers to create mashups. In this paper, we propose Web API selection patterns as a contribution to enable mashup interoperability.

1 Introduction

Nowadays, enterprises are beginning to realize the benefits provided by enterprise mashup, a new development style for non-mission-critical Web applications which are created to satisfy a business need with a limited development effort. Often, the application is short-lived, being intended to satisfy a specific short-term business situation. Examples of such Web applications are enterprise dashboards, that are used in an enterprise context to improve decision making and locating contents allowing users for getting and consolidating information and manage tasks to support their activities [2].

Generally speaking, mashup applications are built exploiting existing data, UI widgets and functionalities to create new applications and software artifacts that could be also reused as components in other mashups. Often mashup components are made available as Web APIs that are linked (through programmatic coupling) to enable the application logics. In the enterprise context, mashups can be implemented by using either components developed internally to the enterprise, for instance to access customer data, either third party components as geo-coding services. A general problem is therefore allowing a developer to explore and understand the space of available APIs and their relationships. This is usually a difficult task because of: (i) the dynamicity of this space and the large number of available APIs (more than 3000 in programmableweb), (ii) the limited time usually allocated for the development of a mashup application; (iii) the limited skills/expertise of the typical mashup developer,

who should develop a new application by looking for suitable Web APIs according to an exploratory perspective, without a wide knowledge about the available Web APIs and how their linkage can be performed.

With reference to these problems, we present and discuss in this paper the concept of APIs selection patterns to proactively assist a mashup application designer in the mashup developing process, we discuss them in the enterprise application development context emphasizing their contribution with respect to the software reuse lifecycle and, finally, we formalize this concept.

Related work. Several efforts have been devoted to the design of tools which support improved development of mashups [3]. In [4], a faceted classification of unstructured Web APIs and a ranking algorithm have been applied to the programmableweb APIs repository to improve the search mechanism. The classification and searching solution is still based on IR techniques. The MatchUp system described in [5] addresses the problem of suggestion of patterns to link mashups components: when the designer selects a set of components, the system suggests code patterns to connect these components on the basis of recurrent patterns in the repository.

A Web-based interface which supports mashup of semantic-enriched Web APIs is proposed in sMash [6]. Possible mashups are shown as a graph, where each vertex represents an API and an edge between two APIs means that they are mashupable, that is, they can be used together in a mashup.

2 Enterprise Mashup Development Scenario

In the enterprise context, mashup has been adopted as development approach both from business functions and IT departments. Enterprise applications can be divided into: *long term-strategic applications* that are developed by the IT department and *short term-tactical applications* that are required for covering situational business needs, as we stated before. The development of the second type cannot be fully supported by IT departments that generally have limited resources for satisfying application development and support. Moreover, it is usually too expensive to adopt traditional skills/processes for creating short term-tactical applications that have a limed audience and lifetime. As a consequence, mashup paradigm can provide a solution if implemented as a process according to the following phases: (i) IT department creates catalog of components and provides code-free assembly tools; (ii) business users can create and share their own mashup applications for personal or team use – without IT intervention.

On one hand, business functions/users get advantage from this process because they experience more control in addressing their needs. In fact, they can implement situational applications in a shorter time and with a relative independence from enterprise IT departments. On the other hand, IT departments implement non-mission critical applications as mashups because their development requires lower effort, programming skills and shorter time. Beside these considerations, mashups techniques

and tools allow enterprise functions to deal with the typical interoperability problem of integrating heterogeneous data and functionalities.

According to [7] the main reasons to adopt mashups in organizations include: (i) reduce uncertainly and compress timeline in projects; (ii) creating a virtuous cycle of reuse; (iii) enabling quick assembling of applications for new situations. In particular, the presence of an effective cycle of reuse (Fig.1, adapted from [7]) creates the conditions for compressing the lngth of the development phases.

However, building a mashup requires the ability for solving problems and making design choices at different abstraction levels: technology, authentication and privacy on the content, choice of the most suitable components and functionalities, choice of component integration at server or client side.

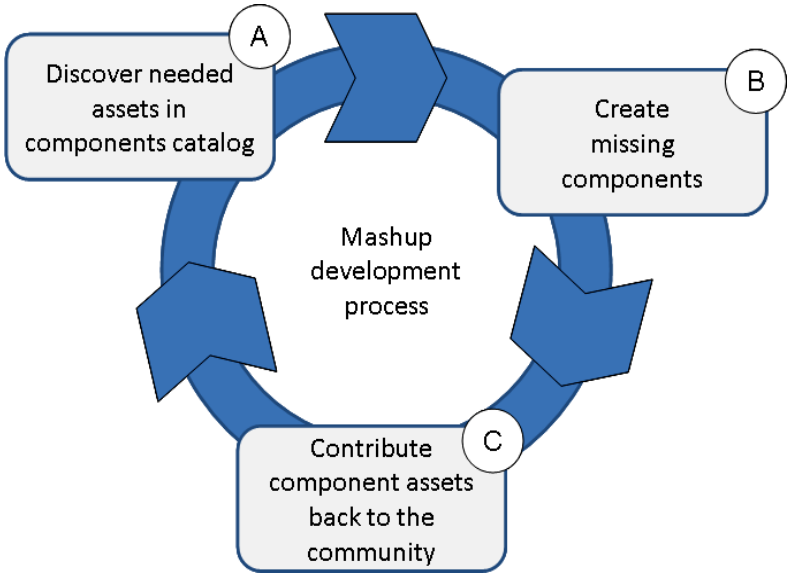

Fig. 1. Cycle of reuse in the mashup development process

.

Specifically, our proposal focuses, as we told, on the problem of selecting and integrating APIs and functionalities. Therefore in the following section, we formalize selection patterns to support interactive and proactive mashup development by helping designer to choose the more suitable APIs. The selection patterns are based on: (i) semantic annotation of Web API descriptions with respect to domain ontologies; (ii) organization of Web APIs based on automated matching techniques apt to establish

semantic links between them according to properly defined similarity and coupling criteria (not presented in this paper).

3 Selection patterns

With reference to the development process of a mashup M (Fig. 1) we identify three relevant useful selection patterns:

– *Search*, to suggest a set of Web APIs that match a given Web API specification W_τ;

– *Completion*, to suggest a list of Web APIs that can be coupled to a given Web API W_τ belonging to M;

– *Substitution*, to suggest a list of Web APIs that can be substituted to the Web API W_τ belonging to M;

Formally, a *selection pattern* is defined as a 4-uple $< W_\tau, m_\tau, \delta_\tau , \leq_\tau >$ where τ is the purpose of the selection pattern, i.e., *Search*, *Completion*, *Substitution*.

The metric m_τ is used to evaluate, respectively, the degree of matching if the pattern is *Search*, the degree of coupling if the pattern is *Completion*, the similarity between each suggested Web API and W_τ if the pattern is *Substitution*. The threshold δ_τ is used to filter out not relevant Web APIs. A Web API W_j is suggested to the designer if $m_\tau(W_\tau,W_j)\geq\delta_\tau$. Finally, $\leq_\tau$ is a function to rank the suggested Web APIs.

With regard to their application to the development process, the selection patterns support both the designer (see Fig.2) in the different phases of the mashup development and in the implementation of the practice of the reuse. With respect to the development phases, *Search* and *Completion* patterns allow the designer, respectively, to search for APIs matching a given specification and to search for APIs that can be coupled with a given one in the application, providing complementary functionalities.

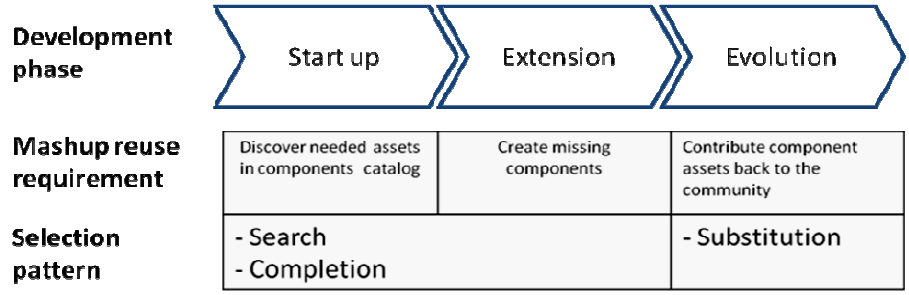

Fig. 2. Selection patterns support to development phases

The *Substitution* pattern allows for substituting APIs in the mashup with other functionally similar APIs under the constraints of minimizing the effort to implement the substitution. In this case, the general purpose of the substitution is that of improving the overall quality of the application by means of replacing those APIs no more available or maintained or with low quality features. The proposed patterns contribute to stimulate the virtuous cycle of reuse highlighted in Fig.1.In particular, given that the *Substitution* pattern is used to improve the quality of the mashup and to make it more stable, it favors the practice of sharing quality mashup within the users/developers community.

3.1 Semantic descriptions of APIs

Selection patterns are based on semantic annotation of Web API descriptions with respect to domain ontologies. In this section, we briefly discuss how this can be obtained. In fact, the design of Web applications from APIs independently provided by third parties is hampered by the semantic heterogeneity of API descriptions (in terms of input/output variables, operations) and by their number, that makes impractical and cumbersome their manual selection. We distinguish two cases in API semantic description: (i) the semantic characterization of available APIs; (ii) the semantic characterization of a requested API, as formulated by the designer.

Semantic annotation of APIs in our framework is obtained according to the steps suggested in the SWEET tool [8]: (a) identification of elements (that is, operations, inputs, outputs) in the unstructured HTML document which represents the API, to produce an hRESTS description; (b) search for ontologies suitable for elements annotation and of taxonomies of categories for API classification; (c) annotation and classification of the API according to the MicroWSMO notation. This last step allows to provide a formal description of an API that is preliminary to the formal definition of

the similarity and coupling metrics[9] used to implement the recommendation patterns.

We define a semantic descriptor W_i for a semantically annotated API as:

$$W_i = <CAT_i, OP_i, EV_i> \quad (\mathbf{1})$$

where CAT_i is a set of categories associated with the API, OP_i is a set of operations, EV_i is a set of events that the API can generate. Each operation $op_k \in OP_i$ is described by the operation name op_k, the operation inputs $IN(op_k)$ and the operation outputs $OUT(op_k)$. Each event $ev_h \in EV_i$ is described by a set of event arguments, used to represent the API state changes triggered by event occurrence. Operation I/Os and event arguments are references to the concepts of the domain ontologies selected during the API annotation process.

The semantic characterization of a request for an API is similar and defined as follows:

$$W_r = <CAT_r, opt_r> \quad (\mathbf{2})$$

where CAT_r is the set of categories featuring the request and $opt_r = <OP(W_r), IN(W_r), OUT(W_r)>$ are the sets of required operation names (resp., input names, output names). With respect to the definition (1), the descriptor W_r has a flattened structure, since the sets $IN(W_r)$ and $OUT(W_r)$ are specified independently from the operation in $OP(W_r)$ they belong to. In fact, according to the exploratory search perspective, the designer could not have a precise idea about the structure of the descriptor to search.

4 Conclusions

In this paper, we have proposed and formalized semantics-enabled recommendation patterns for component selection to support the enterprise mashup development and component reuse process and giving a contribution to implement mashup interoperability. In our proposal, mashups are built from semantically described APIs that are classified and abstracted as semantic descriptors. Suitable metrics have been defined to establishing semantic relationship among APIs and implement the patterns.

Future work includes testing the patterns on real case scenarios. For this the purpose, we have a developed a software tool and we are studying their effectiveness by applying them to publicly available APIs, as the ones from the public registry `www.programmableweb.org`.

5 References

1. J. Palfrey and U. Gasser “Mashups Interoperability and eInnovation” Berkman Publication Series, November 2007, Available at http://cyber.law.harvard.edu/interop/pdfs/interop-mashups.pdf
2. H. Adams (2009). "Executive IT Architect, Mashup business scenarios and patterns". IBM DeveloperWorks. Available at http://www.ibm.com/developerworks/lotus/library/mashups-patterns-pt1/.
3. Ngu, A.H.H., Carlson, M.P., Sheng, Q.Z. and Paik, H.Y. (2010) Semantic-Based Mashup of Composite Applications, *IEEE Trans. On Services Computing*, vol.3, no.1.
4. Gomadam, K., Ranabahu, A., Nagarajan, M., Sheth, A. P. and Verma, K. (2008) A Faceted Classification Based Approach to Search and Rank Web APIs, *6th IEEE Int. Conference on Web Services (ICWS08).*
5. Greenshpan, O., Milo, T. and Polyzotis, N. (2009) Autocompletion for Mashups, *35th Int. Conference on Very Large DataBases (VLDB09)*, pages 538–549.
6. Bin Lu, Zhaohui Wu, Yuan Ni, Guo Tong Xie, Chunying Zhou, and Huajun Chen. (2009) sMash: semantic-based mashup navigation for data api network. In 18th International World Wide Web Conference (WWW2009), pages 1133–1134.
7. Carrier N. (2009) Amplify Your ROI – Getting things done quickly and Economically with Enterprise Mashups, IBM mashup center, Web 2.0 Expo Conference, New York.
8. M. Maleshkova, C. Pedrinaci, and J. Domingue. Semantic annotation of Web APIs with SWEET. In Proc. of 6th Workshop on Scripting and Development for the Semantic Web, 2010.
9. D. Bianchini, V. De Antonellis, M. Melchiori (2011) Semantics-enabled Web APIs selection patterns, In Proc. of 15th International Database Engineering and Applications Symposium (IDEAS 2011) Lisboa, Portugal, pages 204-208.