Collaborative enterprise knowledge mashup

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. In this paper, we describe a proposal of semantic techniques to support enterprise mashup within or across collaborative partners. Mashups are Web applications that integrate data and/or application logics originated from third parties and made available through Web APIs. The aim of the presented techniques is to enable effective searching of mashup components and their composition, by making possible proactive suggestion of mashup components and progressive mashup composition. The approach, called SMASHAKER, includes a model of component semantic descriptor, techniques for building a component repository where semantic descriptors are semantically organized according to similarity and coupling links, and supports an exploratory perspective in mashup development.

1 Introduction

An enterprise mashup is defined as a Web-based application that combines existing content, data or services, from independent sources, by empowering also end users to create and adapt situational application to solve a specific problem. Enterprise mashup focuses on the User Interface integration by extending concepts of Service-Oriented Architecture with the Web 2.0 philosophy [3]. In mashup, data and services are made available through heterogeneous APIs. To better support developers during enterprise mashup development, it is crucial therefore abstract from underlying heterogeneity [1,4].

In this paper, we propose a novel conceptual approach to support progressive construction of collaborative enterprise mashups apt to combine multiple data and/or application logics. The approach is based on semantic annotation of components and semantic matching techniques for their organization, selection and composition.

2 Mashup construction in SMASHAKER

A mashup application is obtained by assembling, possibly with the minimum programming effort, available ready-to-use components. Generally speaking, mashup developing is a process composed of the following phases: (a) component selection from a repository or from the Web; (b) definition of event-operation associations and

I/O mappings among the selected components; (c) development of programming code to actually glue components and their user interfaces to get the final application. Our approach, called SMASHAKER, aims to supports the phases (a) and (b). The output of these phases is what we call a *conceptual mashup*, describing the selected components, associations and mapping. A recommendation system based on this development model should suggest to the developer the components that can be used as alternatives or that can be properly composed in the conceptual mashup.

Different roles must be considered in an enterprise mashup development context [3]:

- the *provider* of the mashup component, that is in charge of supplying the component description with its annotation to enable easy combination with other components;
- the *consumer*, who selects and combines the mashup components to build a conceptual mashup; we refer to this role in the following of the paper with a more specific term, *mashup designer*.

According to the SMASHAKER vision, the component APIs are semantically annotated, classified and made available to be assembled in a conceptual mashup through the following steps, schematically shown in Fig. 1.

Semantic annotation. Each available component is described by means of an annotation of its API. In this phase, the meanings of APIs are made explicit by associating API elements (inputs/outputs/operations) to concepts defined in domain ontologies. The result of this step is a collection of semantic descriptors.

Matching and linking of semantic descriptors. Semantic-based matching techniques are applied to the semantic descriptors previously defined to establish automatically similarity and coupling links between component descriptors.

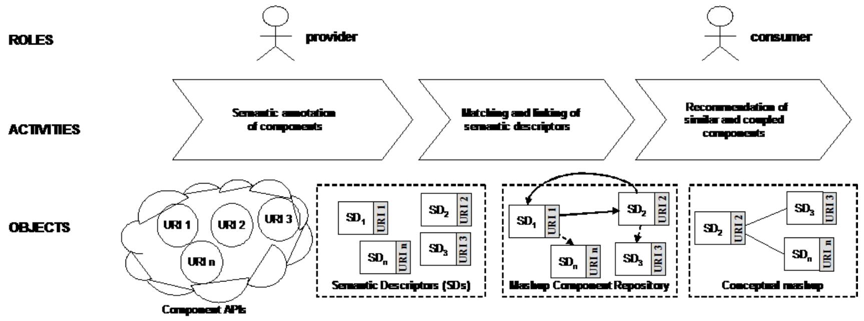
ROLES
provider
consumer
ACTIVITIES
Semantic annotation of components
Matching and linking of semantic descriptors
Recommendation of similar and coupled components
OBJECTS
URI 1
URI 2
URI 3
URI n
Component APIs
SD1 URI 1
SD2 URI 2
SDn URI n
SD3 URI 3
Semantic Descriptors (SDs)
Mashup Component Repository
Conceptual mashup

Fig. 1. The SMASHAKER approach to mashup development.

The links, as result of this phase, are stored in a *Mashup Component Repository (MCR)* to be available for the following step.

Component recommendation. Similarity and coupling links are exploited to obtain proactive recommendation of MCR components. In particular, in this step our framework enables: (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 the exploratory perspective. The result of this step is a conceptual mashup, where component descriptors are properly connected.

3 The component semantic descriptor

To describe a component different elements must be considered. First, it must export a Web API, that is, a list of **operations** (methods signatures). For each operation, its I/O parameters are specified. Second, according to [1], integration of mashup components is typically event-driven: when the user interacts with the UI of components, it reacts with certain state changes and the other components must be aware of such changes to update their UIs accordingly. Each component has a set of **events** and event outputs. An event of a component can be connected to an operation of another component in a publish/subscribe-like mechanism. In a component *semantic descriptor (SD)*, names of operations, operation I/Os and event outputs are annotated with concepts from domain ontologies. Furthermore, a component is associated to a set of **categories**, to provide a domain-driven classification of the component itself.

As an example of component *semantic descriptor (SD)*, Fig. 2 shows a component called *MapViewer* for map visualization similar to the well known *Google Map*. The API of this component includes one operation to show a location on the map by specifying an address, city and country. Moreover, when the user clicks on the map to select a specific point, an event is triggered.

```
<SemanticComponent name="MapViewer_SD"
   url="http://www.mapview.com">

   <categories>
         <item>Mapping</item>
   </categories>
   <operation address="show"
      semanticReference="http://localhost:8080/Travel.owl#showLocation">
   <input
       semanticReference="http://localhost:8080/Travel.owl#Address"/>
   <input
       semanticReference="http://localhost:8080/Travel.owl#Country"/>
   <input semanticReference="http://localhost:8080/Travel.owl#City"/>
   </operation>
   ...
   <event address="selectedCoordinates">
   <output
       semanticReference="http://localhost:8080/Travel.owl#Coordinates"/>
   </event>
</SemanticComponent>
```

Fig. 2. An example of component semantic descriptor

4 The mashup component repository

In our approach, component semantic descriptors are organized in a Mashup Component Repository, to better support collaborative enterprise mashup. In the repository, descriptors are related in two ways: (i) semantic descriptors SD_i and SD_j of components which show an high relatedness between their I/O and therefore can be potentially wired in the final mashup application to the combine functionalities they offers, are connected through a *functional coupling link*; (ii) semantic descriptors SD_i and SD_j of components which perform the same or similar functionalities, are connected through a *functional similarity link*.

To identify coupling or similarity links (resp.), semantic matching techniques can be used. In particular, we have defined the *coupling degree coefficient* $Coupl_{IO}()$and the *functional similarity degree coefficient* $Sim_{IO}()$. These coefficients are based on the computation of name affinity *NA()* between pairs of, respectively, (i) operations names, (ii) I/Os names and (iii) event outputs names used in the semantic descriptors to be matched [2]. *NA()* evaluation is based both on a terminological (domain-independent) matching based on the use of WordNet and on a semantic (domain dependent) matching based on ontology knowledge.

In particular, $Sim_{IO}(SD_R, SD_C)$ between SD_R and SD_C is computed to quantify how much SD_C provides at least the operations and I/Os required in SD_R. and is maximum when SD_C provides at least the operations of SD_R.

$Coupl_{IO}(SD_i, SD_j)$ is maximum if every event *ev* in SD_i has a corresponding operation *op* in SD_j and, in particular, every output of *ev* has a corresponding input in *op*, no matter if SD_j provides additional operations.

4.1 Collaborative mashup developing

The MCR can be exploited for searching, finding and suggesting suitable components to be used in mashup developing. The designer starts by specifying a request SD_R for a component in terms of desired categories, operations and I/Os. A set of components SD_i which present a high similarity with the requested one and such that at least a category in SD_R is equivalent or subsumed by a category in SD_i are proposed. Components are ranked with respect to Sim_{IO} values. Once the consumer selects one of the proposed components, additional components are suggested, according to similarity and coupling criteria: (i) components that are similar to the selected one (the consumer can choose to substitute the initial components with the proposed ones); (ii) components that can be coupled with already selected ones during mashup composition. Each time the consumer changes and selects another component, the MCR is exploited to suggest the two sets of suitable components.

5 Conclusions

In this paper, we described a semantic framework for mashup component selection and suggestion for composition in the context of collaborative enterprise mashup.

Mashup components are semantically described and organized according to similarity and coupling criteria, and effective (semi-)automatic design techniques have been proposed.

6 References

1. Daniel, F., Casati, F., Benatallah, B. and Shan, M.C. (2009) Hosted Universal Composition: Models, Languages and Infrastructure in mashArt, *28th Int. Conference on Conceptual Modeling (ER09),* pages 428-443.
2. Bianchini, D., De Antonellis, V., Melchiori, M. (2008) Flexible Semantic-based Service Matchmaking and Discovery, *World Wide Web Journal*, 11(2):227-251.
3. Hoyer, V. and Stanoevska-Slabeva, K. (2009) Towards a Reference Model for grassroots Enterprise Mashup Environments, *17th European Conf. on Information Systems (ECIS)*.
4. Abiteboul, S., Greenshpan, O. and Milo, T. (2008) Modeling the Mashup space, *Workshops on Web Information and Data Management*, pages 87-94.