Discovery and Reuse of Composition Knowledge for Assisted Mashup Development

Florian Daniel[1], Carlos Rodríguez[1], Soudip Roy Chowdhury[1],
Hamid R. Motahari Nezhad[2], and Fabio Casati[1]
[1]University of Trento, Italy, [2] HP Labs Palo Alto, USA
[1]{daniel,crodriguez,rchowdhury,casati}@disi.unitn.it, [2]hamid.motahari@hp.com

ABSTRACT

Despite the emergence of mashup tools like Yahoo! Pipes or JackBe Presto Wires, developing mashups is still *non-trivial* and requires intimate knowledge about the functionality of web APIs and services, their interfaces, parameter settings, data mappings, and so on. We aim to *assist the mashup process* and to turn it into an interactive co-creation process, in which one part of the solution comes from the developer and the other part from *reusable composition knowledge* that has proven successful in the past. We harvest composition knowledge from a repository of existing mashup models by mining a set of reusable *composition patterns*, which we then use to interactively provide *composition recommendations* to developers while they model their own mashup. Upon acceptance of a recommendation, the purposeful design of the respective pattern types allows us to *automatically weave* the chosen pattern into a partial mashup model, in practice performing a set of modeling actions on behalf of the developer. The experimental evaluation of our prototype implementation demonstrates that it is indeed possible to harvest meaningful, reusable knowledge from existing mashups, and that even complex recommendations can be efficiently queried and weaved also inside the client browser.

Categories and Subject Descriptors

D.2.6 [**Software**]: Software Engineering—*Programming Environments*

Keywords

Assisted mashup development, End user development, Composition patterns, Pattern recommendation, Weaving

1. INTRODUCTION

Mashup tools, such as Yahoo! Pipes (`http://pipes.yahoo.com/pipes/`) or JackBe Presto Wires (`http://www.jackbe.com`), generally promise easy development tools and lightweight runtime environments, both typically running inside the client browser. By now, mashup tools undoubtedly simplified some complex composition tasks, such as the integration of web services or user interfaces. Yet, despite these advances in simplifying technology, mashup development is still a *complex task* that can only be managed by skilled developers.

WWW 2012 Companion, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1230-1/12/04.

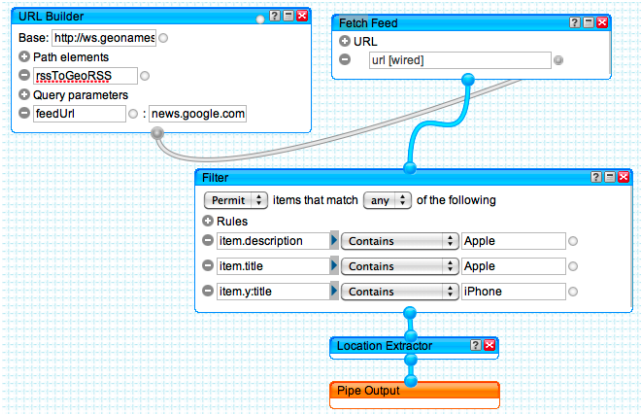

Figure 1: A typical pattern in Yahoo! Pipes

Figure 1 illustrates a Yahoo! Pipes model that encodes how to plot news items on a map. The lesson that can be learned from it is that plotting news onto a map requires enriching the news feed with geo-coordinates, fetching the actual news items, and handing the items over to the map. Understanding this logic is neither trivial nor intuitive.

In order to aid less skilled developers in the design of mashups like the one above, Carlson et al. [1], for instance, leverage on semantic annotations of components to recommend compatible components, given a component in the canvas. Greenshpan et al. [3] recommend components and connectors (so-called glue patterns) in response to the user providing a set of desired components. Elmeleegy et al. [2] recommend a set of components related to a component in the canvas, leveraging on conditional co-occurrence and semantic matching, and automatically plan how to connect selected components to the partial mashup. Riabov et al. [4] allow users to express goals as keywords, in order to feed an automated planner that derives candidate mashups.

We assist the modeler in each step of his development task by means of ***interactive, contextual recommendations of composition knowledge***. The knowledge is reusable *composition patterns*, i.e., fragments of mashup models. Such knowledge may come from a variety of possible sources; we specifically focus on community composition knowledge (recurrent model fragments in a mashup model repository). In this poster, we describe (i) how we mine *mashup composition patterns*, (ii) the *architecture* of our knowledge recommender, (iii) its *recommendation algorithms*, and (iv) its *pattern weaving algorithms* (automatically applying patterns to mashup models).

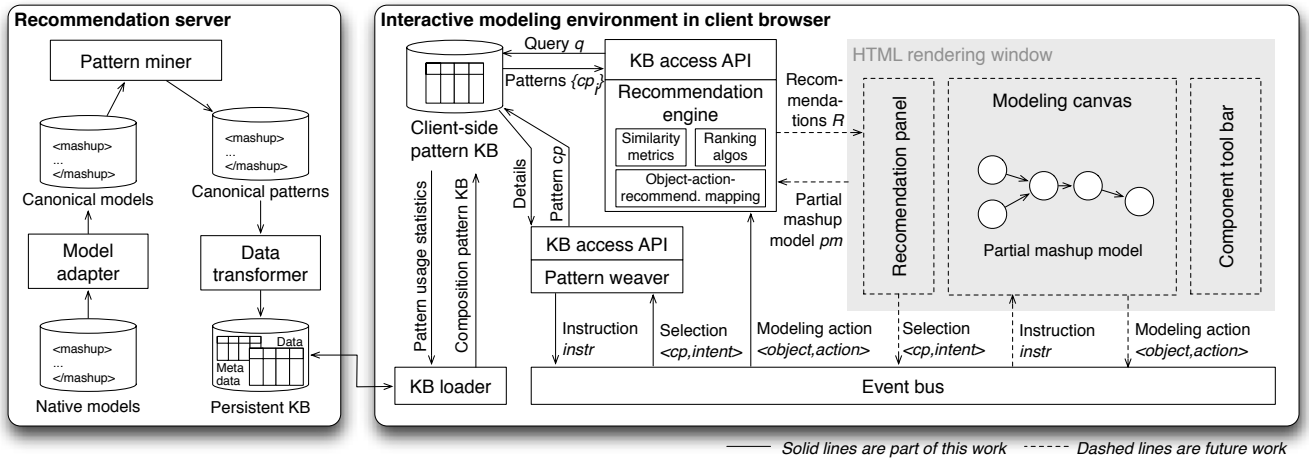

Figure 2: Functional architecture of the composition knowledge discovery and recommendation approach

2. THE RECOMMENDATION PLATFORM

Figure 2 details our knowledge discovery and recommendation prototype. The ***pattern discovery*** logic is located in the server. After converting mashup models into a canonical format, the *pattern miner* extracts patterns, which we store into a knowledge base (KB) that is structured to minimize pattern retrieval at runtime. We support six composition pattern types: *parameter value*, *connector*, *connector co-occurrence*, *component co-occurrence* , *component embedding*, and *multi-component* patterns (cf. Figure 1).

The *interactive modeling environment* runs in the client. It is here where the ***pattern recommendation*** logic reacts to modeling *actions* performed by the modeler on a construct (the *object* of the action) in the canvas. For instance, we can *drop* a component onto the canvas, or we can *select* a parameter. Upon each interaction, the *action* and its *object* are published on a browser-internal *event bus*, which forwards them to the *recommendation engine*. With this information and the partial mashup model *pm* the engine queries the *client-side KB* for recommendations, where an *object-action-recommendation mapping* tells the engine which types of recommendations are to be retrieved. The list of patterns retrieved from the KB are then ranked and rendered in the *recommendation panel*.

Upon the selection of a pattern from the recommendation panel, the ***pattern weaver*** weaves it into the partial mashup model. The *pattern weaver* first retrieves a *basic weaving strategy* (a set of model-agnostic mashup instructions) and then derives a *contextual weaving strategy* (a set of model-specific instructions), which is used to weave the pattern. Deriving the contextual strategy from the basic one may require the resolution of possible *conflicts* among the constructs of the partial model and those of the pattern to be weaved. The pattern weaver resolves them according to a configurable conflict resolution policy.

Our ***prototype*** is a Mozilla Firefox extension for Yahoo! Pipes [6], with the recommendation and weaving algorithms implemented in JavaScript. Event listeners listen for DOM modifications, in order to identify mashup modeling actions inside the modeling canvas. The instructions in the weaving strategies refers to modeling actions, which are implemented as JavaScript manipulations of the mashup model's DOM elements. The server-side part is implemented in Java.

3. EVALUATION

For our experiments we extracted 303 pipes definitions from the repository of Pipes. The average numbers of components, connectors and input parameters were 12.7, 13.2 and 3.1, respectively, indicating fairly complex mashups. We were able to identify patterns of all the types described above. For example, the minimum/maximum support for the *connector patterns* was 0.0759/0.3234, while the one for the *component co-occurrence patterns* was 0.0769/0.2308. We used these patterns to populate our KB and generated additional synthetic patterns to test the performance of the recommendation engine (the sizes of the KBs ranged from 10, 30, 100, 300, 1000 multi-component patterns) [5]. The complexity of the patterns ranged from $3-9$ components per pattern, and we used queries with $1-7$ components. In the worst case scenario (KB of 1000 patterns, approximate similarity matching of patterns), the recommendation engine could retrieve relevant patterns within 608 millisecond – everything entirely inside the client browser. The next step is going online and performing users studies.

Acknowledgment. This work was supported by the European Commission (project OMELETTE, contract 257635).

4. REFERENCES

[1] M. P. Carlson, A. H. Ngu, R. Podorozhny, and L. Zeng. Automatic mash up of composite applications. In *ICSOC'08*, pages 317–330, 2008.

[2] H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin. Mashup advisor: A recommendation tool for mashup development. In *ICWS'08*, pages 337–344, 2008.

[3] O. Greenshpan, T. Milo, and N. Polyzotis. Autocompletion for mashups. *VLDB'09*, 2:538–549, 2009.

[4] A. V. Riabov, E. Boillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan. Wishful search: interactive composition of data mashups. In *WWW'08*, pages 775–784, 2008.

[5] S. Roy Chowdhury, F. Daniel, and F. Casati. Efficient, Interactive Recommendation of Mashup Composition Knowledge. In *ICSOC'11*, pages 374–388, 2011.

[6] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati. Baya: Assisted Mashup Development as a Service. In *WWW'12*, 2012.