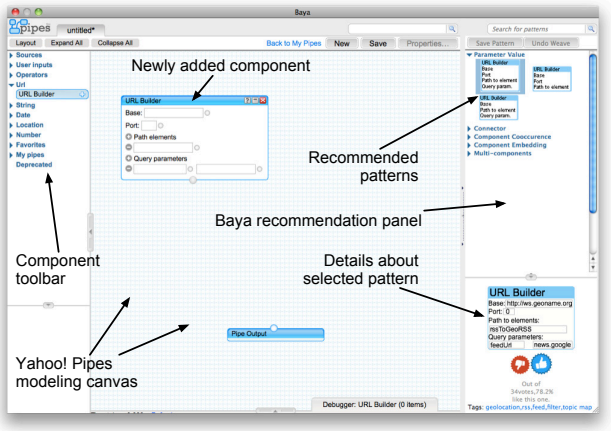

Figure 3: Screenshot of Baya in action.

mashup model. Each of the instructions in the weaving strategy refers to a modeling action, where modeling actions are implemented as JavaScript manipulations of the mashup model's DOM elements. Both the weaving strategies (basic and contextual) are encoded as JSON arrays, which enable us to use the native `eval()` command for fast and easy parsing of the weaving logic.

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) [6]. 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.

4. DEMONSTRATION STORYBOARD

During the live demonstration, we will showcase Baya at work and take our audience through the theoretical as well as the usage aspects of the tool, using a mix of slides and hands-on examples. In particular, we intend to organize the demonstration as follows:

1. ***Introduction***: A short intro to the goals and key concepts of Baya.
2. ***Example***: A simple example developed by us with the use of the interactive recommendations.
3. ***Non-assisted development by audience***: A similar modeling exercise for a member of the audience, however without the help of the interactive recommender.
4. ***Assisted development by audience***: The same modeling scenario as in 3, this time however with the help of the interactive recommender.
5. ***Patterns and discovery***: An explanation of the pattern types supported by Baya, along with the mining approach underlying the pattern knowledge base.
6. ***Architecture and internals***: Explanation of the internal architecture of Baya and of the recommendation and weaving algorithms working behind the scenes.
7. ***Conclusion***: Lessons learned and outline of future works and the evolution of Baya.

This process will allow us to introduce the audience to Baya and help us evaluate the efficacy and usability of the tool. We hope we will get valuable feedback from the audience, in order to further fine-tune Baya's UI and algorithms.

An introduction to and a screencast of Baya is available at `http://www.youtube.com/watch?v=RNRAsX1CXtE`.

5. STATUS AND LESSONS LEARNED

Baya was born in the context of the EU research project OMELETTE, in order to assist mashup development inside the project's own mashup editors. Soon, however, we recognized that the kind of knowledge discovery algorithms we were working on and the conceptual approach to pattern recommendation and weaving are generic enough to be applied in the context of many other modeling or mashup tools. As a proof of concept, we therefore developed Baya, an apparently simple, yet effective tool. The idea of composition knowledge as a service makes it unique among other assisted development approaches, and a-priori definition of pattern structures allows us to extract meaningful knowledge also from single mashup models.

Next, we will extend the mining algorithms to other composition paradigms and develop dedicated clients for different composition tools. The idea is to make Baya publicly available and to study how effectively pattern recommendation and weaving can help users to develop own mashups.

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

6. REFERENCES

[1] A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, editors. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, USA, 1993.

[2] A. De Angeli, A. Battocchi, S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati. End-User Requirements for Wisdom-Aware EUD. In *IS-EUD'11*, pages 245–250.

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

[4] M. Henneberger, B. Heinrich, F. Lautenbacher, and B. Bauer. Semantic-Based Planning of Process Models. In *Multikonferenz Wirtschaftsinformatik'08*, 2008.

[5] A. Ngu, M. Carlson, Q. Sheng, and H. young Paik. Semantic-based mashup of composite applications. *IEEE TSC*, 3(1):2 –15, 2010.

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

[7] J. Wong and J. I. Hong. Making mashups with marmite: towards end-user programming for the web. In *CHI'07*, pages 1435–1444.