Assisting Mashup Development in Browser-Based Modeling Tools

Soudip Roy Chowdhury
University of Trento, Italy
rchowdhury@disi.unitn.it

***Abstract*—Despite several efforts for simplifying the composition process, the learning process for using existing mashup editors remains rather difficult. Due to this difficult learning process, the development of mashup applications is only achievable by expert developers. In this paper, we describe how this barrier can be lowered by means of an assisted development approach that enables the reuse of existing composition knowledge. We further demonstrate and verify how with the help of our efficient interactive pattern recommendation and automated pattern weaving approaches, less-skilled users can successfully build mashup applications. The experimental evaluation of our prototype implementations and the result of user-studies validate the suitability of our assisted development approach for mashup design using browser based modeling tools.**

I. PROBLEM AND MOTIVATION

By now, mashup tools, such as Yahoo! Pipes (http://pipes.yahoo.com/pipes/), undoubtedly simplified some complex composition tasks by providing easy modeling constructs. Yet, despite these advances in simplifying technology, mashup development is still a *complex task* that can only be managed by skilled developers ([1], [2]). In order to better understand the problem that we address in this research, let's have a look at how a mashup is, for instance, composed in Yahoo! Pipes http://pipes.yahoo.com/pipes/, one of the most well-known and used mashup platforms as of today. Let us assume we want to develop a simple pipe that fetches a set of news feeds from Google News website, filters them according to a predefined condition (in our case, we want to search for news on products and services by a given vendor), and locates them on a Yahoo! Map based upon the geo-location associated with each news item. The pipe that implements the required feature is illustrated in Figure 1. As pointed out in Figure 1, for developing even such a simple application at every step a user needs to know which component/s to use, how to set the parameter values for the component, how to define the data mapping among components etc. Understanding this logic is neither trivial nor intuitive.

To have a more detailed understanding of the problem space i.e., end user development, we performed an initial user study [3] with few end users (10 university accountants), who have little technical expertise. The study revealed that less-skilled developers wished to be assisted with automatic/contextual help through out the development process

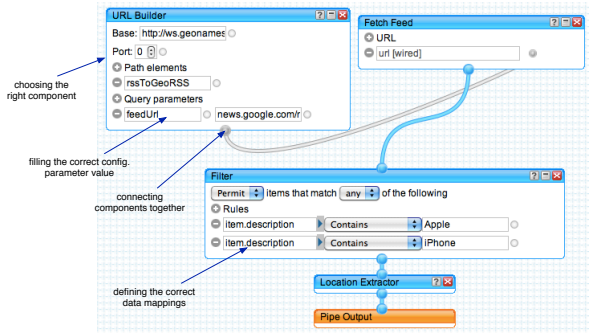

Figure 1. Screenshot of the Yahoo! Pipes mashup environment showing typical composition steps

in case they are developing an application. In order to meet the elicited requirements, in this paper we approach a few of the core research challenges of this vision, i.e., how to design an interactive development recommendation system that supports the reuse of existing composition knowledge for developing mashups in a step-by-step manner. That required us to find answers of the following ***research questions***:

- How to interactively reuse existing composition knowledge for developing new mashups.
- How to represent the composition knowledge that captures the typical modeling steps in mashup designs and that can be recommended to users as reusable knowledge.
- How to support interactive and contextual retrieval of composition knowledge at runtime.
- How to automate the application (weaving) of the modeling edits (addition or deletion of components, connectors etc.), captured in a composition knowledge, to the current composition context.
- How demonstrate and evaluate the usefulness of our interactive, contextual recommendation approach to its target users.
- How to assess the usability and the accuracy of our recommendation algorithms for the target user groups.

II. BACKGROUND AND RELATED WORK

Several related work aim to assist less skilled developers in the application development. Web macro approaches ([4], [5]) capture and reuse navigation patterns in end user

programming. However, the assistance in these approaches are provided only for navigating and configuring data in web pages. In our research, we support composition pattern reuse, by providing development assistance to end-users, in a more complex web mashup compositions scenario.

In the context of *web mashups*, several works aim to assist less skilled developers in the design of mashups. Syntactic approaches [6] suggest modeling constructs based on syntactic similarity (comparing output and input data types), while semantic approaches [7] annotate constructs to support suggestions based on the meaning of constructs. In programming by demonstration [8], the system aims to auto complete a process definition, starting from a set of user-selected model examples. Goal-oriented approaches [9] aim to assist the user by automatically deriving compositions that satisfy user-specified goals. Pattern-based development [10] aims at recommending connector patterns (so-called glue patterns) in response to user selected components (so-called mashlets) in order to autocomplete the partial mashup. The limitation of these approaches is that they partly overestimate the skills of less skilled developers, as they either still require advanced modeling skills (which users don't have), or they expect the user to specify complex rules for defining goals (which they are not able to), or they expect domain experts to specify and maintain complex semantic networks describing modeling constructs (which they don't do).

The *business process management* community more specifically focuses on patterns as a means for knowledge reuse. Among the related works for applying or weaving recommended patterns, the automated pattern application approach [11] uses structural properties of the current composition model to tell the user which pattern (simple merge, exclusive choice, parallel branch, and similar) among the workflow patterns, as are introduced by Van Der Aalst et al. [12], are applicable in the current modeling context. The structural properties of the workflow patterns are verified against the current process model structure to check their applicability. These control flow patterns are not able to capture domain knowledge of the underlying mashup applications, and hence are not contextual.

To address the limitations as identified in the existing related works, in our research, we aim at designing a more generic knowledge reuse approach i.e., in addition to the structural compatibility, we also consider the underlying mashup language (data flow based mashup) while capturing the knowledge to be reused for the assistance. In summary, in this research we design a more extensive assistance mechanism that not only interactively recommends reusable contextual composition knowledge, but also enables the *one-click* knowledge reuse approach by applying (weaving) the knowledge in the current composition context on behalf of a user.

III. APPROACH AND UNIQUENESS

The primary goal of the *interactive pattern recommender* is to assist users in designing mashups in a step-by-step manner. The knowledge we want to recommend is reusable composition patterns, i.e., model fragments that bear knowledge that may come from a variety of possible sources, such as usage examples or tutorials of the modeling tool (developer knowledge), best modeling practices (domain expert knowledge), or recurrent model fragments in a given repository of mashup models (community knowledge [14]). The structure and types of composition patterns that capture the typical design steps in a mashup environment are the core knowledge behind our step-by-step recommendation approach.

Composition pattern types. Composition patterns capture the typical modeling steps performed by a developer (e.g., filling input fields, connecting components etc. as shown in Figure 1) in a Pipes-like mashup tool. By analyzing the model of existing mashups [15], we specifically identified following five types of composition patterns.

Parameter value pattern represents a set of value assignments for the input parameters of a component. This pattern helps filling input parameters of a component that require explicit user input. *Connector pattern* represents a connector between a pair of components, along with the data mapping of the target component. This pattern helps connecting a newly placed component to the partial mashup model in the canvas. *Component co-occurrence pattern* captures pairs of components that frequently occur together. It comes with two associated components as well as with their connector, parameter values, and data mapping logic. *Component embedding pattern* captures the information about which component is embedded into which other component, both being preceded by another component. This pattern helps, for instance, modeling *loops*, a task that is usually not trivial to non-experts. *Multi-component pattern* represents model fragments that are composed of one or more of the patterns as described before. This list of pattern types is not exhaustive, but it contains the most representative steps in mashup design. To read more about the basic intuitions behind coming up with this set of composition patterns, one can refer to our earlier work [15].

Conceptual model for interactive pattern recommender. Figure 2 depicts the conceptual model of our interactive pattern recommendation approach. The "white part" in the right hand side of Figure 2 represents the condition or context under which a recommendation is triggered by the recommendation algorithm. For example, applying a modeling action *fill* to an object of type *parameter field* triggers a recommendation that consists of a parameter value pattern. The *Object-action-recommendation rule* determines the type/s of recommendations to be invoked based on the current partial composition state, modeling action and the

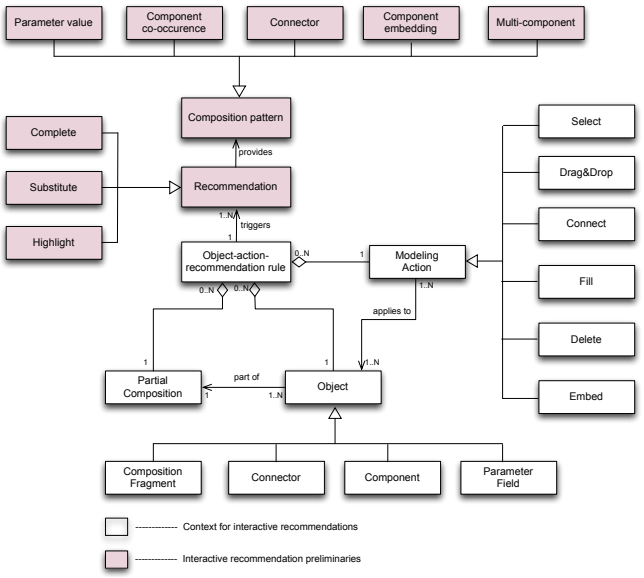

Figure 2. Conceptual model of interactive pattern recommendation approach

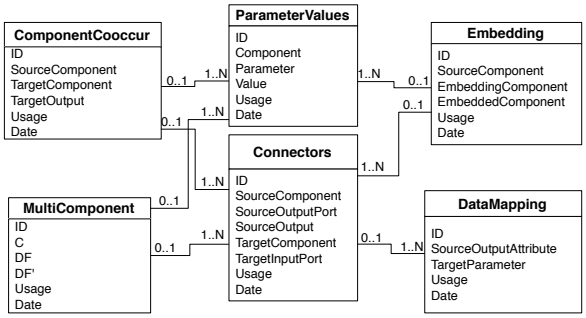

Figure 3. KB schema for composition patterns

object inside the current partial composition. The "grey part" of the Figure 2 represents the concepts related to the recommendation. A recommendation can be to complete a partial composition with a composition pattern or to substitute an existing component/s in the current composition with a similar one from the composition pattern, or the recommendation can highlight compatible components in the current modeling canvas. Composition pattern types are recommended to users in order to help them to proceed with the development steps.

A. Efficient, Interactive recommendation of composition patterns

The interactive pattern recommender consist of a pattern knowledge base (KB) that is structured to support fast retrieval of composition patterns at runtime, an exact and an approximate search algorithm that are designed for the efficient retrieval of graph-like pattern structures.

Pattern KB. The core of the interactive recommender is the KB that stores composition patterns, but decomposed into their constituent parts, so as to enable the incremental recommendation approach. Figure 3 illustrates the structure of the pattern KB. This schema enables fast retrieval of the composition patterns with a one-shot query over a single table. The KB is partly redundant (e.g., the structure of a complex pattern also contains components and connectors), but this is intentional. It allows us to avoid expensive database join operations or to defer them to the moment in which we really need to retrieve all details of a pattern. In order to retrieve, for example, the representation of a component co-occurrence pattern, it is therefore enough to query the *ComponentCooccur* entity for the *SourceComponent* and the *TargetComponent* attributes; weaving the pattern then into the modeling canvas requires querying $ComponentCooccur \bowtie DataMapping \bowtie ParameterValues$ for details.

Exact and approximate search of recommendations. Patterns in the KB are retrieved based upon an *object-action-recommendation* rule, which tells which patterns to be retrieved in the current composition context. The $object$ of an action is used as a query for searching of suitable patterns from the KB. We develop a similarity based algorithm to retrieve suitable patterns for the given composition context. For the fast execution of similarity based algorithm we preprocess the graph structure of the original patterns, and the query object to an intermediate representation. This preprocessing step saves us from the costly graph-traversal steps at runtime. For each retrieved pattern, we compute a rank based on the pattern description (e.g., containing $usage$ and $date$), the computed similarity, and the current partial mashup context. We then sort the patterns based upon their rank value and group the recommendations by their type, and filter out the top$k$ patterns for each recommendation type. Details of our pattern recommendation algorithm are described in [16].

B. Automated weaving of composition patterns

We don't limit our system functionality to only recommending patterns that can be applied to the current composition context, but we also help users in progressing their development task by applying a selected pattern in the current model on behalf of the user. We call this functionality automated *weaving* of composition patterns. Weaving a given composition pattern into a partial mashup model is not straightforward and requires a thorough analysis of the structure and context for both the pattern and the partial mashup, in order to understand how to connect the pattern to the constructs already present in the current composition model. We approached the problem of pattern weaving by first defining a *basic weaving strategy* that consists of steps (mashup operations) required for weaving a composition pattern type. For example weaving a parameter value pattern requires a mashup operation (assigning a parameter value to a parameter field of a component) to be performed. The Basic weaving strategy is agnostic about the current composition context. While applying a *basic*

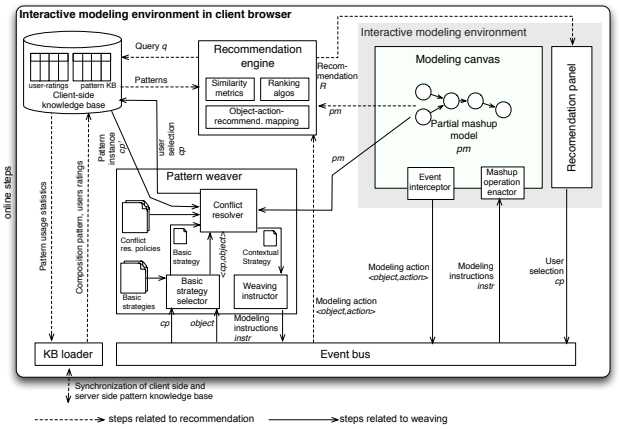

Figure 4. Functional architecture of the interactive pattern recommendation and automated weaving approach

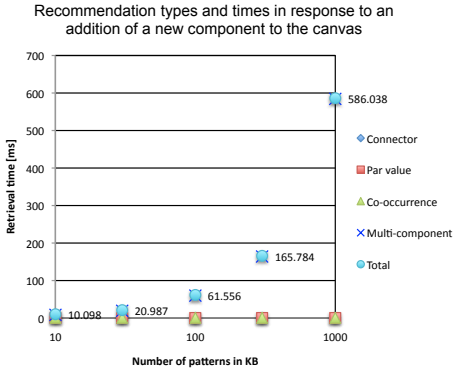

Figure 5. Performance evaluation for the client-side pattern recommender

weaving strategy in the current composition context, we may come across model conflict situations. For example, while applying the value to a parameter field of a component, we may come across a situation in which the selected parameter field for the component already contains a value. In such a case we resolve the modeling conflict with the help of a *conflict resolution policy*, which determines whether to override an existing modeling construct (in this case the existing parameter value) or to keep it. After resolving all possible modeling conflicts, finally, a set of *contextual weaving instructions* are generated which apply a pattern in the current composition context.

C. System design of our assisted mashup development platform

Figure 4 details the internals of our pattern recommender and weaving approach. As shown in the Figure 4 the server side knowledge base and the client side KB get synchronized at runtime and then on the recommendation algorithm queries the client side KB for the retrieval of the composition patterns. This helps to enable fast recommendation inside the client side mashup editors. The recommendation and weaving logic reside in the client side.

Baya: Assisted mashup development for Yahoo! Pipes. We developed a prototype tool named Baya [17] by implementing our novel interactive, contextual pattern recommendation and automated weaving algorithms. Baya is implemented as Mozilla Firefox (http://mozilla.com/firefox) extension for Yahoo! Pipes, adding an interactive recommendation panel to the modeling canvas. Baya targets both expert developers and beginners and aims to speed up the former and to enable the latter to design mashups. To view a working demonstration of Baya's assisted mashup development approach we refer our reader to the screencast at http://www.youtube.com/watch?v=AL0i4ONCUmQ.

Extension of Baya for assisting widget-based mashup. Continuing our work to make the Baya approach applicable to any browser-based mashup tool [18], in the context of an European union project OMELETTE (http://www.ict-omelette.eu/home), we implemented a pattern recommender widget (an extension of Baya). The pattern recommender widget supports widget based mashup design in Apache Rave (http://rave.apache.org/), an open-source mashup platform. To understand better how we extended Baya's pattern recommendation and weaving algorithm in the pattern recommender widget implementation, we refer the reader to [19].

IV. RESULTS AND CONTRIBUTIONS

A. Performance evaluation of our recommendation algorithm

. To validate the efficiency of our recommendation algorithm we performed suitable performance evaluations under different stress conditions. To perform such stress tests, we generate a realistic *test data set*, containing 130 parameter values, 650 component co-occurrence and, 3250 data-mapping, and 1000 multi-component patterns. he complexity of the patterns ranged from $3-9$ components per pattern, and we used queries with varying component set of size $1-7$. In the *worst-case scenario* (KB of 1000 multi-component patterns, approximate similarity matching of patterns), the recommendation engine retrieves relevant patterns within 608 milliseconds– everything entirely inside the client browser (cf. Figure 5). Details of the performance evaluation tests are described in our previous work [16].

B. Empirical evaluation for calculating the accuracy of pattern recommender algorithms

. To evaluate the accuracy of our recommendation algorithms, we followed the widely accepted recommendation-system evaluation metrics such as precision, recall metrics for our experiments. To calculate these metrics we had to calculate the *true positive*, *false positive* and *false negative* metrics with our test results. If the expected modules were found in the top-k list as returned by the recommendation engine then we marked the result as *true positive* (TP), if the

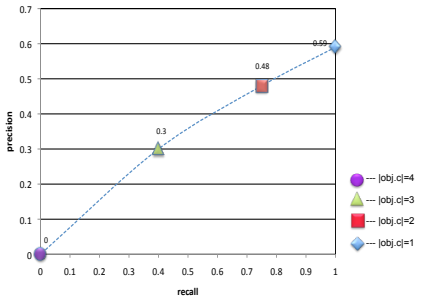

Figure 6. Accuracy measures for the top-10 recommendations by our algorithm

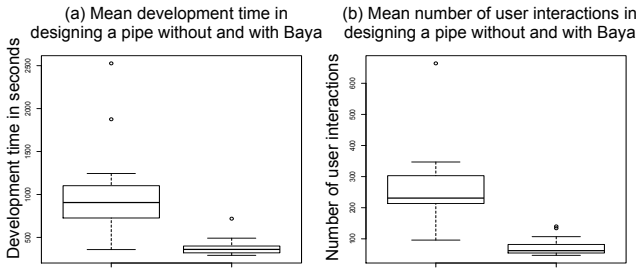

Figure 7. Results for Baya's usability validation test executed on the mechanical turk crowd-sourcing platform with 30 participants split into a control and a test group

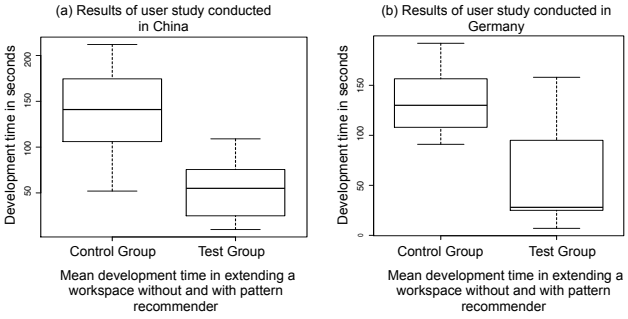

Figure 8. Results for usability validation test executed in China and in Germany to evaluate the benefit of pattern recommender tool

expected module was not found in the recommended top-k list then we marked the result as *false positive* (FP), and if the algorithm was not able to return any recommendation for a step then we marked the result as *false negative* (FN).

We decomposed a set of 100 pipes into subgraphs and then used them as query for our recommendation algorithm. In this selection process for 100 test pipes, we also made sure that we didn't consider those pipes that were included for the pattern KB creation. Based upon the retrieved results we calculated the precision, recall measures for our algorithms. The results of the evaluation of our algorithm, for recommending top-10 relevant patterns against different queries, are shown in Figure 6. As expected, the accuracy of the interactive recommendation algorithm decreases linearly with the increase of object size in the query.

C. User studies and validation

To asses the effectiveness of our assisted development approach, two user studies were performed during the course of this research. The first *user study* was performed by us, with the help of Amazon's Mechanical Turk crowd sourcing platform (https://www.mturk.com/mturk/welcome). We specifically tested whether Baya *speeds up* the development process, *reduces* the user interactions, and reduces thinking time during the development. We performed Welch's t-test for equal sample sizes and unequal variance on the evaluation results data (cf. Figure7) and that could statistically confirm the viability of our hypotheses. In response to the questionnaire that we asked users to fill in with feedbacks, the majority of the users (73% of control group and the 80% of test group) confirm the benefits of such an interactive recommendation utility in the end user development.

To cross-validate the result of our first user study, another follow-up study with target user groups was conducted with our prototype system in the context of an EU FP7 project (OMELETTE http://www.ict-omelette.eu/). The study was carried our by T-Systems http://www.t-systems-mms.com/ and Huwaei http://www.huawei.com/en/, who are industrial partners in OMELETTE project. The aim of that study was again to evaluate the usefulness of our assisted development approach in an end-user mashup design scenario. The study was conducted in China and in Germany and totally 44 participants took part in that study. The study result as shown in the Figure 8 re-confirms the validity of our earlier assumption that Baya *speeds up* the development (mean development time of 57 sec for the test group against 137 sec for the control group). 67% of all users agreed that this feature was important or even essential for a mashup environment. Few users also reported that Baya's assistance helped them to learn the usage of new widgets, which they were unaware of. These study results not only backs our claims about the usefulness of Baya in end-user development paradigm, but they also reveal the didactic value of our interactive step-by-step assistance mechanism. Details of the study and the test results is explained in [19].

V. CONCLUSION

Our interactive recommendation approach was initially designed to assists mashup design in Yahoo! Pipes mashup environment. Soon, however, we recognized that 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 and an extension of Baya to enable assisted development in browser based mashup tools like Yahoo! Pipes and Apache Rave. Next, we aim at extending our interactive recommendation approach in other composition domains such as business process modeling and web service composition etc.

REFERENCES

[1] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Comput. Surv.*, pp. 21:1–21:44, 2011.

[2] F. Casati, "How end-user development will save composition technologies from their continuing failures," in *IS-EUD'11*, 2011, pp. 4–6.

[3] 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*, 2011, pp. 245–250.

[4] C. Bogart, M. Burnett, A. Cypher, and C. Scaffidi, "End-user programming in the wild: A field study of coscripter scripts," in *VL/HCC 2008*, 2008, pp. 39 –46.

[5] A. Koesnandar, S. Elbaum, G. Rothermel, L. Hochstein, C. Scaffidi, and K. T. Stolee, "Using assertions to help end-user programmers create dependable web macros," in *SIGSOFT 2008*, ser. SIGSOFT '08/FSE-16, 2008, pp. 124–134.

[6] J. Wong and J. I. Hong, "Making mashups with marmite: towards end-user programming for the web," in *CHI'07*, pp. 1435–1444. [Online]. Available: http://doi.acm.org/10.1145/1240624.1240842

[7] A. Ngu, M. Carlson, Q. Sheng, and H. young Paik, "Semantic-based mashup of composite applications," *IEEE TSC*, vol. 3, no. 1, pp. 2 –15, 2010.

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

[9] A. V. Riabov, E. Boillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan, "Wishful search: interactive composition of data mashups," in *WWW'08*. ACM, 2008, pp. 775–784. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367602

[10] D. Deutch, O. Greenshpan, and T. Milo, "Navigating in complex mashed-up applications," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 320–329, Sep. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1920841.1920885

[11] T. Gschwind, J. Koehler, and J. Wong, "Applying patterns during business process modeling," in *BPM'08*. Springer, 2008, pp. 4–19. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85758-7_4

[12] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, pp. 5–51, July 2003.

[13] S. Mazanek and M. Minas, "Business Process Models as a Showcase for Syntax-Based Assistance in Diagram Editors," in *MODELS '09*, 2009, pp. 322–336.

[14] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati, "Wisdom-aware computing: On the interactive recommendation of composition knowledge," in *WESOA'10*. Springer, 2010, pp. 144–155.

[15] S. Roy Chowdhury, A. Birukou, F. Daniel, and F. Casati, "Composition patterns in data flow based mashups," in *Proceedings of EuroPLoP 2011*, 2011, pp. 27–28.

[16] S. Roy Chowdhury, F. Daniel, and F. Casati, "Efficient, Interactive Recommendation of Mashup Composition Knowledge," in *ICSOC'11*, 2011, pp. 374–388.

[17] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati, "Baya: assisted mashup development as a service," in *Proceedings of the 21st international conference companion on World Wide Web*, ser. WWW '12 Companion. New York, NY, USA: ACM, 2012, pp. 409–412. [Online]. Available: http://doi.acm.org/10.1145/2187980.2188061

[18] S. Roy Chowdhury, "Assisting end-user development in browser-based mashup tools," in *ICSE*, 2012, pp. 1625–1627.

[19] S. Roy Chowdhury, O. Chudnovskyy, M. Niederhausen, S. Pietschmann, P. Sharples, F. Daniel, and M. Gaedke, "Complementary assistance mechanisms for end user mashup composition," in *Proceedings of the 22nd international conference companion on World Wide Web*, ser. WWW '13 Companion. New York, NY, USA: ACM, 2012.