Complementary Assistance Mechanisms for End User Mashup Composition

Soudip Roy Chowdhury[1], Olexiy Chudnovskyy[2],Matthias Niederhausen[3], Stefan Pietschmann[3], Paul Sharples[4], Florian Daniel[1], and Martin Gaedke[2]
[1]{rchowdhury,daniel}@disi.unitn.it, [2]{olexiy.chudnovskyy,martin.gaedke}@cs.tu-chemnitz.de,[3]{matthias.niederhausen,stefan.pietschmann}@t-systems-mms.com,
[4]p.sharples@bolton.ac.uk

ABSTRACT

Despite several efforts for simplifying the composition process, learning efforts required for using existing mashup editors to develop mashups remain still high. In this paper, we describe how this barrier can be lowered by means of an *assisted development* approach that seamlessly integrates automatic composition and interactive pattern recommendation techniques into existing mashup platforms for supporting easy mashup development by end users. We showcase the use of such an assisted development environment in the context of an open-source mashup platform Apache Rave. Results of our user studies demonstrate the benefits of our approach for end user mashup development.

Categories and Subject Descriptors

H.m [**Information Systems**]: Miscellaneous; D.1 [**Software**]: Programming Techniques; D.2.6 [**Software**]: Software Engineering—*Programming Environments*

Keywords

assisted mashup development, automated compostion, interactive pattern recommendation, end user development, crisis mashup

1. INTRODUCTION

Mashup tools, such as Yahoo! Pipes (`http://pipes.yahoo.com/pipes/`) or Apache Rave (`http://rave.apache.org/`), offer simple modeling constructs, visual metaphors for programming that aim to help end users without programming skills in designing composition logics by re-using existing components/widgets. Despite the popularity of mashup tools in research, their applicability in end user development domain is still a far fetched goal. In practice, building a mashup remains a challenging task for non-programmers and even for less-skilled developers. This is because of the fact that less-skilled users simply lack knowledge about which components are available in a platform and how to use these components in their mashup design, how to configure such components to satisfy the design requirements etc. Above all for less skilled users or end users it is difficult to think of a program from design perspectives and, hence, these users find it difficult to define a consistent composition logic that integrates multiple components.

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

In order to aid such users in the design of mashups, related work has proposed two distinct approaches: ***goal-oriented solutions*** [4, 6, 9] aim to assist end users by automatically deriving compositions that satisfy user-specified goals; ***pattern-based development*** [3, 5] aims to recommend composition patterns in response to modeling actions, e.g., to auto-complete partial mashup models. Goal-oriented solutions strive for simple interactions with users to elicit the goal, i. e., intent of a composition without requiring users to actually model the mashup, while pattern-based approaches interactively assist them throughout the modeling process. None of the two individually may thus provide sufficient assistance to an user in developing mashups. Previous work therefore motivates the joint use of goal-based and pattern-based approaches for mashup composition. Authors in the paper [7] proposed such a hybrid approach, but they don't provide detailed insight into how to implement such a system in practice.

Our paper demonstrates how such a *hybrid* assistance solution can be designed in practice. Our system is developed on top of an existing open-source, widget-based mashup platform *Apache Rave* by combining the simplicity of a dialog-based automatic composer with the step-by-step assistance by an interactive pattern recommender [8]. In this demo we show how these two techniques *complement* each other well in assisting end users. We further show the usability of our system in an end user development scenario (emergency management). The user studies performed with our system provide evidences for the effectiveness of our hybrid assisted development approach in mashup development.

2. CHALLENGES AND CONTRIBUTIONS

To identify the challenges and requirements for end user assistance, let us introduce a real-world scenario, which also provides the context for our demonstration. In August 2002, a devastating flood caused by heavy rains hit the east of Germany and several other parts of Europe. Such a crisis situation demands for IT systems that support information seekers as well as humanitarian activity coordinators. The former want to quickly get an overview of the overall situation to understand the impact of an emergency incidents. This requires them to aggregate and filter information from different data sources (e. g., news articles, social streams, etc.). The latter need tools that help them to coordinate rescue tasks, calculate risks, and communicate with both their teams and the local authorities.

These situational requirements underline the necessity of a mashup platform that not only supports the quick devel-

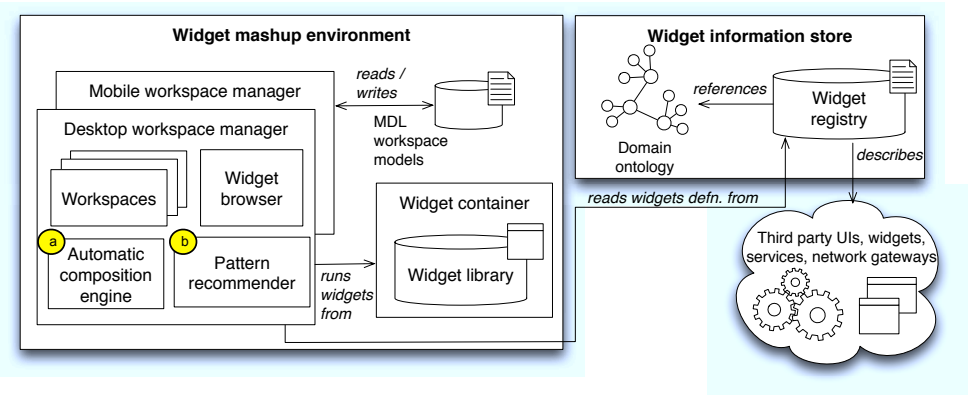

Figure 1: Functional architecture of the OMELETTE assisted development approach

opment of such applications but can also serve users with various technical backgrounds. However, to design such situational applications end user developers face several technical challenges, which are described below: First, while the intention of building a mashup is clear to end users, they do not necessarily know which kind of widgets they need. Second, if they do know, they still have no understanding of whether these widgets are available in the current mashup design platform and if they are not available then how to find an alternative widget having similar functionality. This process can be a cumbersome and error-prone for different reasons, but mostly due to insufficient or unsuitable widget descriptions (e.g., meta-data). Finally, end users typically lack knowledge about how to define the data and/or control flow in a composition, which are complex programming concepts for them to understand.

In this demonstration, we show how we address the design related issues for a hybrid development assistance approach. We further show how our approach of combining two complementary assistance mechanisms, namely an *automatic composition engine* and a *pattern recommender* can efficiently help end users in designing situational mashups.

The contributions of this paper are as follows:

- We describe our *goal-oriented dialog system* that enables the automatic composition of workspaces to less skilled users.
- We describe our *pattern-based recommendation system* that enables more skilled users to extend and/or refine mashup designs in a step-by-step fashion.
- We explain the *implementation* and integration of the respective algorithms into the open source mashup platform Apache Rave.
- We report on a *user study* conducted with 44 participants, demonstrating the benefit of combining both assistive techniques in one environment.

In the following, we present the realization of these contributions in the context of the EU FP7 project OMELETTE (`http://www.ict-omelette.eu/`). After that, we provide an overview of the demonstration workflow and close this paper with our findings from the user studies and ideas for future work.

3. OMELETTE APPROACH TO ASSISTED MASHUP COMPOSITION

Figure 1 gives an overview of the OMELETTE mashup architecture. Therein, widgets represent full-fledged application modules integrating both business logic and UI. The widget mashup environment allows end users to compose mashups (so-called workspaces) by placing one or more widgets on the composition canvas. Technical complexity of defining the data flow logic among components are abstracted from the user and handled automatically by the platform and the widget implementations. To facilitate assisted development, OMELETTE extends the mashup engine of Apache Rave and implements additional functional modules, e. g., inter-widget communication, widget information store, widget registry, etc.

Even though mashup creation sounds very simple, end users need support to cope with the challenges discussed in the previous section. Therefore, we introduce *two* tools to support users in implementing the desired applications without being overwhelmed by the technical details of the underlying platform and widget specifications. The first tool, the **Automatic Composition Engine** (ACE), targets novices who have no or very little experience in mashup development and need to be guided through the composition process. The second tool, the **Pattern Recommender** (PR), addresses those users who are already familiar with the composition environment, but need help in finding appropriate building blocks.

3.1 Automatic Composition Engine

The ACE allows end users to focus on the *goal* of the composition instead of the individual building blocks and their accurate "wiring". The ACE (cf. Figure 1.a) extends the mashup environment with a dialog-based interface that enables end users to specify their goals in an interactive manner. The dialog takes place in form of a question-answer game, during which the system elicits and refines user goals. Eventually, this process results in the automatic composition of a workspace derived from the identified goals.

The basis for this mechanism is an extensible knowledge base and a rule engine guiding the dialog with the user and ensuring the goal of a composition is clear at the end. The knowledge base comprises a domain ontology with facts about the application domains (e. g., project management or trip organization) and a set of functions defining the behavior of the dialog agent. The first function dedicated to

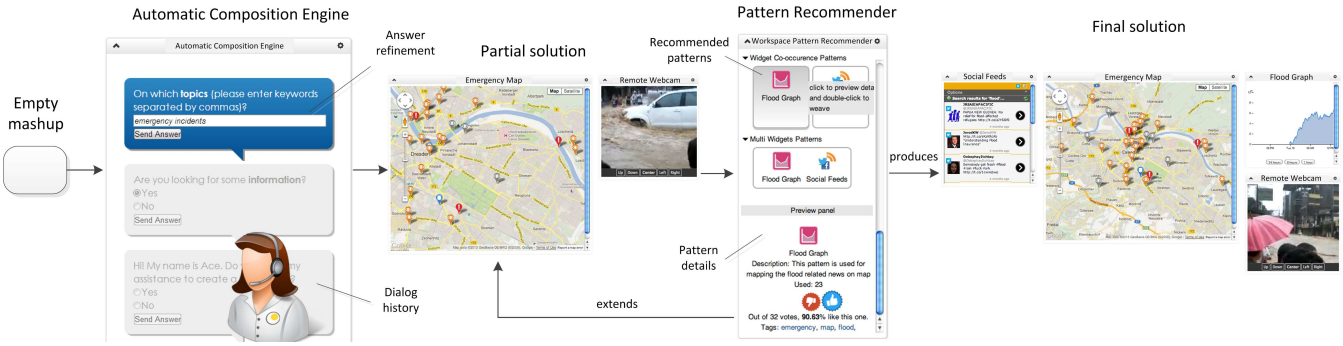

Figure 2: Hybrid recommendation system

question building is language-aware and responsible for the conversation flow with the user. The rule-based definition of this function enables dynamic and flexible conversations taking application context into account, e. g, the availability and capabilities of mashup components. Based on the user responses, the ***evidence collection*** function produces an overlay model of the domain ontology. The ***widget selection*** function defines a mapping between possible sets of collected evidences and a SPARQL search query to be issued to the widget registry, containing semantic descriptions of functional and non-functional aspects of the available widgets. The current implementation filters keyword annotations from the domain knowledge against widgets' textual attributes, such as title, description, tags, and category. To guarantee the best interoperability between the resulting widgets, they are filtered based on the information about their inter-widget-communication capabilities. Finally, the ***workspace configuration*** function derives a set of configuration parameters to be applied to widgets and workspace based on the evidences collected during the dialog.

In summary, the dialog agent helps users to select and configure widgets out of a large number of potentially incompatible components in an interactive and natural fashion. The more widgets with similar functionality and from different vendors are registered in the platform, the better results are achieved by ACE.

3.2 Pattern Recommender

The design goals behind the PR – as explained in our prior work [1] – can be summarized as follows: it supports less skilled users in *extending* mashups created by the ACE and it also provides autonomous assistance for skilled developers in building mashups from scratch. The PR helps users to reuse existing composition knowledge: the knowledge behind the PR's recommendations is harvested from existing workspace models. Extracted patterns are stored in a knowledge base (KB), which is structured to minimize database join operations for pattern retrieval at runtime. Currently, the PR supports two composition pattern types: ***widget co-occurence*** and ***multi-widget*** patterns. During composition, the PR reacts to user modeling *actions* (adding, deleting, or selecting a widget, etc.) on widgets (the *object* of an action) in the current workspace design. Upon each interaction, the *action* and its *object* are captured by the *recommendation engine* via suitable event listeners. With this information, the PR 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 filtered and ranked based on the current composition context (current workspace model) and rendered in the *recommendation panel*. The panel is the user interface of the PR and allows users to select a recommended pattern or browse its details. Upon selection of a pattern, the PR automatically weaves it into the current workspace model, resolving possible model conflicts.

4. IMPLEMENTATION

OMELETTE's assisted development approaches are implemented by extending two active Apache Software Foundation projects: Apache Rave and Apache Wookie (`http://wookie.apache.org/`). While Rave is used as the core mashup engine in OMELETTE, Wookie serves as a repository and runtime container for W3C widgets, accessible by both assistance mechanisms for retrieving widget information. Both assistance tools were realized as W3C widgets and are shown in action in Figure 2.

Apart from its client-side UI, **ACE**'s answer processing and evidence collection take place on the server side via a dedicated RESTful interface. The conversation and question building strategies are specified using production rules, which are executed on the server side by the JBoss Drools engine (`http://www.jboss.org/drools/`). To find widgets that best fit a user's needs, the server part of ACE uses a dedicated semantic registry (widget registry cf. Figure 1) based on the WebComposition/DataGridService [2].

The **PR** widget contains recommendation and weaving algorithms implemented in JavaScript. The client-side pattern KB runs on an in-browser SQLite (`http://www.sqlite.org/`) implementation. This eliminates performance overheads of client-server communication for retrieving recommendation patterns at runtime. Client and server-side pattern KBs are synchronized when loading the PR into a user's workspace. From then on, all queries triggered by the PR to retrieve patterns from the KB are directed to the client-side only. JavaScript event listeners capture the triggering events for pattern retrieval, i. e., DOM modifications (e.g., adding a widget, deleting a widget) of the workspace model.
Both the ACE and the PR rely on three new APIs introduced into Apache Rave to provide the necessary integration points between the mashup platform and the assistance tools. The first API allows the pattern mining algorithm of PR's server-side component to fetch all workspace models from the workspace repository. The second one is used by the recommendation algorithm implemented in PR to retrieve information about the set of widgets present in the

current workspace model. Finally, the third one is used by ACE and PR to populare workspaces with new widgets.

5. DEMONSTRATION STORYBOARD

In the demo session, we plan to showcase the two assistive techniques described above using the scenario introduced in the Section 2. We will show how workspaces can be created ad hoc by end users in the case of an emergency situation, and how such workspaces can be rapidly extended with the help of the assistance provided by the OMELETTE approach.

The demo will start from a blank workspace. By interacting with the dialog system of the ACE, the user will express his/her composition intentions, e. g., to gather information about emergency incidents and getting public web cam footage to gauge the impact on the ground zero. This will lead ACE to automatically populate the workspace with suitable widgets and their implicit wiring, without any further user interaction.

The second part of our demo will involve the extension of this workspace model with other widgets (e.g., telco widgets) that may be of interest to an emergency coordinator in the given scenario. We will show how the interactive recommender helps users to extend an existing workspace with a new set of widgets in a step-by-step manner.

Finally, we will explain the architecture behind our assisted platform and share the lessons learned during the implementation and user studies.

A screencast of our approach at work is available at `http://www.ict-omelette.eu/assisted-composition`.

6. EVALUATION AND FUTURE WORK

In order to evaluate our approach with a potential user group, we have conducted a user study including both the ACE and the PR. The study was conducted in China and Germany. In total, 44 participants attended the study, with only 11 of them had previous experiences of using widgets or configuring portal interfaces. Participants were equally distributed in test and control groups.

For evaluating the **ACE**, users were given the task to create a simple mashup that required them to build a workspace with at least three widgets. The required widgets were not explicitly named, but rather described by their functionality. The control group was assigned the same task, but had to use Apache Rave's widget store to search for suitable widgets. Interestingly, the study shows that participants using the ACE took more time on average than the control group (261 vs. 159 seconds). One decisive factor for this is the tool's learning curve: While all participants had the chance to try out the widget store before, the ACE was newly introduced. Even further, a few usability issues led some users far astray (hence the high variance), requiring them to start over multiple times.

The **PR** was evaluated in a similar fashion. Here, participants had to modify a given workspace with additional functionalities. The test group used the PR, whereas the control group again used the widget store to find right widgets. As the results show, the PR significantly reduced the overall task completion time (57 vs. 137 seconds). While the difference of the mean development time between the groups seems rather big, it must be noted that the PR has the distinct advantage of not requiring the user to leave his workspace in search of widgets.

For user satisfaction, there was a huge gap between user groups: while 64% of Chinese users said that they found our assistance mechanism to be useful and would use it again, only 36% of German users did so. However, 61% of all users agreed that this feature was important or even essential for a mashup environment.

Overall, the study shows that users with little experience in mashups prefer being guided through the processes of creating and extending their workspace. While no significant increase in efficiency could be verified for the process of creating a new mashup from scratch, results show that the recommendation of additional widgets based on an existing workspace provides significant benefits for users.

As a follow up to this user study, we are currently addressing usability issues of Apache Rave and of our assistance mechanisms. We are also working on a more natural goal elicitation system for the ACE to give users more freedom in creating new workspaces. The improvements on the ACE are going to be part of another evaluation, the results thereof we plan to present at the demo session. Future work includes improvement of composition patterns coverage in the PR's pattern KB as well as providing more explanations with each recommendation step to help users understand and decide whether to follow a recommendation or not.

Acknowledgment. This work was supported by the European Commission (project OMELETTE, contract 257635). Authors thank Vadim Chepegin from TIE Kinetix b.v. for his contributions to the design and implementation of the ACE.

7. REFERENCES

[1] S. R. Chowdhury, C. Rodríguez, F. Daniel, and F. Casati. Baya: assisted mashup development as a service. In *WWW'12 (Companion Volume)*, pages 409–412.

[2] O. Chudnovskyy and M. Gaedke. Development of Web 2.0 Applications using WebComposition / Data Grid Service. *Service Computation'10*, pages 55–61.

[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*.

[5] A. H. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H.-y. Paik. Semantic-based mashup of composite applications. *IEEE Trans. Serv. Comput.*, 3(1):2–15, Jan. 2010.

[6] S. Pietschmann, C. Radeck, and K. Meißner. Semantics-based discovery, selection and mediation for presentation-oriented mashups. In *MASHUPS'11*.

[7] C. Radeck, A. Lorz, G. Blichmann, and K. Meißner. Hybrid recommendation of composition knowledge for end user development of mashups. In *ICIW'12*, pages 30–33.

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

[9] V. Tietz, G. Blichmann, S. Pietschmann, and K. Meißner. Task-based recommendation of mashup components. In *ICWE'11*, pages 25–36.