Hybrid Recommendation of Composition Knowledge for End User Development of Mashups

Carsten Radeck, Alexander Lorz, Gregor Blichmann, Klaus Meißner
Technische Universität Dresden, Germany
{*Carsten.Radeck,Alexander.Lorz,Gregor.Blichmann,Klaus.Meissner*}*@tu-dresden.de*

***Abstract*—Due to the increasing number of available web APIs and services, mashups have become a prominent approach for building situational web applications. Still, current mashup tooling is not suitable for end users lacking detailed understanding of technical terms or development knowledge. A promising approach to ease mashup development is end user guidance utilizing recommendations on composition knowledge gained from experienced, similar users and semantic component annotations. In this paper, we present a novel hybrid recommendation approach suggesting suitable advice for the application development and adaptation to the end user. Composition knowledge in terms of common composition patterns is applied. Pattern instances are generalized by determining semantic exchangeablity of components to allow for context-sensitive recommendations. In addition, they are automatically integrated with the running application.**

***Keywords*-Mashup; End User Development; Hybrid Recommendation; Runtime Composition.**

I. Introduction

While the amount of available application programming interfaces (APIs) and third party resources in the Web is steadily increasing, the emerging *mashup* paradigm enables loosely coupled application components to be reused in several scenarios by simply combining them. Besides conventional mashup tools focussing on aggregation and processing of data from heterogeneous sources, there are proposals for the *universal composition* of mashups such as [1]. It includes the uniform composition and integration of distributed web resources. The latter are encapsulated in uniformly described components spanning all application layers including the user interface.

End User Development (EUD) aims at supporting the construction of individual applications addressing situational user needs. To this end, mashups are in principle a promising approach due to the reuse of building blocks and lower development efforts. We address domain experts, not necessarily with programming or technical knowledge, as end users. For those, early mashup platforms, e. g., Yahoo! Pipes, are unsuitable since they usually require an understanding of underlying technologies or programming skills. Furthermore, given the increasing amount of web resources and services, finding the right components or compositions is a challenging task, especially if there are only technical interface descriptions like WSDL. To overcome this and to foster EUD, recommendations for meaningful composition steps are of increasing importance [2], and gain momentum in the mashups domain as well. Our approach of universal composition by the given target group of end users at the application's runtime leads to several *challenges*:

C1 Besides the classic cold start problem, taking context-sensitivity and QoS into account is a main challenge in the domain of web services [3]. Given universal composition, this is of particular importance since we apply binding and instantiation of components in the runtime environment and support different target platforms. Compositions should also be restorable on different platforms later on. In such a setting, the components' suitability to the current context, especially device and software capabilities, has to be guaranteed.

C2 Since mashups claim to fulfil long-tail user needs, not only the "most popular" components and composition steps should be considered.

C3 The reason for and origin of recommendations should be presented to the end user [4]. Additionally, we argue that recommending functionality based on composition knowledge should be applied to hide technical details.

C4 The ad-hoc reconfiguration of the mashup at runtime necessitates the actual integration of recommended composition parts and should be highly automated. In addition, interface heterogeneity of semantically compatible components should be resolved automatically.

We propose a novel hybrid recommendation approach presenting suitable advice for composition steps to the end user. Therefore, we leverage composition knowledge in terms of patterns, either mined statically from existing applications or dynamically based on the lightweight semantic component annotations. By reasoning on composition patterns' functionality, determining semantic exchangeablity of components and integrating patterns in running applications, we aim to overcome the limitations of prevalent solutions.

The remaining paper is structured as follows. Related approaches for recommending components or composition knowledge and the conceptual foundation of our work are briefly discussed in Section II. Next, our hybrid recommendation approach is presented in Section III. Section IV, finally, summarizes the vision and outlines work in progress.

II. Related Work and Foundation

In the following, we discuss related approaches and define our conceptual basis in more detail.

A. Related Work

In general, recommender systems can be classified as collaborative filtering, content-based, and hybrid [5]. While collaborative filtering approaches utilize preferences of users similar to the active user, content-based ones recommend items similar to the active user's preferences.

There are a number of approaches using semantic similarity of component interfaces, e. g., [6]. Such matching approaches work well for the determination of alternative and possible follow-up components including their coupling. Additionally, rather "uncommon" solutions are recommendable and cold start can be avoided. On the other hand, semantic component descriptors are required, which have to be sufficiently expressive and, thus, complex to calculate whole compositions. However, the proposals mentioned above do not consider context information, collective knowledge, and the integration of recommendations.

Other approaches build up on previously defined compositions of other similar users. [7] utilizes collective knowledge by reuse of *mashlets* and *glue patterns* to suggest missing components and connections. [8] extends this approach by ranking compositions with regard to multiple QoS and context criteria. Based on semantic component descriptors, semantic matching and AI planning, *MashupAdvisor* calculates compositions probably fitting user goals (desired outputs of the mashup) [9]. Thereby, the statistical (co-)occurrence of input and output concepts are derived from existing compositions. In wisdom-aware computing [10], composition knowledge is provided as *advices* associated with *patterns* comprising the actual knowledge; *triggers* state the condition under which the *advices* are offered. The proposal relies on implicit semantics gathered by different mining techniques and statistical data analysis on existing compositions. This work covers the integration of recommendations in mashups. These approaches suffer from cold start, prefer popular solutions, and lack awareness of device or user context.

Other work focusses on the derivation of composition recommendations, which can, for instance, be achieved by mining frequent web service sequences [11] or matching the *composition context* (a composition fragment of connected services around a certain service) [12]. However, they suffer from cold start and context-aware substitution of services or their integration with the mashup are not addressed.

Hybrid recommender systems for web services are proposed in [13] and [3]. Although they overcome cold starts, both recommend single web services only and dynamic service integration is not considered.

In summary, none of the solutions fulfils the requirements identified in Section I. The next sections present the prerequisites and the overall concepts of our envisioned approach.

B. Universal Composition in CRUISe

Universal composition is in our case provided by CRUISe, which allows for platform and technology independent composition of arbitrary web resources and services [1]. Components encapsulating these resources are uniformly described using the Semantic Mashup Component Description Language (SMCDL) [14]. SMCDL covers non-functional properties and the public component interface including functional and data semantics of operations, events, and properties by means of ontology concepts. A declarative composition model describes the mashup application including the components, their state, event-based communication, and layout [1]. *Templates* as part of composition models allow for the context-aware selection of semantically compatible components suitable for the target runtime environment and user preferences. To this end, templates are equally characterized by a component interface, but additionally include non-functional requirements for ranking candidates.

III. Recommendation approach

In this section, we outline our novel concept for an EUD mashup platform utilizing hybrid recommendation. We extend the basic concepts of related proposals in several directions due to the challenges (*C1–C4*) sketched above.

The overall approach is shown in Figure 1 and has similarities to an adaptation loop, which, separated from the application, includes continuous monitoring of the context and the application, analyzing (i. e., evaluation of trigger conditions and calculating recommendations in terms of patterns), planing (i. e., deriving an action specification), and adapting the application by executing the plan (i. e., realization of the action specification). Details on the main steps and concepts are provided next.

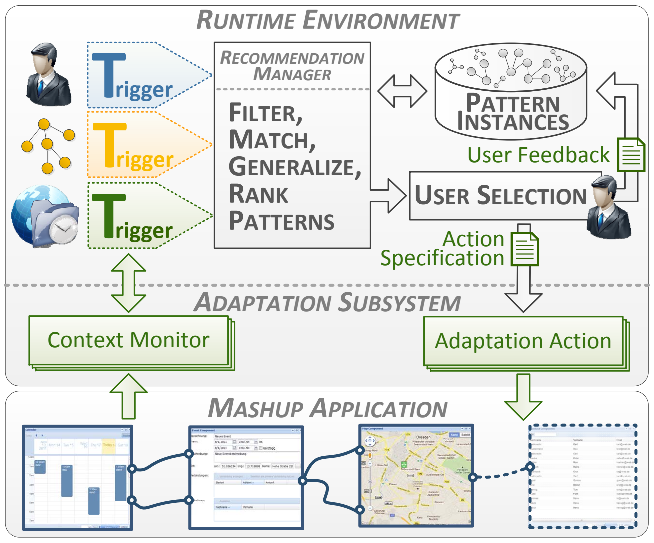

Figure 1. Overview of the recommendation approach

A. *Triggering the Recommendation Process*

We propose a unified trigger concept to cover the whole composition process starting from an empty application canvas to the interactive development at runtime. As an extension of [10], we distinguish *explicit* and *implicit* triggers. The former, colored blue in Figure 1, require the user's intention to ask for recommendations. The latter can be distinguished in *reactive* (colored green) and *proactive* (colored orange) and are generated by the platform based on context information. Reactive triggers respond to context changes, while proactive ones require the platform to continuously request and analyze additional information that might lead to recommendations. The following list provides examples:

- The user enters a brief description of his / her current task to be solved in a text field (*explicit*).
- Compared to compositions of similar users, a certain component is not part of the mashup (*proactive*).
- An event fired by a map component is not yet connected to any component (*reactive*).
- Entering a meeting room, a new service offered by a digital whiteboard becomes available (*reactive*).
- The underlying web service of a component has been unavailable for several requests (*reactive*).

Triggers model conditions setting them off, e. g., a user action and an affected component in case of an explicit trigger. Further, triggers are associated with a defined set of pattern classes, which are outlined in the next section.

Implicit, reactive triggers continuously receive notification of context changes. Those notifications are generated by the runtime environment's *adaptation subsystem* [15] that monitors context, like the user's position or device state. The *Recommendation Manager* interprets trigger events to suggest helpful composition fragments. As further data sources, it has access to the current composition model, a repository of composition models, and the user's context model.

B. *Modelling and Querying the Composition Knowledge*

Composition knowledge is represented by patterns describing common composition fragments, e. g., components and their connections. Pattern classes and pattern instances can by distinguished. In addition to the pattern classes *co-occurence*, *coupling*, *configuration*, and *complex* identified by [10], we introduce *occlusion*, *exchangeability*, and *layout*. While *exchangeable* components provide the same functionality, an *occluding* component offers additional functionality. *Layout* represents a typical arrangement of components.

Besides a characteristic composition fragment described by the composition model mentioned in Section II-B, each pattern has a rating and an origin to create trust and enable traceability. The origin describes whether the pattern was detected by semantic reasoning or from collective knowledge. A further attribute is the functionality, which can be derived from the components' semantic annotations. This allows for a user-appropriate visualization of the recommendations primarily showing *what* will happen and, secondarily, which composition fragments will be involved.

Pattern instances are decoupled from their mode of detection. Annotation-based semantic reasoning on exchangeable and connectible components takes place either statically or upon request. This way, the cold start problem is avoided and "niche requirements" can be met (*C2*). On the other hand, statistical analysis and data mining of existing compositions, for example, as proposed by [13] and [3], allow for utilization of (complex) collective composition knowledge.

Based on a trigger, a preselection of suitable patterns utilizing the mapping from trigger to pattern classes is conducted. Then, pattern instances are matched against the current composition and the conditions modelled by a trigger. To allow for context-sensitive recommendations (*C1*), we abstract pattern instances using *templates*.

Example: A mashup contains the component "Calendar", which is, as identified from collective knowledge, frequently connected with "Facebook Contacts" (*coupling*). The context model states that the user has no Facebook account, but is registered at Google+. A second component, "Google Contactor", is recognized as a semantic substitute for "Facebook Contacts" in relation to the coupling with "Calendar". Therefore, the coupling with "Google Contactor" is recommended.

Those equivalence classes of exchangeable components can be determined by semantic matching of component interfaces [14]. This way, we are able to generalize pattern instances by identifying the occurrence of abstract component classes, represented by templates, instead of concrete components. The selection of a concrete component for a template utilizes context information and non-functional annotations. Thus, the suitability of components for satisfying the user's preferences and their compatibility with a certain runtime platform and device are considered.

Collaborative filtering takes place to rank the patterns with respect to their popularity, ratings and relevance for the current user. Existing solutions can be leveraged for this task, e. g., clustering of users as proposed by [3].

Finally, the declarative *action specification* is derived. It represents the changes in the composition model and, thus, states all necessary steps to integrate a particular pattern instance with the current mashup.

C. *Presenting and Integrating the Composition Knowledge*

When visualizing suitable patterns for a trigger, the additional functionality provided to the user is emphasized (*C3*).

Example: The user requests the extension of the mashup containing the "Calendar". The functionality *Invite persons to your appointment* is offered. After the user agrees, a selection of appropriate pattern instances, containing amongst others the coupling with "Google Contactor", is shown.

One important requirement is the dynamic integration of the recommended composition knowledge with the mashup under development (*C4*). For this purpose, the defined action specification for the pattern instance selected by the user is the main input and interpreted by the runtime environment. Again, we utilize and extend the adaptation subsystem of the runtime environment since it provides the necessary means to realize adaptations at the component and the composition layer, cf. Figure 1. Amongst others, declarative *adaptation actions* for adding, exchanging, and reconfiguring components, adapting the layout, as well as creating communication channels, and registering publishers and subscribers are supported. This allows for a seamless integration of our concepts with the underlying infrastructure. The correct order of adaptation actions and their transaction-oriented execution are guaranteed by the adaptation subsystem.

Example: The action specification for the coupling pattern from the previous example comprises *adding "Google Contactor"*, *creating a communication channel between "Calendar" and "Google Contactor"*, and *registering the components as publisher or subscriber with this channel*.

The user's feedback on a pattern instance's suitability is collected, both explicitly by ratings and implicitly depending on whether the pattern instance has been applied or not.

IV. Conclusion and Future Work

In this paper, we have given a brief overview of our approach to recommend composition knowledge. The latter is derived from existing compositions of similar users (collaborative filtering) or from semantic component descriptions (content-based) and weaved it into running mashups. Based on a unified notion of recommendation triggers and composition patterns as well as an adaptation-enabled runtime, we provide continuous development support for end users during the usage of a mashup application. To allow for context-sensitive recommendations, patterns are generalized by semantically reasoned exchangeability of components.

The envisioned concepts are still early work in progress requiring further elaboration, e. g., detection mechanisms for implicit triggers as well as aggregation and processing of trigger events. We continuously work on prototypes within the CRUISe architecture to study practicability and feasibility in different application domains. One of the next steps is to incorporate and extend means for mediation resulting from our previous work [14]. Further, a user study is planned to evaluate the recommendation approach.

V. Acknowledgments

Funding for the EDYRA project is provided by the Free State of Saxony and the European Union within the European Social Funds program (ESF-080951805).

References

[1] S. Pietschmann, V. Tietz, J. Reimann, C. Liebing, M. Pohle, and K. Meißner, "A metamodel for context-aware component-based mashup applications," in *12th Intl. Conf. on Information Integration and Web-based Applications & Services (iiWAS 2010)*. ACM, Nov. 2010, pp. 413–420.

[2] A. Namoun, T. Nestler, and A. De Angeli, "Service composition for non-programmers: Prospects, problems, and design recommendations," in *8th European Conference on Web Services (ECOWS 2010)*. IEEE, Dec. 2010, pp. 123–130.

[3] L. Liu, F. Lecue, and N. Mehandjiev, "A hybrid approach to recommending semantic software services," in *Intl. Conf. on Web Services (ICWS 2011)*. IEEE, Jul. 2011, pp. 379–386.

[4] A. De Angeli, A. Battocchi, S. Roy Chowdhury, C. Rodriguez, F. Daniel, and F. Casati, "End-user requirements for wisdom-aware eud," in *End-User Development*, ser. LNCS. Springer, 2011, pp. 245–250.

[5] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[6] D. Bianchini, V. De Antonellis, and M. Melchiori, "A recommendation system for semantic mashup design," in *Workshop on Database and Expert Systems Applications (DEXA)*, 2010, pp. 159–163.

[7] O. Greenshpan, T. Milo, and N. Polyzotis, "Autocompletion for mashups," *Proc. of the VLDB Endowment*, vol. 2, no. 1, pp. 538–549, Aug. 2009.

[8] M. Picozzi, M. Rodolfi, C. Cappiello, and M. Matera, "Quality-based recommendations for mashup composition," in *Current Trends in Web Engineering*, ser. LNCS. Springer, Jul. 2010, pp. 360–371.

[9] H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin, "Mashup advisor: A recommendation tool for mashup development," in *International Conference on Web Services (ICWS 2008)*. IEEE, Sep. 2008, pp. 337–344.

[10] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati, "Wisdom-aware computing: On the interactive recommendation of composition knowledge," in *Service-Oriented Computing*, ser. LNCS. Springer, Dec. 2010, pp. 144–155.

[11] A. Maaradji, H. Hacid, R. Skraba, and A. Vakali, "Social web mashups full completion via frequent sequence mining," in *World Congress on Services (SERVICES 2011)*. IEEE, Jul. 2011, pp. 9–16.

[12] N. Chan, W. Gaaloul, and S. Tata, "Composition context matching for web service recommendation," in *Intl. Conf. on Services Computing (SCC 2011)*. IEEE, 2011, pp. 624–631.

[13] C. Zhao, C. Ma, J. Zhang, J. Zhang, L. Yi, and X. Mao, "Hyperservice: Linking and exploring services on the web," in *Intl. Conf. on Web Services (ICWS 2010)*. IEEE, Jul. 2010, pp. 17–24.

[14] S. Pietschmann, C. Radeck, and K. Meißner, "Semantics-based discovery, selection and mediation for presentation-oriented mashups," in *5th Intl. Workshop on Web APIs and Service Mashups (Mashups)*. ACM, Sep. 2011, pp. 1–8.

[15] S. Pietschmann, C. Radeck, and K. Meißner, "Facilitating context-awareness in composite mashup applications," in *3rd Intl. Conf. on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2011)*. XPS, Sep. 2011, pp. 1–8.