Enabling End Users to Build Situational Collaborative Mashups at Runtime

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

***Abstract*—Web based collaboration gains importance in everyday life. However, users have to switch between dedicated collaboration tools that are not interoperable and do not satisfy the long tail of user needs. To overcome this, users would have to develop customized applications by themselves. Caused by a lack of support for users without programming experiences, existing approaches lack support for end user development (EUD) or cause high cognitive load leading to frustration. Therefore, we utilize the mashup paradigm to empower end users in configuring their collaboration environment independently. To address arbitrary collaboration demands, we enable to synchronize collaborative and non collaborative components during an application's runtime. In addition, users are able to share mashups as well as components, entirely or only in part, with an arbitrary number of collaboration partners. To further allow for individual user preferences, amongst others, we facilitate a semantic component description to synchronize different implementations of functionally similar applications.**

Keywords-Collaborative End User Development; Synchronous Groupware; Mashup Composition

I. Introduction

Indicated by an increasing number of solutions even the technological requirements of collaborative applications can be met by web browsers [1]. In this course, the number of demands for individualized collaboration tools increases, too. Especially within unstructured collaboration, characterized by its informal nature and non-hierarchical organization, not all of the users' demands can be anticipated during the development of groupware [2]. Hence, end users have to become developers themselves [3]. Within this, two basic challenges exist: Enabling users with no programming experiences to build individual collaboration tools and allowing for simultaneous development and usage.

To cope with these challenges, mashups are a promising approach, but neglect collaborative scenarios so far. Universal composition [4] combines arbitrary web resources spanning all application layers, i. e., data, logic and UI. Therefore, end users can build desired applications by combining existing components via, e. g., drag and drop. Combining these benefits and supporting end users to build solutions for synchronous collaboration independently, we propose our vision for collaborative mashup EUD in this paper. Based on an extensive literature review and experience with previously projects, we identified three major challenges for this approach, to whose solution we will contribute:

C1 To lower the training effort in new usage scenarios, users should be able to collaborate using their preferred applications. Therefore, all components, regardless of their implementation or built-in support for collaboration, have to be synchronizable and connectable. We have to develop a mechanism for unified handling of non collaborative and collaborative components, even if the latter include more sophisticated means for, e. g., awareness, which is insufficiently addressed by universal composition approaches. To this end, at least a unified component description has to be developed.

C2 In order to allow each user to use his desired device or select components according to his preferences, we will have to synchronize differently implemented components with identical functionalities. In addition, novel mechanisms for awareness between collaboration partners are required, taking into consideration the existence of different implementations and individually granted sharing levels for different application parts.

C3 To support unstructured collaboration, fine-grained sharing of mashup composition parts across all application layers is necessary. An appropriate rights management, especially regarding visualization and interaction metaphors for non programmers, is not provided sufficiently yet, and one of the major keys for success within the solution to be developed.

Addressing these challenges, we propose a mashup environment where users without programming experiences are supported to fulfil their individual collaboration needs. The approach is part of the EDYRA project, which utilizes recommendations [5] to enable mashup EUD. To extend the current solution for collaborative scenarios, we strive for a uniform handling of collaborative and non collaborative components, synchronization of differently implemented components with equal functionalities as well as fine-grained sharing of arbitrary application parts.

Consider the following scenario we refer to throughout the paper. Bob plans a sight seeing trip. To this end, he builds a mashup including a map, a hotel search and a sight advisor component. To contact Alice for some inspiration, he adds a facebook messenger component. While the latter offers built-in collaboration, the other components are not originally intended for collaborative use. However, Bob can share all of them for a synchronous usage with Alice. Because she uses a smartphone, components will automatically be replaced with functionally identical pendants optimized for touch based devices. Bob restricts Alice's rights so that she can only view the selected location, but can not change it. To get additional information about the discussed sights, Alice inserts a Wikipedia component but keeps it private.

Before we present our envisioned approach in Section III, we give an overview of our foundations in Section II. Section IV briefly discusses related approaches. Finally, Section V summarizes the vision and outlines future work.

II. Conceptual Foundation

In the following, we give a brief overview of our conceptual basis, the CRUISe platform [4]. CRUISe provides universal composition through platform and technology independent combination of arbitrary web resources and services. Resources encapsulated as components are uniformly described using the Semantic Mashup Component Description Language (SMCDL) [6]. SMCDL covers the public component interface and non-functional properties, like price, author and input modalities. The interface consists of properties as well as parametrized operations and events. Optionally, ontology concepts can be annotated to clarify data semantics of parameters and properties as well as functional semantics of operations. In general, UI and service components are distinguished. Including all components, their state, event-based communication, and layout, a composition model declaratively describes mashup applications [4]. To enable context-aware selection of semantically compatible components according to, e. g., suitable target runtimes or user preferences, so called *Templates* are used [6]. To this end, templates are equally characterized by a component interface, but additionally include non-functional requirements for ranking candidates. With regard to the support for synchronous usage, CRUISe currently neither enables to synchronize components nor to individually share parts of the mashup with others.

III. Collaborative Mashup EUD

In this section, we present our environment for collaborative mashup creation by end users. The Mashup Runtime Environment (MRE) is shown in Figure 1. We take advantage of loosely coupled clients, server side components and access control using the server as proxy, through a centralized architecture. *Mashup Runtimes*, which will be reused, for instance, from CRUISe, act as execution container for mashups that context-sensitively retrieve components from the *Component Repository*. To enable execution on client and server, each environment has a Mashup Runtime. Thus, we execute private components only in clients' environments. Therefore, a collaborative mashup equals the union of all composition fragments from all clients and the server. To enable collaborative scenarios, Mashup Runtimes are extended by the following middle-ware functionalities.

Through the *Communication Manager* clients can exchange messages with the server. Each client and the server includes a *Coordination Manager*, which receives every message published at the corresponding part of the MRE first. Local messages are sent to the server, where the Coordination Manager orchestrates further processing steps. First, the *Concurrency Manager* ensures correct handling of concurrent messages by preserving their global order and drops redundant ones. Next, the *Access Manager* checks which of the MRE parts has the right to receive the current message with the help of a routing table, which includes users' sharing definitions. An *User Service* ensures trusted authentication and is tightly coupled with a *Context Service* that comprises a semantic user model. The *Awareness Manager* analyzes the current message and generates a command for all clients concerned to display, e. g., information about a new component. Therefore, the Awareness Manager is tightly couple with Access Manager, to ensure that only clients that are granted receive this information. The *Session Manager* handles necessary session data, like connected users or included components.

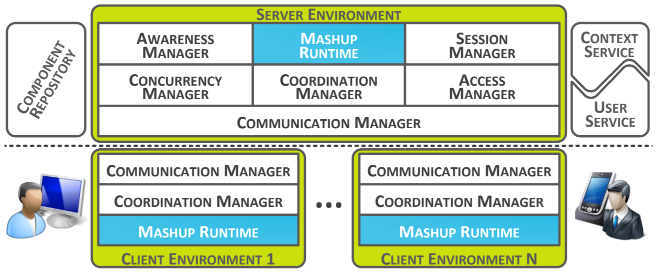

Figure 1. Architecture of the collaborative Mashup Runtime Environment

In the following, we briefly describe how we solve the identified three challenges *C1-C3* with the use of unified mashup synchronization, synchronization of heterogeneous composition fragments, and fine-grained application sharing.

A. Unified Synchronization for Mashup Components

In contrast to distinguish between collaborative and transparently adapted, non collaborative components, we strive for a hybrid approach, where users can uniformly use both kinds in one environment. Thus, regarding *C1*, users can choose components they are preferring or familiar with. Besides service or UI, components will be classified in collaborative and non collaborative. Furthermore, the former are differentiated by support for: communication, e. g. *chat*, coordination, e. g., *task list* or cooperation, e. g., *text editor*. Thereby, users can receive advice during composition extensions like, e. g., adding a communication component if only coordination and cooperation components exist. All components will be uniformly described through SMCDL.

Assisting component developers with no knowledge about collaboration support, the platform can synchronize non collaborative components by using the interface definition within the SMCDL comprising properties, operations and events. Therefore, after, e. g., a marker of a map is changed, the changed property triggers an event that is captured through the corresponding client side Coordination Manager. A further client side event delegation is suppressed. Instead, the event is send within a message to the server's Coordination Manager which then, after processing the message as described above, distributes it to all Mashup Runtimes that are allowed to receive. Now, the events are propagated within the event bus of each client Mashup Runtime in parallel. Although this approach allows only for limited awareness support and basic concurrency control, its advantage is genericity: no matter which implementation the component has or which functionality it offers, it can be synchronized.

B. Synchronizing Heterogeneous Composition Fragments

To meet *C2*, first, detecting functionally similar components with different implementations has to be possible. We utilize the findings from [6]. Thereby, semantically annotated component templates are used to find components with equal interfaces. Additionally, subsumption-based matching detects differences of operations' and events' data parameters.

To provide a more expressive functionality description, we use *Capabilities*, basically consisting of an activity, e. g., *search* and an entity on which this activity is performed, e. g., *hotels*. Both are represented as ontology concepts. Based on this, we try to detect functionally equal components with different interfaces. Once two components where detected as equal, mediation techniques [6] are employed to realize synchronization. As an example, we can detect that a Google Map and a Open Street Map fulfil the functionality *location selection* and synchronize them even if one is representing current marker positions as latitude and longitude and the other as pure address string.

Enabling end users to successfully collaborate via components offering the same functionality but requiring mediation rises additional challenges concerning rights management, concurrency control and awareness. As pointed out earlier, the latter is very important for supporting end users. To offer awareness across different components, we strive for awareness ports within a component's SMCDL. These ports can be semantically annotated to express the kind of awareness information they offer, e. g., *text highlighting within a text editor*. Known semantic based mediation techniques can now be used to exchange concrete awareness information in spite of different implementations.

C. Fine-grained Release of Composition Fragments

Besides synchronizing an entire application, we enable users to share only parts of it. Facing *C3*, we facilitate to provide these parts with different restrictions to an arbitrary number of users during application runtime. To specify access privileges, so called *sharing definitions* are used characterizing three aspects: *object*, *subject*, and *permission*.

The object indicates the application part to be shared. This can potentially be any kind of composition fragment like the whole application, a couple of components, a single component, a part of a single component, or at least nothing. The subject signals the persons with whom a user wants to share an object. In principle, sharing can be divided in *private*, *shared*, or *public*. While being private only the user himself can access the object, public allows access to all individuals in the session. Shared parts are accessible for single persons or groups. How people can interact with objects can be expressed by six permissions: no right to even see the object, right to only view it, right to interact, the right to reconfigure or edit the sharing object, e. g., *add a component or change wirings between components*, and right to again create sharing definitions for others.

A collaboration session starts by initially sharing application parts with others. Within a session, every user can create an arbitrary number of sharing definitions concerning objects he has inserted or he has been granted the permission. That means, not only the session initiator can define sharing definitions. After accessing the collaborative mashup, every participant can individually extend their part of the application and individually define access rights for others. Due to different definitions affecting, e. g., the same object, conflicts can occur. Therefore, a conflict detection within the *Access Manager* checks every definition. If a conflict is detected, a visual user feedback about the conflicting definitions, the conflict reasons and proposed resolutions will be displayed.

To ensure that the proposed permission can in principle be realized with every object, components can additionally offer different *modes*. Due to the use of black box components, especially when sharing only parts of them, such modes are needed. Take a map as an example. Besides its default mode, it offers a more restrictive mode where a user only can zoom or shift the map section, but is unable to set or delete markers. Provided modes are uniformly declared in the component's semantic descriptor.

IV. Related Work

Regarding EUD of composite collaborative web applications, a wide area of previous research was reviewed.

While early groupware frameworks, like Agilo [7], try to ease groupware development through reuse of application code, due to their programmatic approach, they lack in support for users with no programming experience. To lower the development complexity, many component or widget based approaches have been presented. While Cheaib et al. [2] only focus on web services, and do not regard UI components, approaches like DyCe [8] enables to work on shared components synchronously, but offers no fine-grained right restrictions or dynamic switching between private and shared. The ROLE project [9] supports learners building their own personal learning environment through combination of widgets. While users also are able to define widgets as shared or private, shared widgets are usable for every one. No adjustment during usage and no further fine-grained sharing is possible. In addition Graasp [10] distinguishes space members between owner, editor or viewer, but neither offers such fine-grained sharing definitions like we do nor a detailed concept for an end user appropriate right management. In addition, non of the solutions focus on the synchronization of heterogeneous components.

The idea to transparently synchronize encapsulated black box components by just accessing their outer interface was proposed earlier in the area of transparent adaption and collaboration transparency. Heinrich et al. [11] propose a generic approach for DOM-based rich internet applications. While they are able to synchronise arbitrary web applications, e. g., text or spread sheet editors, their solution demands identically application instances. Further, they provide no solution for end users to share only some application parts with others as well as to connect two application instances.

In the domain of mashups, only two solutions addressing synchronous collaboration were evident. Chudnovskyy et al. [12] explicitly focus on the combination of telco widgets and do not address the synchronization of arbitrary components as well as the definition of fine-grained sharings. The same holds true for Fox et al. [13], which only focus on secure collaboration and present no further concepts for sharing components through end users. Beyond this, actually no known approach for synchronously develop, share and extend mashup applications with others was proposed.

To sum up, none of the solutions provide sufficient support for the challenges identified in Section I.

V. Conclusion and Future Work

In this paper, we presented our vision for supporting end users developing individual, situational web applications for synchronous collaboration. Based on CRUISe, we address situational user needs through mashing up applications. The envisioned solution will provide three contributions. We enable to integrate and combine transparently synchronized non collaborative components as well as components explicitly supporting collaboration. In addition, to enable users with different contexts to collaborate, we synchronize differently implemented components with similar functionality using semantic component capabilities. Thirdly, we allow for fine-grained sharing of arbitrary composition fragments.

As the proposed vision is still in an early stage, further elaboration is necessary. We currently work on detailing the semantic description of collaborative components and a user study to evaluate our concepts for suitable visual interaction techniques. As a foundation we continuously expand our prototype to get user feed back.

VI. Acknowledgments

EDYRA is funded by the Free State of Saxony and the European Union within the ESF program (ESF-080951805).

References

[1] C. Gutwin, M. Lippold, and T. C. N. Graham, “Real-time groupware in the browser: Testing the performance of web-based networking,” in CSCW. ACM, 2011, pp. 167–176.

[2] N. Cheaib, S. Otmane, and M. Mallem, “Tailorable groupware design based on the 3c model,” in Int. J. Cooperative Inf. Syst., vol. 20, no. 4, 2011, pp. 405–439.

[3] T. Schümmer, “A pattern approach for end user centered groupware development,” Ph.D. dissertation, 2005.

[4] S. Pietschmann et al., “A metamodel for context-aware component-based mashup applications,” in 12th Intl. Conf. on Information Integration and Web-based Applications & Services (iiWAS). ACM, 2010, pp. 413–420.

[5] C. Radeck, A. Lorz, G. Blichmann, and K. Meißner, “Hybrid recommendation of composition knowledge for end user development of mashups,” in Proceedings of the 7th Intl. Conf. on Internet and Web Applications and Services (ICIW), 2012, pp. 30 – 33.

[6] 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, 2011, pp. 1–8.

[7] A. Guicking and T. Grasse, “A framework designed for synchronous groupware applications in heterogeneous environments,” in CRIWG, ser. Lecture Notes in Computer Science, vol. 4154. Springer, 2006, pp. 203–218.

[8] D. A. Tietze, “A framework for developing component based cooperative applications,” Ph.D. dissertation, TU Darmstadt, Sankt Augustin, 2001.

[9] S. Govaerts et al., “Towards responsive open learning environments: The role interoperability framework,” in EC-TEL, ser. Lecture Notes in Computer Science, vol. 6964. Springer, 2011, pp. 125–138.

[10] E. Bogdanov et al., “A Social Media Platform in Higher Education,” in Proceedings of the Global Engineering Education Conference (EDUCON), 2012, pp. 1–8.

[11] M. Heinrich, F. Lehmann, T. Springer, and M. Gaedke, “Exploiting single-user web applications for shared editing - a generic transformation approach,” in Proceedings of the 21st Intl. Conf. Companion on World Wide Web (WWW). ACM, 2012, pp. 1057–1066.

[12] O. Chudnovskyy et al., “End-User-Oriented Telco Mashups: The OMELETTE Approach,” in Proceedings of the 21st Intl. Conf. Companion on World Wide Web (WWW). ACM, 2012, pp. 235–238.

[13] R. Fox, J. Cooley, and M. Hauswirth, “Collaborative development of trusted mashups,” in Proceedings of the 12th Intl. Conf. on Information Integration and Web-based Applications & Services (iiWAS). ACM, 2010, pp. 255–262.