Triple-based Sharing of Context-Aware Composite Web Applications for Non-programmers

Gregor Blichmann, Carsten Radeck, Robert Starke and Klaus Meißner
Technische Universität Dresden, Dresden, Germany

Keywords: Synchronous Collaboration, Rights Management, Mashups, End User Development.

Abstract: Composite web applications are a promising way to support the long tail of user needs. While most mashup platforms only support single-user scenarios, CRUISE enables the reconfiguration of multi-user mashups during runtime. Thereby, synchronizing different parts of an application based on black-box components from different vendors causes special challenges for the rights management system. Cause we additionally focus on non-programmers as target group, an adequate user interface concept is needed. To overcome these challenges, we present a triple-based rights management concept as well as a corresponding user interface support. It supports fine-grained sharing of whole applications, single components or UI parts of components under configurable permissions. Thereby, users can select semantically compatible components during the collaborative session. The practicability of our concept is validated by a prototypically implementation as well as a user acceptance test.

1 INTRODUCTION

Within the last years, the number and acceptance of collaborative web applications rapidly rose in a variety of domains, like synchronous text and graphic editing or project and document management. Tools, like those of Google or Zoho, empower users very well in fulfilling predefined group tasks. But they do not support an easy adaption of the collaborative application to support individual user needs.

The trend of mass customization requires single-user applications to be adaptable to respect individual use cases which standard tools do not cover. Mashup approaches like (Radeck et al., 2013) or (Picozzi, 2013) address this long tail of user needs. End users without programming skills can build individual applications by composing heterogeneous components. Thereby, platforms facilitate recommendation techniques or visually hide applications' complexity.

However today's mashup application platforms offer no or only limited support to use and build such individual applications synchronously in a group of users. We argue that combining the two paradigms seems promising for users to use and adapt individually created applications for synchronous usage scenarios. Mashups following the universal composition paradigm (Pietschmann, 2009) are composed of self-contained black-box components which can exchange data. Parts of such a black-box-based composite web application (CWA) in form of single components can easily be shared. This ensures they also work correctly when being isolated from the rest of the application, what can not be guaranteed for traditional built rich internet applications (RIAs).

But, to enable end users to build CWA for synchronous usage, it first is necessary to define a rights management concept which is able to handle the specific requirements of a collaboration based on generic black-box components. In addition, end users with low or no programming knowledge require an adequate user interface (UI) metaphor within a graphical editor to define new or reconfigure existing sharing definitions. These definitions allow other users to access a certain part of the application with specific rights. It is necessary to provide awareness about all participants and their access rights to some parts of the application. Parts can be either single components or also parts of them in form of single UI elements whereby the rest of the component's UI is hidden. Likewise, users have to be aware about all parts of their application that were provided by others and again the rights that are granted for them. In detail, users have to understand which parts of the application they can edit or which they can only observe and which parts maybe blocked due to privacy needs.

Current platforms for mashup end user development (EUD) either do not support synchronous collaborative applications at all or lack in concepts for handling various access rights on different parts of a CWA as well as an adequate UI support.

The contributions of this paper are threefold:

1. Introduction of a triple-based rights management concept for sharing and collaboratively using CWA.
2. Presentation of a UI and interaction concept which allows users without deep technical knowledge to share parts of their application with others and support them to be aware about all parts they shared or received by others.
3. Evaluation by a user study based on a prototype as part of a distributed mashup runtime environment.

The remainder of this paper first discusses the paper's underlying research questions based on a reference scenario (Section 2). Section 3 describes the conceptual foundation of our work. After Section 4 introduces the triple-based rights management, Section 5 presents details of the UI support for end users. Afterwards, Section 6 highlights results of our evaluation. Section 8 summarizes the paper and outlines future work.

2 REFERENCE SCENARIO AND RESEARCH QUESTIONS

To illustrate underlying research questions, a reference scenario is presented:

The scientists Peter and Mary are planing to join a conference in Rome by the help of a collaboratively used CWA. It includes an event editor, a calender to store the conference date and duration, a map to display the conference's location as well as hotels, and a hotel search component. After passing the conference date, duration and location to the calender, the latter triggers the search for available rooms during the time of the conference sorted by the distance to the conference location. Hotels are additionally visualized in the map. To discuss the proposed hotels, Peter shares the hotel search and map with Mary, which accepts the invitation and joins. While Peter's application includes three components, Mary's only the hotel search and the map. Because Peter provides Alice the right to make changes, both can adjust the list of hotels, which permanently gets synchronized. Because Mary is not satisfied with the visual presentation of the map, she exchanges her local map with an alternative from another vendor. Due to their semantic interoperability, the different maps of Peter and Mary can still be synchronized. In addition, Mary adds a public transport component to her private part of the application to ensure the hotels' reachability without informing Peter.

Because Peter prefers four-star hotels, he blocks the corresponding input element on the UI for Alice. Next, he integrates a component which allows the digital request approval for business trips. After both filled in the form, Mary likes to ask some colleagues for verification. Peter allows Mary to re-share the component with others, but first defines his personal data to be visually hidden for others, like his correspondent bank account. Mary creates a new group and shares the component. After a member of the group gives Peter and Mary hints for cheaper hotels, they revoke the sharing with the group and change the hotel selection.

To support scenarios like these and solve the challenges mentioned in Section 1 several research questions have to be tackled: The basic question is, how to provide users without deep understanding of programming the possibility to share their individually created mashup application or parts of it with other users during runtime for synchronous collaborative usage. As target group, we focus on non-programmers which regularly use the web and are familiar with the usage of web based applications like Google Mail or Docs. First, there is a need for a rights management system which on the one hand is less complex than existing approaches like in the domain of operating systems and on the other hand faces the challenges which have to be tackled to support CWA. In detail, sharing of applications has to be uniform for whole applications, single components or even parts of them, even if the components used not offer collaborative features by themselves and have to be synchronized using their public interface description. In this context, respecting privacy issues in the sense of blocking parts of the UI during synchronous usage is also a challenging task when using black-box components. Thereby, the system has to guarantee that components with partly shared UIs are still usable. Additionally, the concept has to empower the user to understand which parts of his application he shared with other users under which access rights and which parts are granted to him by others. Also the synchronization of semantically compatible components form different vendors causes special needs for a rights management UI.

Before we present our solution in Section 4 and 5, we first introduce the foundations of our work in the next chapter.

3 FOUNDATIONS

Our concept adheres to the universal composition approach introduced by (Pietschmann, 2009). Thereby, CWA are built by combining arbitrary components from all application layers, including data, logic and UI. Necessary application information, like used components, communication connections, layout, screen flows or the adaptive behavior are described in a platform and technology independent composition model. A runtime environment interprets this model and context-sensitively selects suitable components from a component repository.

Components are realized as black-boxes and can encapsulate arbitrary web resources like data feeds, databases, web services, or UI widgets. All components have a public interface comprising operations, events and properties. Properties are uniquely typed key value pairs and represent a snipped of a component's inner state. State changes, indicated by events, can be used to invoke operations by passing a set of parameters. The component model is realized in XML by the Semantic Mashup Component Description Language (SMCDL). It declaratively describes the component's meta data, like information about author or price, the already mentioned interface and the component's bindings. The latter include references to all used frameworks and resources. Within the SMCDL, the component developer can describe the component's functional as well as data semantic by annotating concepts of third-party ontologies, e. g., describing the traveling domain.

Within Composition of Rich User Interface Services for Everybody (CRUISE) the model-driven development approach has been extended to support dynamic application reconfiguration during runtime by non-programmers. Among other things, the semantic annotations of the SMCDL were extended by *capabilities* (Radeck et al., 2013). They can either describe general functionalities of components or specific ones of single interface elements, like operations or properties. Therefore, basically a combination of *activities*, like ”Search” or ”Select” and *entities*, like ”Weather Information” or ”Location” is used. Both refer to concepts of third-party ontologies. Capabilities can either describe possible user interactions or system behavior. To represent cause and effect relations, it is possible to chain them. CSS selectors are used to establish a link between functional semantic describing capabilities and the UI by so called *view bindings*. To detail, e. g., the cause of a capability, *interaction operators*, like ”mouse click” or ”drag”, can be attached. The extended semantic component description, for example, is used to establish communication channels between components from different vendors which have non-equal but semantically compatible interfaces (Radeck et al., 2014).

Amongst others, the CRUISE platform is realized by a distributed client server runtime environment (CSR). It uses a centralized architecture to offer, for example, server-side execution of service components or server-side coordination of multiple clients in case of multi-user or multi-device scenarios. As presented in (Blichmann et al., 2013), the synchronous usage of CWA is supported by an architecture that blocks the initial local execution of state changes on each client to send them to the server first. The server afterwards informs all corresponding clients about the state change and ensures their parallel execution by a globally synchronized event queue. Following the advantages of the transparent synchronization paradigm, we consider that application components are usually built for single user scenarios and therefore, do neither provide any functionality for synchronize states, manage permissions of different users nor present awareness information. It is supposed, that all of these features are generically realized by the platform which uses the SMCDL of the components , e. g., to synchronize state changes by utilizing properties. Thereby, we only consider components based on HTML5 technologies (which are based on a DOM). Due to their vanishing relevance, plugin-based technologies, like Flash, are not in scope of our approach.

How users can specify which parts of their CWA are private and which should be synchronized with a certain group of collaborating participants is described in the next two sections.

4 TRIPLE-BASED RIGHTS MANAGEMENT

In this section, after defining basic terms and roles first, the proposed rights management concept for collaborative CWA is described in detail. Afterwards, its runtime support is presented.

4.1 Basic Terms and Roles

To understand the proposed rights management, first, it has to be clarified which parts of a CWA can be shared during runtime with participants of the collaborative session. As proposed in Section 1, a major advantage of using universal composition for synchronous collaboration compared to traditional CSCW tools is the possibility to share different functional parts of the application with different users.

We define such arbitrary parts of a CWA as *composition fragments* which can for example represent whole compositions, groups of connected components, single components or only parts of them.

Every shared composition fragment, is associated with exactly one *owner*. An owner is defined as the person who added the fragment to the application or initialized the application in case the fragment equals the whole composition. Owned fragments can be edited, removed or shared at any time with any other user. Thereby, users who are invited by an owner of an application for synchronous usage, can still extend this application and receives the ownership for the extended fragment. As for example presented in Section 2, an application owned and shared by Peter, potentially can have multiple composition fragments included, for example in form of single components like the public transport component, owned by Marry. Ownership cannot be restricted by, deleted by or handed over to somebody.

4.2 Triple-based Sharing Definitions

All composition fragments are by default marked as private for their owner and, thereby, can only be accessed and viewed by them no matter whether they were initially part of the composition or are added during runtime as long as they are not shared with somebody.

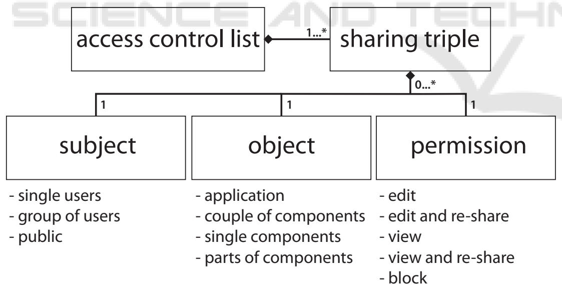

Figure 1: Basic triple scheme.

As presented in Figure 1, the rights management is based on an *access control list (ACL)*. To grant access rights for different collaborating partners, we utilize a set of sharing definitions based on *triples* which basically comprise a *subject*, an *object* and a *permission*. The subject determines the person **who** I want to share the part of my application with. It can be further distinguished in *public* sharing definitions for everybody and sharing definitions for *groups of users* or *single users*. The object specifies **what** the previously defined subject should be able to access. This includes all possible composition fragments, like the whole application or some arbitrary parts of a single component, like the location of an event or the list of selected hotels in the hotel component of the reference scenario (Section 2). Third, the permission is used to define **how** subjects can access the object. We differentiate between the right to only *view* the shared composition fragments as well as its state changes and the right to *edit* them, e. g., by interacting with the UI. The latter allows to interact with the UI of the object, for example by dragging the marker of a map. In general, the right to edit includes the right to view. The chosen terms were proposed by users during our evaluation (Section 6) and replaced the previous, more technical terms consume and contribute.

Within the process of sharing, an *inviter* is the user who shares an object with a subject. This can be either the owner of this composition fragment or a user with the permission to *re-share* the objects he received access to. The latter can be specified for each single sharing definition independently and allows users to share received objects with the permissions they have (see permissions at Figure 1). The *invitee* is a person which joins an application based on a previously received invitation of the inviter.

Due to the usage of CWA where components exchange data via a channel-based publish subscribe mechanism, sharing parts of an application has to take care of communication channels. Consider the following example. Peter uses and shares a component which presents the list of hotels based on the selected location of a map with Mary, but keeps the map as private. If Peter is changing the marker, his hotel component will be updated. To updated the component of Mary, the platform has to copy the event of the map and additionally invoke the corresponding operation ot Mary's hotel component, even if Mary does not have a map component at here client. Consequently, users of our target group do not have to explicitly share communication channels as part of the sharing definition, because they probably do not know what they do and which data is transfered. In fact, as motivated by the example, we assume that all events on incoming or outgoing channels of a shared component will be replicated to all users who received access to the component no matter whether the channel is connected to private or shared components. To add a channel which has a shared component as subscriber or publisher, the user has to have at least the right to edit or has to be owner of the component.

4.3 Process of Sharing

Within the CSR triples are realized by synchronized ACLs on client and server. The client-side ACL only stores the triples which are related to this client. These include all triples where the corresponding user of the client is either the inviter or one of the invitees. Thereby, the client-side ACL is used for the following functionalities:

- view, edit or reconfigure parts of the application,
- determine the composition fragments the user is allowed to share,
- present some awareness information about the current access right configuration.

The server-side ACL represents triples from all collaboration partners. This is needed to fulfill the following two functionalities:

- Routing the upcoming messages from each client to all clients that are allowed to receive the message, e. g., containing information about the state change of a component.
- Provide a mapping between a sender's and a receivers component interface in case of synchronization of differently implemented components.

To enable users to create, reconfigure and delete triple-based sharing definitions during runtime, the platform facilitates the following invitation process. If a user wants to share a composition fragment, he is able to open a dialog box (see Section 5 for details) and to define a new or adjust an existing triple. After the user opens the corresponding UI dialog, the system checks his current permissions via the client-side ACL. All fragments which can not be shared due to missing access rights are hidden. After its valid creation, the temporal sharing triple is send to the server as part of an *invitation* and marked on the client-side as to be approved. The server once again checks the user's permission to share the proposed triple. If the invitation is valid, the server requests semantically equivalent components from the component repository for all components part of the object separately for each subject to address individual user requirements. Thereby it facilitates the recommendation sub system of the CRUISE platform (Radeck et al., 2012) and attaches the list of alternative usable components to the invitation. Details according to the calculation of alternative components are not in scope of this paper. Next, all clients which where part of the invitation's subject are informed. As soon as the invitation arrives at a client-side runtime, a pop-up dialog appears. Therein, the invitee can decide to either reject or accept the invitation. In case of rejection, an invitation response is send back to the server, informs the inviter, and causes the deletion of the temporally created sharing triple. In case a user accepts the invitation, additionally he can choose to either use the component(s) initially shared by the inviter or to use at least one of the alternatives presented, like the Bing Map in the reference scenario (Section 2).

If the original component was selected, the invitation response which is send to the server and forwarded to the inviter leads to a persisting of the sharing triple at the client of the inviter and invitee as well as at the sever-side ACL. After finishing the invitation process, the generated sharing triples are used for example to route upcoming state changes of the application to all clients which are allowed to receive them. If the inviter wants to stop the sharing, he either can delete the triple using the sharing dialog from the beginning, or remove the corresponding composition fragment from his application. In this case, the composition fragment is deleted by all invitees too, including the related triples of the ACL. If an invitee decides to remove a certain composition fragment, it only will be deleted on his client. All other clients still can use it. If the invitee stops the sharing by deleting the corresponding triple, the inviter gets asked whether the other user is still allowed to use a local, not synchronized copy of the composition fragment.

To support users with no programming skills in sharing arbitrary composition fragments during runtime, extended user guidance and UI support is needed and is discussed in detail in the next section.

5 UI-SUPPORT FOR RIGHTS MANAGEMENT BY NON-PROGRAMMERS

To support users of our target group during the sharing process proposed in the last section, an adequate UI is needed. Therefore, it adheres to the triple metaphor, which promises the following advantages: First, it eases the creation of new sharing definitions while considering the challenges discussed in Section 2. Second, it allows simple understanding of defined permissions. Thereby, the UI concept follows a closed world assumption, i. e. all composition fragments which are not explicitly shared by a sharing definition are private.

5.1 Triple-based Rights Overview

Figure 2 presents the triple-based overview panel in screen **A**, which serves for managing sharing definitions. It is a separate window that overlays the application's UI. The panel basically consists of six parts. ① presents statistical information like the number of triples currently shared, the number of collaborative users and groups that currently exist as well as infor-

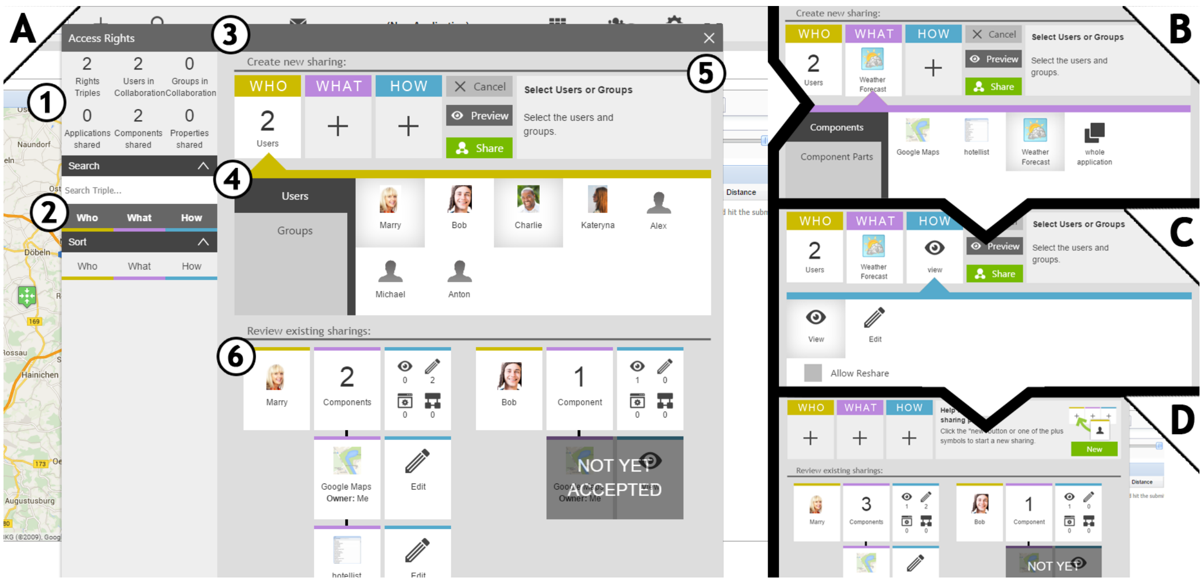

Figure 2: Prototypically implemented UI for sharing CWAs based on triples.

mation about open invitations. The filter and search panel ② enables to search, sort or filter the triples of previously assigned rights displayed in ⑥ by subject, object or permission. ③ allows users to create new sharing definitions by visually composing a new triple. Therefore, the user is guided by a message panel ⑤ on the right side which includes a stepwise instruction how to create a new triple. Already defined triples are displayed in a grid-based overview at the bottom ⑥ and are clustered by subjects. The latter was one of the features requested by the user study (Section 6). Thereby, as can be seen by the map and hotel component of Marry, the top-most triple summarizes the number of components and rights assigned. Initially, this triple is collapsed. Sharings which are not accepted yet, are greyed out. The size and amount of the triples displayed depends on the size of the device's display the user is currently using. The presented triples include both, the ones where the current user is the owner and the ones where the user is one of the invitees. For each object, information about the owner is attached.

To present the interaction steps necessary to create a new sharing, Figure 2 extends the reference scenario by a weather forecast component. Let us assume, Peter wants to share this component with Marry and Charlie. After clicking on the plus symbol of the **who** area, a panel for selecting suitable collaboration partners opens ④. The user can select either a number of single users currently registered at the platform or individually defined groups (see screen **A**). As indicated in screen **B**, by clicking on the plus symbol below **what**, a similar panel allows to select all components the user is allowed to share. Next, within screen **C**, the user can either select edit or view as permission and additionally can activate the check box to allow for re-share. Before the user clicks on the share button, he can test his sharing configuration. Thereby, a separated window is opened showing an instantiated copy of the selected components with respect to the selected permissions. Finally, indicated at screen **D**, after Peter clicked on the share button, area ③ will be reseted to its initial state and ⑥ includes the new triples marked as to be approved.

By hovering the mouse over an existing triple, the platform allows to delete or reconfigure it. For reconfiguration, the triple is loaded once again to ③.

As soon as an invitee accepts an invitation, the triple is displayed without the grey highlighting. If he rejects, the triple is removed. In both cases a short notification is presented to the user. If the invitee selects an alternative component, again a notification is presented. In addition, the triple visualizes this by a small icon at the area of the object. If a user hovers the mouse over this icon, the alternative component is displayed as object.

5.2 Awareness and Live View Support

As a result of our user study (Section 6), users recommended to integrate an easy to understand access right representation on the application's live view.

As indicated by Figure 3, the live view support is realized by specific click-able icons at the top right corner of each component. Thereby, three functionalities are integrated:

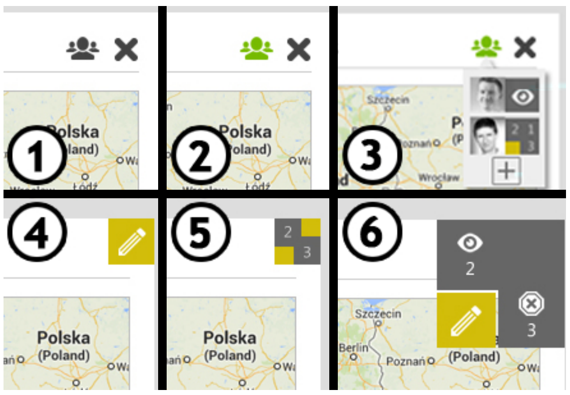
Figure 3: Live view support for inviters and invitees.

1. A short cut to share the corresponding component.
2. An indicator whether the component was already shared in general and some details of the specific collaboration partners and their access rights.
3. A representation that informs which components or their parts are shared with the current user under certain permissions.

To ease the overview of all components that can be shared by the user, the icon presented in ① is used. If the user already shared a component, the color changes to green ②. On hovering the icon with the mouse, a small dialog shows all users and their corresponding rights ③. By clicking the plus symbol the user can open the share menu presented in Section 5.1. The same functionality is achieved by directly clicking on the icon. In both cases, the triple-based overview starts a new triple creation process and preselects the component as object.

To indicate components that were owned by others, the granted access right is represented at the top right of the component, like for example the presented icon for the edit right in ④. If a component for example additionally includes parts which are blocked or parts which are only visible by read-only mode, a mixed representation is used ⑤, which can be enlarged by hovering the mouse over this icon ⑥. If users again hover the indicators for, e. g., blocked elements, the platform highlights the corresponding areas of the component's UI. Therefore, the CSS selectors of the view bindings are used which were referred by the capabilities that are part of the corresponding object of the triple. If the whole application was shared, each component of the application includes a corresponding icon.

Presenting further awareness information about, e. g., changed access rights are realized by a widget-based configurable awareness subsystem (Blichmann et al., 2015) and therefore are not part of this paper.

6 EVALUATION

The proposed rights management system as well es the corresponding UI support for CWAs were evaluated by a reference implementation within the existing CRUISE runtime environment (Section 6.1) and a user acceptance test of this reference implementation (Section 6.2)

6.1 Prototypical Implementation

The prototype is implemented as extension of the CSR. The server-side coordination layer uses Enterprise Java Beans (EJB) and is implemented as a singleton for all clients. The client-side is realized by pure HTML5, CSS and JavaScript (JS) technologies. Additionally, we use Ext JS[1] and JQuery[2] to fasten development. In order to provide a channel-based web socket communication between client and server, we use the Apache Apollo framework[3].

As proposed earlier, the ACL was implemented on client-side using JSON and on the server-side using plain Java. Updates between server and client are exchanged by dedicated commands and events which uses the communication channels of Apache Apollo. The client-side ACL clusters triples by users to allow an easy look up for the triple-based overview. Further, to detect owners of components, beside each sharing triple, the platform stores an additional triple with a permission representing the ownership. Each triple is associated with a state, to indicate, e. g., already accepted invitations. On server-side, a mapping for semantically compatible components is necessary. Thereby, the corresponding triples will be marked as alternatives. If a state change occurs at one client, the server can easily determine the components of the other participants that have to be informed. Potentially necessary data transformations are realized by the existing mediation infrastructure individually for each client.

6.2 User Acceptance

We used the *Thinking Aloud* methodology to conduct a user study which achieves two main goals: First, test the general acceptance of the triple metaphor for sharing applications and their parts. Second, analyze the usability of the proposed UI concept. Therefore, paper mockups were used to simulate the latter.

The user study was conducted with the help of ten male and four female participants including ages from

[1]https://www.sencha.com/products/extjs
[2]https://jquery.com/
[3]https://activemq.apache.org/apollo/

22 to 47 (average of 28.5). Six participants had no programming skills, five considered themselves as beginner and three were average-level programmers.

The process was three-staged: First, users get introduced by a video as well as a basic scenario to understand what mashups are and how they potentially can ease collaborative work. Second, they where encouraged to solve six tasks with increasing complexity by using the paper prototype. The tasks were created with the help of the reference scenario in Section 2. The moderator noted all comments and thoughts as well as updated the prototype based on the desired interactions of the participants. To get a comparable and standardized result, the participants were asked to fill in the System Usability Scale (SUS) questionnaire.

The overall results were very positive. Basic UI structures and metaphors were understood correctly without further explanations by nearly all participants (12/14). Three explicitly mentioned the colored separation as helpful. We found, that users preferred to use a share icon placed directly on top of the components instead using the icon in the main menu bar. This finding caused the extended live view support presented in Section 5.2. The used icons where understood by all users quite well. 12 of 14 users were able to select multiple triple elements without help. Thereby, we found out that users, probably due to their daily usage of social networks, often thought in a group-oriented way. This underlines the necessity of groups as subject, which we provide. The component icon visualizing whether a component was shared or not was instantly understood by 7 of 14 participants. All users understood the messaging symbol and menu to read and react on their invitation, the similarity to established messenger programs like Facebook was considered very positive. But, 9 of 14 users criticized the missing drag and drop support. The initially provided dedicated dialog for creating new triples was not understood by 9 of 14 and therefore skipped in the final result and replaced by the presented inline editing functionality in the triple overview panel. Icons were used inconsistently, confusing participants, and were therefore replaced by unique ones. Surprisingly, the possibility to use an existing triple as template for a new one was not considered as intuitive by the majority and removed from the final concept.

The average SUS score equals 77 with 62.5 as lowest and 98 as highest single user rating, which we consider as a promising result. The comments of the users showed that the general approach was understood and accepted very well. Suggested features or misunderstood elements were re-worked and are already part of the presented concepts. Further limitations are discussed in the next section.

6.3 Discussion

As already mentioned, some elements of the concept are influenced by the results of the user study, thus some concepts, like the final selection of icons for the live view, have to be validated in a second study. However, the majority of features have successfully been proven to be acceptable. Beside a good user acceptance, further challenges for a successful sharing support have to be discussed. First of all, to enable users, e. g., the blocking of single parts of the UI, quite rich component descriptions are needed. Component developers have to annotate semantic concepts in form of capabilities and view bindings to ensure that the platform can offer all functionalities presented above. We argue that the additional effort a component developer has to invest is quite less in comparison to the benefit users can achieve by using the proposed functionalities. In addition, we support component developers by a developer guide which includes best practices for, e. g., component annotation. The components itself have to use technologies which rely on the Document Object Model (DOM). From our perspective, plugin-based technologies like Flash can be omitted due to their decreasing dissemination. The technology stack of HTML5 and CSS3 offers an increasing number of functionalities and Application Programming Interfaces (APIs).

Concurrency control is inevitable for collaborative systems, but a detailed concept is very challenging in context of black-box components and therefore not part of this paper. Due to the usage of black-box components, we can not prevent that component changes already are finished locally before they can be processed and synchronized with all other participants. In case of conflicts, one of the conflicting state have to be rolled back. This disturbs usability and confuses the users. In addition, it is still challenging to give a quick access right overview on the live view while allowing specific restrictions on single UI elements.

7 RELATED WORK

The platform for distributed interactive workspaces (DIWs) (Ardito et al., 2014) allows users to synchronously use and edit component-based applications for joined learning. Thereby they can annotate, live edit or freeze parts of the application and declare theses changes as only private, public or group-width visible. In contrast to our solution, no user guidance during creating or reviewing access rights are given. In addition, no concept to block or share parts of a component's UI to respect privacy needs is presented.

Collaborative sessions where participants use different components are not considered.

Tschudnowsky et al. (Tschudnowsky et al., 2014) suggested an abstract architecture for mashup applications that can be used and reconfigurated synchronously by multiple users. But, the approach does not present any detailed concepts for rights management as well as user guidance at all.

MultiMasher includes a visual tool for creating multi-device mashups by marking parts of a website's UI and afterwards sharing them (Husmann et al., 2013). Due to the focus on co-located scenarios, no explicit awareness support about created right assignments or support for different components exist. It is also not possible to define single UI parts as private. Therefore, different views for the same resource can not be created. The component selection as well es interactions between components are not discussed.

The personal learning environment (PLE) Graasp (Bogdanov, 2013) enables the creation and sharing of resources and widgets with participants by grouping into spaces. But, privacy settings are only maintained at space level and can not be configured more fine-grained. Additionally, the rights management is based on a dedicated set of roles which may ease the right assignment in collaborative learning scenarios, but can not be used in generic application platforms like the one we propose. Another PLE, CURE, facilitates a room key metaphor to restrict access rights (Schümmer et al., 2005). CURE allows to share resources during runtime by end users similarly to the space approach of Graasp. Access rights are defined by keys which represent access to specific spaces. This mainly focuses on sharing of documents and works fine so far. Sharing arbitrary functional parts of an application under different rights with different participants is not feasible with this metaphor.

Within social networks, the circle metaphor introduced by Google+[4] tries to enable an easy grouping of users that should receive the same content. This particular solution is quite interesting for solving sub problems but misses a strategy to represent different rights for different parts of an application efficiently.

The most content or document management systems, like Typo3[5], Drupal[6] or Joomla[7], facilitates the matrix metaphor to manage the user's access rights on different data objects. These approaches potentially allow for fine-grained sharing definitions with different rights on all sub parts of a mashup application. But, as the number of users and elements of the application increases, the visualization gets quite complex and hard to understand for non-programmers. In addition, highlighting dependencies between sharing definitions or respecting different used components of different users is not possible.

A number of solutions, like (Angulo et al., 2012) or (Drogkaris et al., 2014), presented UIs for defining and reviewing the consumption of private data of third party service providers in websites. These approaches work well for reviewing and configuring requested data access from, e. g., different API providers, but do not offer any possibility to generically create access definitions for parts of an application's underlying data model for collaborating participants.

In summary, non of the examined approaches offer full support for the requirements and research questions that arise when enabling end-users to individually share parts of a CWA.

8 CONCLUSION

Today's mashup platforms offer no or only limited support for collaborative applications. We argue that CWAs are promising for non-programmers to use and adapt individually created applications for synchronous usage scenarios. In this paper, we presented an approach for the rights management as well as an adequate UI concept in context of collaboratively used CWA. After presenting the current state of the CRUISE runtime environment for CWAs based on black-box components, the basic rights management concept using triples of subject, object, and permission were introduced in Section 4. The access right concept allows to assign multiple owners to different parts of an application. The proposed sharing process is empowered by a client- and server-side ACL. To support end users with no programming skills in creating and managing different access rights for their CWA during runtime, we propose a triple-based overview dialog, with two major functionalities: First, it provides a possibility to share parts of the CWA with single users or a group of users during runtime. Second, the UI enables an easy to understand visualization of existing right assignments where the user is the inviter or the invitee and their current state. To present awareness information about shared components and the present permissions in the application's live view, Section 5.2 introduced a set of icons and interaction techniques. We demonstrate the technical feasibility of the approach by implementing the complete rights management concept as well as major parts of the UI within the CSR platform. Finally, a user survey showed that the concepts work as ex-

[4]https://plus.google.com

[5]http://typo3.org/

[6]https://drupal.org/

[7]http://www.joomla.de/

pected and are accepted by the target group. Thereby, the majority of mentioned drawbacks and suggestions for improvements are already respected within the presented concepts.

In future, we iteratively will re-work our approach based on the results of a second user study. Thereby, we primary focus on the sharing of component parts, where we plan to conduct an A/B testing to compare a task-centered sharing approach against a data-centered sharing approach. In addition, we extend the overall user guidance for sharing single parts of components. In this context we strive for a concept to identify and prevent erroneous or meaningless sharings leveraging semantic annotations and heuristics.

ACKNOWLEDGEMENTS

Carsten Radeck is funded by the Free State of Saxony and the European Union within the EFRE program. Gregor Blichmann is funded by the German Federal Ministry of Economic Affairs and Energy (ref. no. 01MU13001D).

REFERENCES

Angulo, J., FischerHbner, S., Wstlund, E., and Pulls, T. (2012). Towards usable privacy policy display and management. *Information Management & Computer Security*, 20(1):4–17.

Ardito, C., Bottoni, P., Costabile, M. F., Desolda, G., Matera, M., and Picozzi, M. (2014). Creation and use of service-based distributed interactive workspaces. *Journal of Visual Languages & Computing*, 25(6):717 – 726. Distributed Multimedia Systems {DMS2014} Part I.

Blichmann, G., Radeck, C., Hahn, S., and Meißner, K. (2015). Component-based workspace awareness support for composite web applications. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWas 2015)*.

Blichmann, G., Radeck, C., and Meißner, K. (2013). Enabling End Users to Build Situational Collaborative Mashups at Runtime. In *Proceedings of the 8th International Conference on Internet and Web Applications and Services (ICIW2013)*, pages 120 – 123.

Bogdanov, E. (2013). *Widgets and Spaces: Personal & Contextual Portability and Plasticity with OpenSocial*. Theses, Ecole Polytechnique Fédérale de Lausanne (EPFL).

Drogkaris, P., Gritzalis, A., and Lambrinoudakis, C. (2014). Empowering users to specify and manage their privacy preferences in e-government environments. In K, A. and Francesconi, E., editors, *Electronic Government and the Information Systems Perspective*, volume 8650 of *Lecture Notes in Computer Science*, pages 237–245. Springer International Publishing.

Husmann, M., Nebeling, M., and Norrie, M. C. (2013). Multimasher: A visual tool for multi-device mashups. In Sheng, Q. Z. and Kjeldskov, J., editors, *Current Trends in Web Engineering - ICWE 2013 International Workshops ComposableWeb, QWE, MDWE, DMSSW, EMotions, CSE, SSN, and PhD Symposium, Aalborg, Denmark, July 8-12, 2013. Revised Selected Papers*, volume 8295 of *Lecture Notes in Computer Science*, pages 27–38. Springer.

Picozzi, M. (2013). *End-User Development of Mashups: Models, Composition Paradigms and Tools*. PhD thesis, Politcnico di Milano.

Pietschmann, S. (2009). A Model-Driven Development Process and Runtime Platform for Adaptive Composite Web Applications. *Technology*, 2(4):277–288.

Radeck, C., Blichmann, G., and Meißner, K. (2013). CapView - Functionality-Aware Visual Mashup Development for Non-programmers. In Daniel, F., Dolog, P., and Li, Q., editors, *Web Engineering*, volume 7977 of *Lecture Notes in Computer Science*, pages 140–155. Springer Berlin Heidelberg.

Radeck, C., Blichmann, G., Mroß, O., and Meißner, K. (2014). Semantic mediation techniques for composite web applications. In Casteleyn, S., Rossi, G., and Winckler, M., editors, *Web Engineering*, volume 8541 of *Lecture Notes in Computer Science*, pages 450–459. Springer International Publishing.

Radeck, C., Lorz, A., Blichmann, G., and Meißner, K. (2012). Hybrid Recommendation of Composition Knowledge for End User Development of Mashups. In *Proceedings of the Seventh International Conference on Internet and Web Applications and Services (ICIW 2012)*.

Schümmer, T., Haake, J. M., and Haake, A. (2005). A metaphor and user interface for managing access permissions in shared workspace systems. In Hemmje, M., Niederée, C., and Risse, T., editors, *From Integrated Publication and Information Systems to Virtual Information and Knowledge Environments, Essays Dedicated to Erich J. Neuhold on the Occasion of His 65th Birthday*, volume 3379 of *Lecture Notes in Computer Science*, pages 251–260. Springer.

Tschudnowsky, A., Hertel, M., Wiedemann, F., and Gaedke, M. (2014). Towards real-time collaboration in user interface mashups. In Obaidat, M. S., Holzinger, A., van Sinderen, M., and Dolog, P., editors, *ICE-B 2014 - Proceedings of the 11th International Conference on e-Business, Vienna, Austria, 28-30 August, 2014*, pages 193–200. SciTePress.