Facilitating Context-Awareness in Composite Mashup Applications

Stefan Pietschmann, Carsten Radeck, and Klaus Meißner
Technische Universität Dresden, Germany
{*Stefan.Pietschmann,Carsten.Radeck,Klaus.Meissner*}*@tu-dresden.de*

***Abstract*—Dynamic adaptation and adaptivity of web applications have been subject to research for over a decade. With the shift from document-centered hypermedia to rich web applications, i. e., software as a service solutions, the applicability of traditional adaptation methods and techniques is in question. We first investigate what it means to facilitate adaptation within composite service-oriented applications and deduce adequate adaptation techniques. Then, we introduce a generic adaptation system which can be integrated with existing composition platforms to facility runtime adaptivity on a component and composition level, based on a platform-independent definition of adaptive behavior. With the help of a comprehensive sample application we eventually show the suitability and practicability of our approach.**

***Keywords*-mashups; composite applications; context-awareness; adaptation; CRUISe.**

I. INTRODUCTION

The Internet has become an open application platform and is not anymore a static, document-centered source of information. Based on the service-oriented paradigm, *mashups*, i. e., applications composing distributed web resources, have gained momentum. While initially their use was limited to the integration of data and business logic based on proprietary platforms, recently the need for *presentation integration* has been underlined [1]. Several research projects have since addressed this problem resulting in lightweight component and composition models that even allow for a *universal composition* [2], [3] of applications. This entails the uniform description and integration of resources covering different application layers, ranging from services encapsulating data and business logic, to user interface parts.

Alongside this development, the context of web applications has become increasingly complex and diverse. Developers are facing a growing heterogeneity of users, devices, and usage situations. Applications can no longer be developed for specific platforms or environments: they shall be available and usable both in the office with a desktop computer and in the tram using an iPad or the like. Users of different skills should be able to interact with them, taking into account specific user abilities and preferences, and also their current situations.

Context-awareness, especially in the web application domain as addressed by our work, has been subject to research for a long time. Yet, most of this research has been targeted at "traditional" and closed-corpus hypermedia systems [4]. Alternative approaches from service computing have studied adaptive service composition at the data and business logic levels, primarily with regard to QoS measures. However, presentation-oriented, universal compositions pose new challenges regarding the dynamic configuration and composition, which have not been addressed by prevalent concepts [5].

The adaptation of composite applications can be looked at from different angles. On the one hand – since we propose a model-driven development [3] – design time concepts for the authoring, reuse, and maintenance of adaptive behavior are needed and discussed in [6]. On the other hand, runtime adaptation involves both the initial context-aware service selection and the adaptation of a composition. This paper focuses on the latter, i. e., the system architecture and mechanisms necessary to monitor, manage, and use context information to dynamically adapt composite mashups.

Starting from traditional adaptation methods and techniques, we investigate into techniques facilitating adaptation and adaptability within mashups. We then present a generic adaptation system designed to be used in conjunction with existing composition environments. It addresses the shortcomings of available concepts and combines context monitoring and management, the platform-independent modeling of adaptive behavior, its evaluation and finally the realization of dynamic adaptations at the component and composition level at runtime. Thereby, adaptive behavior and adaptation techniques are decoupled from concrete platforms and applications, catering reusability, maintenance and extensibility.

In the next section we define the conceptual foundations for our system, present exemplary adaptation scenarios, and deduce corresponding adaptation techniques. Afterwards, we discuss related efforts to enable context-awareness in application and service compositions. After giving a brief overview of our concept, the following section presents the application-independent definition of adaptive behavior, context management facilities, and the realization of dynamic adaptation techniques including component exchange. We then provide details on the implementation of our system, its integration with the CRUISe composition environment [3], and its validation with the help of a sample application. Finally, we conclude this paper and outline future work.

 ISBN: 978-1-61208-156-4

II. Adaptation in Mashup Applications

To facilitate adaptive behavior in mashups, we first need to clarify the relation between traditional *Adaptive Hypermedia* and modern applications "mashed up" from distributed web resources. To this end, this section provides a brief characterization of the type of applications addressed by our work, presents exemplary adaptation use cases and deduces corresponding techniques.

A. Composite Mashup Applications

Mashups indicate a way to create new applications by combining existing web resources utilizing data and web APIs [5]. While originally restricted to data and application logic, there are ongoing efforts to enable compositions including user interfaces. In contrast to most programmatic mashups, we focus on *universal* composition efforts [3], [2] which imply certain component and composition models. Thus, we call them *composite* mashup applications (CMA) to denote their component-based nature.

While *Adaptive Hypermedia* stipulates closed-corpus systems [4] for data-driven, document-centered applications, CMAs are composed from black-box software components. Those can be integrated, used and adapted only via their interfaces, described by WSDL in the case of web services. Most importantly, these "building blocks" are distributed and strongly decoupled by definition. Despite this, the resulting applications are highly interactive, so components must by tightly integrated. Finally, unlike traditional applications using a hypernet of web pages, CMAs are usually single-page solutions offering different views on an application.

It becomes evident that techniques like *page variants*, *sorting fragments*, or *link annotation* can not be applied to CMAs directly, as they are based on different metaphors. While *pages* can be considered *views* in our vocabulary of concepts, and *fragments* may correlate with *components*, the techniques still imply hypertext documents and are rather located *within* components in our concept space.

Next we present a number of exemplary adaptation use cases, which allow us to deduce higher-level adaptation techniques specific to CMAs.

B. Adaptation Use Cases

To illustrate the set of adaptation possibilities, it is necessary to identify the adaptation triggers, i. e., the context information available. Generally, four context categories can be taken into account [7]: First, the *delivery context* defines the technical environment of the application at runtime, e. g., the capabilities of the device, web browser and network connection. The second category includes the *user*, his identity, roles, characteristics, preferences, etc. Third, the *physical environment* is characterized by information such as the location, noise level, and brightness. The last aspect includes *situation and time*, e. g., being at home or at work, the current weather, or the season. Since CMAs are composed from distributed services, the *quality of service* is an additional trigger which has been intensively studied in the context of web service compositions (cf. [8]).

This wide range of context information implies a variety of adaptation scenarios. A special field that has gained lots of popularity are *location-based services*, such as adaptive route planners. The following use cases give a practical insight into the possibilities and challenges of context-awareness: Imagine, you are on vacation and use the services provided by a travel guide application: it shows locations of nearby restaurants and hotels, or sights on a map. For each of them, you can check further details in the form of videos, images, and text, calculate a route. The suitability and usability of the application in this rather simple scenario largely depends on the reasonable integration and use of context knowledge. Examples including the type of context (in parentheses) are:

- The application including all components use your native language (*user*).
- The map initially centers at the your current position by default (*physical*).
- All components are constantly synchronized with your location: the route planner adapts the route calculation once you move (*physical*).
- The selection of locations matches your interests and preferences: a vegetarian user is offered special restaurants that cater his needs (*user*).
- Route planning considers your mobility (foot/car) and whether you have any disabilities that might impact on transfer times (*user and situation*).
- The layout changes with respect to the available screen real estate, and big images are replaced with textual descriptions on mobile devices (*delivery*).
- You can minimize, remove, and exchange components, e. g., to choose your preferred map provider – a choice that is monitored and stored to select the correct component for you in the future (*user*).
- A low battery level on your mobile device triggers the exchange of resource-intensive components, e. g., of a video player with an image slide show (*delivery*).
- Once a component or service becomes unavailable, it is automatically replaced with an alternative (*QoS*).

C. Adaptation Techniques for CMAs

While the abstract *adaptation methods* as presented in [9] and the like remain valid, the corresponding techniques must be updated and applied to the notion of CMAs. We regard *adaptation techniques* as any alteration of an application composition subject to context changes. Hence, adaptation forms an additional aspect that can be applied to a composition model. The reasons therefore range from *adaptive*, *corrective*, *perfective*, to *extensive adaptations* [10]. Consequently, runtime adaptation for CMAs can be realized with the following techniques:

 ISBN: 978-1-61208-156-4

Component Adaptation

- **Adding Components** to the composition supports perfective and extensive adaptations: New components may provide additional data and information to support a user in his task.
- **Removing Components** which are not used or not necessary anymore facilitates perfective adaptations. It must be ensured that removals don't affect application operability and stability, and that data loss is prevented.
- **Exchanging Components** combines the above techniques for corrective and adaptive reasons. Challenges include the state transfer and concurrency problems during the exchange.
- **Reconfiguring Components** basically changes configuration properties, which can result in internal state changes, the use of different algorithms, data sources, etc. This technique is limited to adaptation scenarios foreseen by component developers.
- **Adapting Component Interfaces** implies a change of a component's external interface for interoperability reasons, e. g., when exchanging one component with another. This technique involves both the adaptation of operations, events or the like using adapters, and the mapping or mediation of data.
- **Migrating Components** between different devices, or between client and server, can be seen as a combination of addition and removal along the lines of component *exchange* (see above).

Composition Adaptation

- **Adapting Communication** between components is a powerful technique to adapt control and data flow to the context.
- **Adapting Layout** of the application means to rearrange frontend components on the canvas. This can also involve changes to dimensions of individual components within the layout.
- **Adapting Screenflow** changes the order of views, i. e., which UI component is visible at a certain application state, and the transition between those views.

As stated in [11], both *parameter adaptation* (reconfiguration) and *architectural adaptation* (addition, removal, exchange) are needed and sufficient to support all scenarios of dynamic adaptation. Thus, with the help of the above-mentioned techniques, all adaptation examples sketched out before are supported.

An additional dimension of adaptation includes the dynamic reconfiguration and migration of the runtime environment, e. g., the dynamic redistribution between client and server depending on the server load. In this paper, though, we focus on the adaptation of the component-based application within a fixed composition environment.

III. Related Work

Several concepts for runtime adaptation have been proposed. A typical representative from *Adaptive Hypermedia* is AMACONT [12], which offers an extensible context framework – similar to our solution – and realizes adaptation with the help context-dependent variants being part of its document model. In contrast to our work, the concepts are based on dynamically generated documents, and adaptation takes place only when a new document is requested.

Schmidt et al. [13] introduce an event-based adaptation system for RIAs, which employs an ontology-based user model and Event-Condition-Action adaptation rules, both of which served as inspiration for our approach. The solution does not imply the use of any component or composition models, though, and concentrates on event detection and condition evaluation. Consequently, techniques like component exchange are not supported.

ACCADA [14] uses architectural models and an external control loop for context monitoring and dynamic adaptation. Thus, adaptation logic is strictly separated from the application. However, it is used to satisfy functional constraints only – components and adaptation techniques at the presentation layer are not considered. Similarly, the extensive body of research in the field of dynamic service composition fails in this regard. The joint effort here is to enforce composition plans and utility functions. While initially, this involved service exchange with regard to functional and QoS requirements [8], more complex service adaptations [15] and more diverse context information [16] have been used, lately.

Only few academic works have addressed context-awareness for mashup systems, most of which again focus on the initial composition [17] as discussed above. Most importantly, the Mixup framework [18] comes with a component model and an event-based runtime similar to our concepts. *Context components* resemble sensors which publish events upon context changes. A sophisticated context management is not included, though, and the adaptation techniques are quite limited, e. g., adaptive layout and screen flow as well as component exchange are not supported. Further, context components (resembling monitors) are first class concepts of an application, while we strive for decoupling adaptive behavior from application, i. e., composition logic.

In summary, none of the approaches to date provides a generic solution which covers both context management and dynamic adaptation for interactive CMAs. In the next section we introduce our solution to this end.

IV. Adapting Composite Mashups

In this section we present a concept for the runtime adaptation of component-based mashup applications. Therefore, we propose an adaptation system which can be used with existing composition environments – in our case the CRUISe platform [3]. The latter facilitates context-aware component

selection and integration at application initialization time. Based on an abstract composition model, suitable components are dynamically chosen with regard to the particular context. The concept presented here focuses on adaptation at run time, once the application has been initially composed. Following a brief overview of the conceptual architecture, we discuss the definition of adaptive behavior, the strategy to monitor and manage context data, the realization of adaptation techniques, and the adaptation workflow, subsequently.

A. Architectural Overview

As Figure 1 shows, our adaptation infrastructure is divided into two parts responsible for the adaptation and context management, which are coupled with a composition environment.

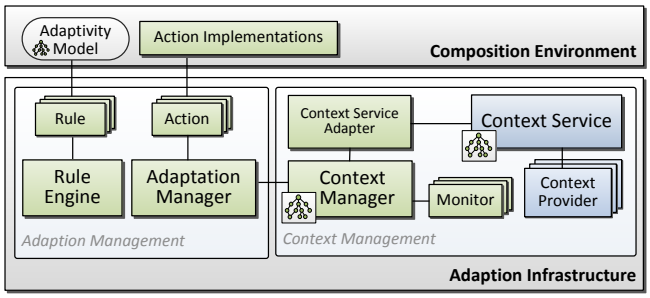

Figure 1. Overview of the adaptation system

Adaptation Management carries out adaptations based on generic Event-Condition-Action (ECA) rules. Those *adaptation rules* address specific parts of a composite application (component properties and instances, layouts, etc.) to be modified by adaptation *actions* subject to (optional) conditions. The evaluation of adaptation rules is carried out by the *Rule Engine*, which subsequently calls the *Adaptation Manager* to execute adaptation actions as specified.

Context Management comprises both context monitoring and modeling. Monitoring within our architecture is carried out by *Monitor* components. Context modeling, reasoning and consistency checking is done by a dedicated system. We argue that this complex task is best externalized, e. g., to a *Context Service* like discussed later. To increase performance and decrease network load, the *Context Manager* holds a local representation of a part of the remote context model and is updated whenever the latter changes. To keep the system flexible and extensible, the *Context Service Adapter* concept hides service specifics and thus makes it easily exchangeable.

The application-specific configuration of the adaptation infrastructure is deduced from platform-specific composition models. They include adaptive behavior, but may also define the supported levels of adaptability and the configuration of context monitors, e. g., update thresholds. Optionally, the context service can provide a structural description of the context model to allow the service adapter to do the necessary mappings.

B. Description of Adaptive Behavior

The definition of adaptive behavior is usually intertwined with application code and heavily depends on the composition platform used. For the adaptation system presented here, we use a platform-independent representation in the form of declarative XML-based ECA rules. They come with certain implications regarding the composition environment, such as event-based communication and a uniform component model. Yet, those are concepts found with the majority of existing mashup systems, ranging from industrial solutions like Yahoo! Pipes to academic approaches such as mashArt [2]. Generally, the rules remain independent from specific composition platforms and languages. They are either written manually, but can also – as in our case – be generated from corresponding information in the composition model [6]. Each of these *Adaptation Rules* consists of three parts: *event* specifies triggers for the adaptation, *condition*s need to be fulfilled to carry out the adaptation, and *action* defines the adaptation techniques to be applied. Additionally, every rule has a unique *id* and a *priority* to define the processing order.

Events define the triggers for adaptation, which may not only be published by application components, but by arbitrary parts of the composition environment, as well. *Application events* are actuated by components of a CMA and usually signalize state changes. *Context events* are issued by the *Context Manager* and contain updates of context parameters. *Runtime events* are published by the composition environment, e. g., upon successful component integration. Thus, they can be used to realize error handling with the help of adaptation rules instead of imperative programming. Finally, *complex events* describe non-atomic event patterns, such as causal dependencies. This way, a certain sequence of events can be defined to trigger an adaptation.

Listing 1 shows an exemplary event part of an adaptation rule. The latter is evaluated whenever a change of the user's device's battery level is detected and propagated.

```
1   <contextEvent contextParam="/user:currentDevice/dev
        :state/dev:batteryLevel" />
```

Listing 1. Definition of a rule's trigger *event*

The execution of adaptation rules is subject to **Conditions**. These consist of *terms* that can be combined with the help of logic operators (`AND`, `OR`) and use binary comparison operators (>,>=,!=, etc.). Further operators are supported, such as `CONTAINS` or such useful for semantic context representations (`TYPE`, `ISA`). Terms contain literal values or refer to context or event parameters. A typical case for using the latter is the extraction of a menu item's name

from a corresponding `itemSelected` event. In case of more complex conditions, SPARQL [19] can be used.

Listing 2 shows a condition, which restricts the adaptation to mobile devices (lines 7–10) with a battery level below 30% (lines 3–6).

```
1  <condition>
2    <and>
3      <term operator="lt">
4        <contextParam>/user:currentDevice/dev:state/dev
             :batteryLevel</contextParam>
5        <literal> 0.3 </literal>
6      </term>
7      <term operator="eq">
8        <contextParam>/user:currentDevice/dev:isMobile
             </contextParam>
9        <literal> true </literal>
10     </term>
11   </and>
12 </condition>
```

Listing 2. Exemplary *condition* for a rule

Finally, rules contain **Actions** representing concrete adaptation techniques, which are processed following their priority in a descending order. We use a generic vocabulary for these actions, which is derived from the techniques presented in Section II and easily extensible. It includes *component actions* addressing one or multiple components (e. g., removeComponent, reconfigure, setVisibility), *channel actions* adapting communication relationships (e. g., subscribe, unsubscribe, removeChannel, fireEvent), and *complex actions* changing application-wide concepts (e. g., changeLayout, updateContext).

Listing 3 shows an exemplary component reconfiguration – in this case the property *map_city* is set to the user's current location (a city) referenced from the context model.

```
1 <componentAction priority="9" pointcut="sightmap">
2   <reconfigureComponent property="map_city">
3     <contextParam>/user:currentLocation/space:name</
          contextParam>
4   </reconfigureComponent>
5 </componentAction>
```

Listing 3. Adaptation action to reconfigure a component property

The availability and consistency of context data is ensured by several components forming the context management subsystem, which are discussed in the next section.

C. Context Monitoring and Management

In this section we discuss how context is monitored, stored, and made available to adaptation system.

1) ***Remote Context Management – CroCo:*** Due to its complexity, we externalize context management to a dedicated *Context Service*. Therefore, we use the ontology-based service CroCo [20], which allows arbitrary *context providers* to submit, and *context consumers* to request context data via specific service interfaces. Thus, it serves as a general context supplier for external context-aware applications and platforms, such as ours.

Within CroCo context data is represented by a generic ontology-based context model, which uses several sub-ontologies describing aspects like time, place, the user, and his device. With the help of *domain profiles*, this model can be extended with additional concepts and relationships particular for specific domains. For the scenario mentioned before, we use concepts such as `UserProfile` (related to a `Person`), `WebBrowserConfiguration` (related to a `WebBrowser` being part of a `UserProfile`), and `UserLocation` (a type of `City`).

By using CroCo, our adaptation architecture highly benefits from its sophisticated means to check the consistency of the context model, to detect conflicts, and to infer additional knowledge with the help of semantic reasoning. These mechanisms guarantee, that we can rely on a valid and comprehensive model at minimal cost. Context information can either by requested synchronously, or consumers register for specific data, so that they are notified once it changes.

More detailed information on its architecture, inner workings, and the ontology-based model can be found in [20], including several examples of use.

2) ***Local Context Management:*** Our concept entails a local (client-side) context management to stay independent from the context service used, and to improve performance. A slice of the context model relevant for the application is cached by the adaptation system. That way, not every context request must be sent to the remote service, but can be evaluated with the local model which is continuously synchronized in the background. The following components are responsible for context management within our system.

Context Monitors are software components of the adaptation system that sense context data. Once it changes, monitors publish context updates to the *Context Manager*. Internally, they process raw context data to comply with the context model used, e. g., by mapping it to semantic concepts of an ontology. In addition to the actual data, they can provide a *confidence* value as discussed in [20].

The behavior of monitors can be configured as part of the application model, e. g., to set the minimal time threshold between context updates, or to request user confirmation before context updates are published to CroCo. Monitors provided by our system itself need not cover all context parameters used by the application as arbitrary external context providers may as well contribute to the model. Further, by implementing a dedicated monitor interface, new sensors can easily be added.

The **Context Manager** provides a uniform interface to the context model, using the local and remote context store, transparently. Its central responsibility is the local management of context information to minimize synchronous requests to the *Context Service* for every single parameter needed. Therefore, it initially downloads all parameters required for component configurations and rule evaluations from CroCo and registers there to be notified in case of

 ISBN: 978-1-61208-156-4

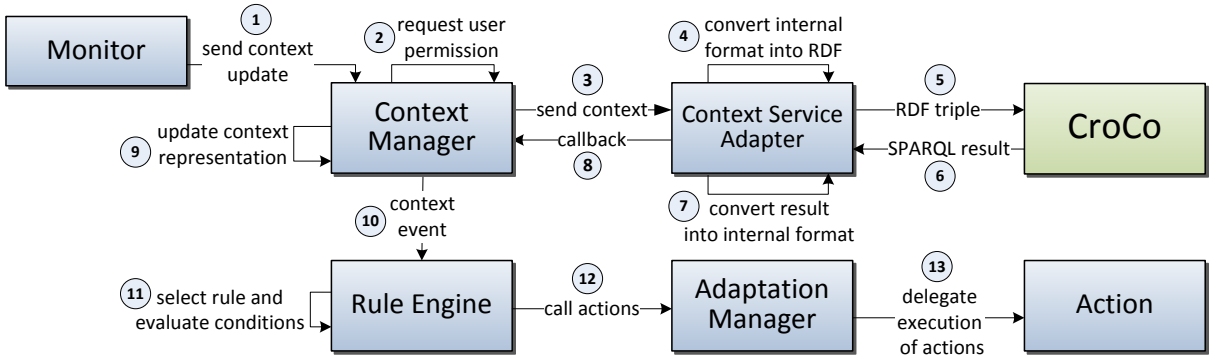

Figure 2. Workflow of the adaptation mechanism

model updates. Upon notification, the local model representation is synchronized with the remote model and context update events are issued, which trigger processing of the corresponding adaptation rules. Only on special occasions, e. g., when complex SPARQL requests need to be evaluated, requests are forwarded to CROCO.

Additionally, the *Context Manager* manages all local monitors and offers methods for (un)registering and (de)activating them. Their data is passed to CROCO for validation, consolidation, merging with the model and reasoning.

Communication with the external context management facility is handled by *Context Service Adapters* (CSA), which hide implementation and interface specifics of the context service used behind a uniform interface. Hence, it can be exchanged with an arbitrary local or remote solution, as long as a proper adapter exists. With regard to CROCO, the corresponding adapter transforms the context query and update events into RDF. Thereby, the implicit creation of instances and relations may be necessary. To accomplish this task, structural knowledge about the context model is needed, which can either be hard-coded into the adapter, or be provided dynamically by the context service, as is the case with our system.

D. Dynamic Adaptation Management

Figure 1 illustrates the modules involved in the dynamic adaptation process and their relation.

The **Adaptation Manager**'s main responsibility lies in providing and managing adaptation techniques. Whenever the evaluation of a rule implies a certain adaptation, the manager delegates its execution to the corresponding action implementation. It is also the central management entity within the adaptation system. Externally it offers a facade, i. e., an interface to request context parameters and process adaptation rules. Internally it is responsible for initializing and managing the other adaptation components. Before the application initialization, it loads configuration data and rules and subsequently initializes the *Rule Engine*, *Context Service Adapter* and *Context Manager*. The *Rule Engine* is provided with the necessary rules, and context parameter references from component configurations are replaced with the actual values requested from the *Context Manager*.

The **Rule Engine** is designed to evaluate adaptation rules. Therefore, it is automatically registered to be notified with all application and runtime events that are referenced by rules. Upon notification, the engine identifies all affected rules and sorts them with respect to their priority. Subsequently, their conditions are evaluated and the *Adaptation Manager* is called to execute the resulting actions.

Actions are platform-specific implementations of adaptation techniques as discussed above. For every abstract action there exists a corresponding implementation, which is naturally specific to the composition environment.

E. Adaptation Process

Figure 2 illustrates the steps of a dynamic adaptation and the components taking part in this process. Basically two very similar scenarios can be distinguished: In the simple case, context updates are sent from the external context service and result in an adaptation. In the extended case, updates result from data sensed by local monitors. Figure 2 presents the second case – the simple one is included starting with step six.

1. A monitor instructs the Context Manager to publish a detected context change.
2/3. The Context Manager checks, whether a user confirmation is required to publish such (possibly sensitive) data to the external context service. As discussed before, this can be configured as part of the adaptivity model. If a confirmation is needed, the data is presented to the user to be approved. Upon confirmation, the updated context data is handed to the CSA.
4/5. The latter parses the data, generates a representation particular to the context service and sends it there. Since we use CROCO, RDF triples are built to conform with its service interface.
6. CROCO carries out consistency checks, validation, and reasoning of context data. Then, context consumers registered with the affected data are notified. Consequently, the CSA is informed that parameters relevant for the evaluation of adaptation rules have changed.

 ISBN: 978-1-61208-156-4

7/8. After parsing and transforming the updated context data into the internal representation, the CSA forwards it to the Context Manager.

9/10. The latter updates its local representation of the context model and informs the Rule Engine about model changes using the corresponding context events.

11. The Rule Engine analyzes the events, searches for adaptation rules that apply to the updated context parameters, and evaluates their conditions. If additional context parameters are referenced, those are obtained with the help of the Context Manager.

12. Finally, the Rule Engine returns a list of adaptation actions to be executed to the Adaptation Manager.

13. The latter then chooses a suitable action implementation, triggers and coordinates its execution.

Even though the context information coming from monitors could be directly merged into the local model, it is instead first sent to the context service. This is for the obvious reason that CROCO employs mechanisms for validation, consistency checking and reasoning which determine whether the information are merged into the model or not. Furthermore, this round trip ensures that only the context data needed is processed by the Context Manager.

V. IMPLEMENTATION AND USAGE

Based on the concepts discussed we implemented a prototype of the adaptation system and integrated it with our composition platform CRUISe. In the following, we outline relevant parts of our implementation including an exemplary application realizing the travel guide scenario.

A. Integration of the adaptation system in CRUISe

The adaptation system was integrated with the CRUISe *Thin Server Runtime* (TSR), a browser-based composition environment. Its event-based communication allows for an easy integration by adding the Adaptation Manager as a publisher and subscriber to the global event bus. Several JavaScript-based monitors were implemented, e. g., to detect browser settings and the approximate location by means of the *Geolocation* API (http://dev.w3.org/geo/api/). The local context model is a JSON representation of CROCO's OWL-based model. Communication with CROCO is realized via SOAP using a dedicated adapter that generates SPARQL queries for synchronous requests. Asynchronous context notifications rely on CometD (http://cometd.org/).

The adaptation rules are derived from the composition model and dynamically evaluated by the Rule Engine. The corresponding actions are realized in JavaScript, extending a universal interface. Adaptability is offered by a component panel (see Figure 3) with options to minimize, close, and exchange components with alternatives. The latter proved to be the most challenging due to the dynamic integration of new code and the state transfer realized using proxies and the memento pattern.

B. Sample Application *TravelGuide*

To validate our concept and implementation, we built the *TravelGuide* application (Fig. 3) based on the use cases discussed at the beginning of this paper. To realize the desired behavior, a number of components were implemented, including a Google Map (on the right, indicating sights around the user and calculating routes there), a details form (bottom left), and an image gallery for the sight (top left). As an alternative to the latter, a video viewer was built which fetches videos of the sight from YouTube and has the same interface as the gallery, hence both are exchangeable. All visible components can be removed, restored, and explicitly exchanged with alternatives using a drop down menu.

The adaptivity modeled includes context-sensitive configurations, e. g., with user's native language, and the following behavior: 1) The map is continuously synchronized with the user's location; 2) His interest in cultural activities leads to the display of optional sights, e. g., museums, on the map; 3) The overall layout depends on the available screen estate and dynamically switches between a single- and multi-column layout; 4) When the battery power of the user's device falls below 30%, the video player component is exchanged with the image gallery in case the device is mobile.

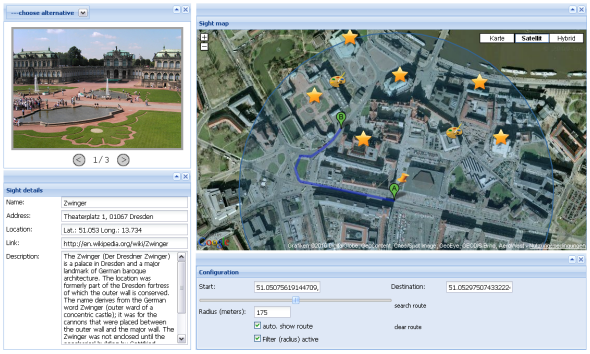

Figure 3. Adaptive composite *TravelGuide* application

Overall, our prototype proved to be stable and sufficiently supported all of the adaptation techniques defined, including the adaptation of components, their properties and the layout – both implicitly and explicitly by the user – as well as the dynamic adaptation of the control and data flow. Adaptation of component interfaces, i. e., mediation, was not explicitely modeled, as it is inherentely provided by the CRUISe platform.

VI. CONCLUSION AND FUTURE WORK

In this paper we present an adaptation concept which successfully transfers the principles of traditional *Adaptive Hypermedia* techniques to composite mashup applications. The definition of adaptive behavior is based on platform- and application-independent rules. They define dynamic adaptation techniques, such as component reconfiguration and exchange, data and control flow as well as adaptive

 ISBN: 978-1-61208-156-4

layout and screen flow at run time. To this end, the system provides context monitoring as well as context management entities. For the latter, a dedicated, ontology-based context management service is used, which can be transparently exchanged. A prototypical implementation of our solution and its integration with the CRUISe composition platform proved the feasibility and practicability of our concept.

In summary, we provide a valuable approach for the adaptation of presentation-oriented mashup applications. It combines a sophisticated context management with an efficient adaptation runtime, while keeping all means of context-awareness separated from the application. Thus, adaptation can be handled as an additional aspect. Our system can be integrated with other composition environments, provided that platform-specific adaptation actions are implemented.

One aspect that proved challenging was the definition of adaptive behavior. Even though the rules are easy to understand, the interactions between rules and the interactions with component-internal adaptivity are hard to overlook at design-time. Therefore, we are currently investigating on how to improve modeling and authoring of adaptive behavior. As part of this effort, we strive for a simplification and abstraction of the adaptivity model including higher-level *adaptation aspects* which take into consideration the semantics and self-adaptation capabilities of components. Furthermore, we are working on improving context management, by 1) letting CROCO autonomously learn and gather knowledge with the help of other services and 2) incorporating a better deduction of context from user interactions.

VII. ACKNOWLEDGEMENTS

The CRUISe project was funded by the BMBF under promotional reference number 01IS08034-C. The work of Carsten Radeck is funded by the ESF and Free State Saxony (Germany) under reference number ESF-080951805.

REFERENCES

[1] S. Pietschmann, T. Nestler, and F. Daniel, "Application Composition at the Presentation Layer: Alternatives and Open Issues," in *Proc. of the Intl. Conf. on Information Integration and Web-based Applications & Services*, 2010.

[2] F. Daniel, F. Casati, B. Benatallah, and M.-C. Shan, "Hosted Universal Composition: Models, Languages and Infrastructure in mashArt," in *Proc. of the 28th Intl. Conf. on Conceptual Modeling*, November 2009.

[3] S. Pietschmann, "A Model-Driven Development Process and Runtime Platform for Adaptive Composite Web Applications," *Intl. Journal on Advances in Internet Technology*, vol. 4, 2010.

[4] P. Brusilovsky and N. Henze, "Open Corpus Adaptive Educational Hypermedia," in *Adaptive Web: Methods and Strategies of Web Personalization*, vol. 4321, 2007, pp. 671–696.

[5] D. Benslimane, S. Dustdar, and A. Sheth, "Service Mashups," *Internet Computing*, vol. 12, no. 5, pp. 13–15, 2008.

[6] S. Pietschmann, V. Tietz, J. Reimann, C. Liebing, M. Pohle, and K. Meißner, "A Metamodel for Context- Aware Component-Based Mashup Applications," in *Proc. of the Intl. Conf. on Information Integration and Web-based Applications & Services*, 2010.

[7] D. Lizcano, J. Soriano, M. Reyes, and J. J. Hierro, "EzWeb/FAST: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services," in *Proc. of the 10th Intl. Conf. on Information Integration and Web-based Applications & Services*. New York, NY, USA: ACM, 2008, pp. 15–24.

[8] A. Erradi, P. Maheshwari, and V. Tosic, "Policy-Driven Middleware for Self-adaptation of Web Services Compositions," in *Proc. of the Intl. Conf. on Middleware*, 2006, pp. 62–80.

[9] P. Brusilovsky, "Adaptive Hypermedia," *User Modeling and User-Adapted Interaction*, vol. 11, no. 1-2, pp. 87–110, 2001.

[10] A. Ketfi, N. Belkhatir, and P.-Y. Cunin, "Automatic Adaptation of Component-based Software: Issues and Experiences," in *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, 2002, pp. 1365–1371.

[11] K. Geihs, P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjorven, S. Hallsteinsen, G. Horn, M. U. Khan, A. Mamelli, G. A. Papadopoulos, N. Paspallis, R. Reichle, and E. Stav, "A comprehensive solution for application-level adaptation," *Software – Practice & Experience*, vol. 39, no. 4, 2009.

[12] M. Hinz, "Kontextsensitive Generierung adaptiver multimedialer Webanwendungen," Ph.D. dissertation, Technische Universität Dresden, 2008.

[13] K.-U. Schmidt, R. Stühmer, and L. Stojanovic, "Gaining Reactivity for Rich Internet Applications by Introducing Client-side Complex Event Processing and Declarative Rules," in *Proc. of the AAAI Spring Symposium on Intelligent Event Processing*, 2009, pp. 67–72.

[14] N. Gui, V. Florio, H. Sun, and C. Blondia, "ACCADA: A Framework for Continuous Context-Aware Deployment and Adaptation," in *Proc. of the 11th Intl. Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2009, pp. 325–340.

[15] F. André, E. Daubert, and G. Gauvrit, "Towards a Generic Context-Aware Framework for Self- Adaptation of Service-Oriented Architectures," in *Prof. of the 5th Intl. Conf. on Internet and Web Applications and Services*, 2010.

[16] Q. Z. Sheng, B. Benatallah, Z. Maamar, and A. H. Ngu, "Configurable Composition and Adaptive Provisioning of Web Services," *IEEE Transactions on Services Computing (TSC)*, vol. 2, no. 1, pp. 34–49, 2009.

[17] T. Fischer, F. Bakalov, and A. Nauerz, "Towards an Automatic Service Composition for Generation of User-Sensitive Mashups," 2008.

[18] F. Daniel and M. Matera, "Mashing Up Context-Aware Web Applications: A Component-Based Development Approach," in *Proc. of the Intl. Conf. on Web Informa- tion Systems Engineering*, 2008, pp. 250–263.

[19] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," 2008, http://www.w3.org/TR/rdf-sparql-query/.

[20] A. Mitschick, S. Pietschmann, and K. Meißner, "An Ontology-Based, Cross-Application Context Modeling and Management Service," *Intl. Journal on Semantic Web and Information Systems*, Feb. 2010.

 ISBN: 978-1-61208-156-4