HAL
archives-ouvertes.fr

A Runtime Environment for Distributed Mashups in Multi-device Scenarios

Oliver Mroß, Klaus Meißner

▶ **To cite this version:**

Oliver Mroß, Klaus Meißner. A Runtime Environment for Distributed Mashups in Multi-device Scenarios. 9th International Conference on Network and Parallel Computing (NPC), Sep 2012, Gwangju, South Korea. pp.532-541, 10.1007/978-3-642-35606-3_63 . hal-01551337

HAL Id: hal-01551337
https://hal.inria.fr/hal-01551337

Submitted on 30 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

A Runtime Environment for Distributed Mashups in Multi-Device Scenarios

Oliver Mroß and Klaus Meißner

Faculty of Computer Science
Technische Universität Dresden, Germany
{oliver.mross, klaus.meissner}@tu-dresden.de

Abstract. Multi-device scenarios pave the way for new applications that allow the user to interact collaboratively with other users or enhance the interaction through the combination of different devices. To realize such applications, a new development paradigm is needed. Traditional methods do not support the dynamic extension or alteration of the application's feature set. Our approach is based on the idea of modeling the composition and distribution of several application components, so that the component implementations and their assignment to specific execution environments (component distribution) can be determined dynamically. All application components are linked together as an overlay network in the context of the multi-device environment. Therefore, we propose an architecture of a runtime environment and an overview of the component distribution process during the load-time of a multi-device mashup.

Keywords: Multi-Device Environment, Distributed Mashups, Runtime Environment, Model-based Mashup Development

1 Introduction

The quickly growing popularity of mobile devices and their rich feature set are leading to the need for applications, in which several interactive devices can be used in combination. Such applications allow the user to collaborate with other users across the borders of their personal devices. These scenarios range from simple data sharing to complex applications, in which multiple users can connect with each other and create a shared view on distributed content that can be modified from different devices, e. g., a collaborative decision making application. In each of these scenarios the devices are providing a distributed user interface of the application.

The device mobility can cause the dynamic adaption of the distribution state during the application's runtime. If another person and her Smartphone, for example, are entering the interaction environment, the application could integrate a new UI element that allows the person to take part in the collaboration. To distribute the user interface in the context of a multi-device environment the application developer has to provide a specific UI for each device. The problem here is the development effort to create multi-device applications. Another problem is the dynamic availability of mobile devices. The application developer has no knowledge of the configuration of the mul-

ti-device environment. This means that a static distribution of the user interface is not possible during the application's design-time.

In our approach, we address both problems through the use of the development methods from the mashup domain. Hence, we denote such applications as model-based distributed mashups. From the perspective of the application developer, these methods allow to reduce the development effort, because they introduce a strict separation between the application and the UI development. The application developer describes only the composition of the application, the communication between the application components and their distribution as part of an application model. We assume there are several classes of UI components and each class contains device specific UI component implementations. They are provided in a public component repository service that allows registering reusable black-box application components. To address the problem of the dynamic device availability, the composition and distribution descriptions declare properties of required application components and runtime environments, i. e., they do not include concrete assumptions about the availability of application components or devices. To make clear what we understand by the notion of a distributed model-based mashup application, we will give a simple development example now.

Example 1. An application developer defines an application model that includes the description of three application components. The first one is a UI component, which is responsible for the presentation and manipulation of a UML diagram in the context of a Smart TV. The second one is a service component that transforms all raw data of the acceleration and position sensor of a Smartphone into a two-dimensional representation, which is used as input parameter for the previously mentioned UI component. The third component has the same functionality as the second, but it has to be executed on a Tablet computer. Besides the component requirements, the distribution and the communication requirements between the application components are described in the application model as well. In this scenario it contains two communication channels. The first one connects the UI component with the service component that should be executed on the Smartphone. The second communication channel connects the previously mentioned UI component with the service component that should be executed on the Tablet computer.

To generate and execute the distributed mashup, a new kind of runtime environment is needed, which we will explain in this paper. Hence, the remaining paper is structured as follows. Related work will be discussed in the following Sect. 2. In Sect. 3 we will give an overview of the architecture of a runtime environment that enables the execution of a distributed mashup. After that, we will sketch the individual phases of the dynamic loading process of a model-based distributed mashup in Sect. 4. In the final Sect. 5 we draw conclusions and will give an outlook on the next research steps.

2 Related Work

In [1] the vision of a new model of pervasive applications is based on three precepts. The second precept says that different devices can be seen as portals to one virtual application. We share this view, but we understand the notion of a virtual application as a network of components that are distributed and executed on different devices in a distributed runtime environment. In the CRUISe project [2] a model-based development method and a browser-based thin-server-runtime [3] of mashup applications were developed. The application's composition on the presentation layer is defined as part of the composition model. It contains component templates that are matched and ranked against a set of already existing mashup components, i. e., the mashup application is generated dynamically during the application's load-time. However, the CRUISe approach does not support the execution of components in distributed runtime environments nor it provides needed operations to distribute UI components dynamically on a set of heterogeneous devices. Our current project DoCUMA is based on the CRUISe research results and will extend the mashup development methods to create applications for collaborative multi-device scenarios. To model the distribution of UI components during the design time of an application a meta-model was developed in [4]. Its entities represent digital and physical elements of the smart environment on a high-level abstraction layer. Specific aspects like the distribution requirement description that is needed to compute the dynamic distribution of application components are not part of the meta-model. The approaches in [5,6] describe the use of UI properties to characterize the distribution of abstract UI elements on model layer – the CARE-properties [7]. They allow to bind an abstract UI to a concrete device of the smart environment during the design time. Our approach addresses the dynamic distribution of black-box UI components, thus a static binding between UI components and their runtime environment is not feasible. Hence, a runtime environment descriptor for each interactive device is needed to compute the mashup component distribution dynamically. In [8] the device description model of the UPnP standard [9] is transferred into an ontological representation. It allows the dynamic determination of the valid mapping between the set of required mashup components and the set of their appropriate runtime environments.

Finally, in [10] a distributed runtime environment is described that enables the migration of UI components between heterogeneous devices. It is based on a reverse engineering process that comprises the generation of an abstract user interface model from a UI implementation, which is the starting point of a model-to-code transformation. The result of the transformation is a device specific UI. In our approach we use black-box UI components, which do not allow a reverse engineering in the way of the approach in [10]. To allow the migration of UI components between heterogeneous devices, we will focus on the concept of adapting the component implementation by replacing the migrating application component [11] with an optimal component that is a device specific variant of the same component class during the application's runtime.

3 Overview of the Architecture

In this Sect., the architecture of the runtime environment of a distributed mashup application is presented. In the remainder of this paper it is denoted as the **Client-Server-Runtime (CSR)**. Figure Fig. 1 visualizes the main parts of the CSR – the client- and the server-side runtime environment. Both can execute different application components in parallel, which are linked together as an overlay network that is built on top of the connected devices.

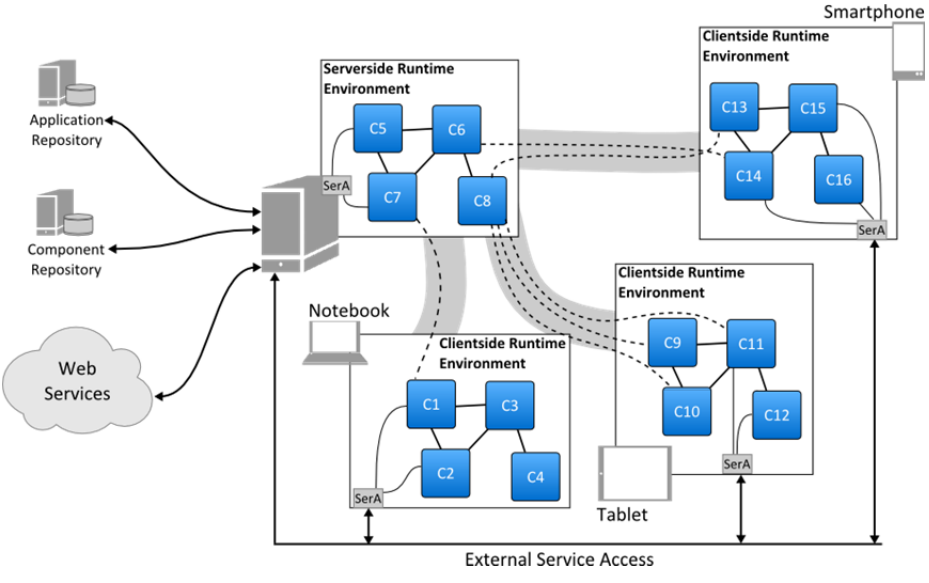

Fig. 1 Overview of the CSR

In this paper, the client-side part of the CSR is denoted as **CRTE** (*Client Side Runtime Environment*) and the server-side is denoted as **SRTE** (*Server Side Runtime Environment*). The CRTE is responsible for the instantiation, initialization, execution and visualization of the mashup UI components. Further, it encapsulates all communication features that are needed to execute a distributed mashup successfully. These include local and remote communication features, e. g., the creation of local and distributed communication channels. To migrate and replicate UI components between different devices, each CRTE provide serialization and deserialization capabilities that allow the exchange of component state information between different devices.

The SRTE is responsible to register multiple applications. Hence, it provides a service that allows the application developer to register several application models, which describe the application composition and communication patterns of several mashup components. Another feature is the dynamic discovery and registration of multiple devices and their runtime environment descriptions. Furthermore, the SRTE mediates messages between all distributed application components and between the runtime environments. Another important feature of the SRTE is the execution of the

application's loading process, in which the application components and the appropriate CRTEs are determined dynamically. This is discussed in Sect. 4.

In the left part of Fig. 1 several services are depicted, which are needed to load and execute a distributed mashup application. In the following Sect., we will dwell on this aspect.

Services of the CSR

Application Repository (AppRe). With the help of this service the application developer can provide predefined application models in the context of the multi-device environment. From this perspective the *AppRe* service acts as a central registry for multi-device applications whose execution context is the whole interactive environment. Furthermore, the service provides functionalities that enable the SRTE to publish all available multi-device applications like a broadcaster. This is necessary, because we assume that the user has no knowledge of the existence of the smart environment and its applications, e. g., in public places. This means that the SRTE and CRTE have to provide automatic discovery mechanisms, which enables the user to discover available multi-device applications. We assume further, that each user has installed the CRTE on his (mobile) device, since the discovery mechanisms cannot be realized only with the help of client-side web technologies.

Component Repository (CoRe). This service is a fundamental part of the generation procedure of the distributed mashup, because it provides functionalities to access several application component implementations and their interface descriptors in the SMCDL format *(Semantic Mash-up Component Description Language)* [2]. We assume that for each platform (Smartphones, Tablet computers, Desktops), there are specific application components, which provide the foundation of generating a distributed mashup across the borders of heterogeneous devices. Another fundamental function of this service is the registration of several application components. This includes the registration of device-specific components, which encapsulate services of the interactive device, e. g., the access to embedded sensors or in- and output devices.

After we have given an overview of our approach of a distributed runtime environment, we will describe the dynamic loading process of a model-based distributed mashup and its individual phases in the next Sect. 4.

4 Loading Process of a Distributed Mashup

In our approach the load-time of a model based distributed mashup can be divided into the following phases. Each phase uses runtime environment features that are distributed on the client- and server-side. A more detailed description on what middleware components are used in which phase, is not part of this paper.

Process Phases

Discovery. During this phase the SRTE collects several pieces of information. The first ones are characterizing the device and the software platform that forms the foundation to execute future application components (*CRTE discovery*). The second part of information are the user credentials (*user discovery*), which are needed to determine the valid set of available applications. In other words every user has a predefined application set that depends on the user preferences and the associated user role. The applications per user are the third information piece (*application discovery*). Once the user has logged in, he can choose to join an active or load an associated application. The first case is out of the scope of this paper.

Application Selection. After the completion of the discovery phase, the next step of the loading process is the presentation of the determined application set to the user. Therefore, the SRTE analyses every application description in the set of the available applications and sends a reduced information set to present all available application models as visual entries in the context of the client device. That means that the user can initiate the loading process of a distributed mashup from his personal device. We denote it as the *initial device*. After the user selects a model reference, a message is sent from the user's CRTE to the SRTE, where it causes the initiation of the server-side model interpretation procedure.

Interpretation. The main goal of this phase is to determine what mashup components are needed by the application, what components are available and what CRTEs could be used to realize the specified distributed mashup (represented by application model). Therefore, following steps are executed. First, the SRTE resolves the associated application model from the received model reference that was selected by the user. The model is used as input parameter during the invocation of the server-side mashup loading process. Second, all CRTE descriptors are retrieved and used as input parameter during the loading procedure invocation. That means that every CRTE could be used as a potential execution environment of the mashup components. Besides these two input parameters, the third information set – the interface descriptors of all application components – is already provided by third party component developers as entities of the previously mentioned *CoRe* service. After the SRTE has retrieved all required information, the interpretation procedure will be triggered. The knowledge of the needed application components is extracted from the application model. It encompasses functional properties of the required application components, e. g., supported properties, operations and event messages. Furthermore, the knowledge of the component distribution is extracted from the application model too, because in our approach the application developer defines the distribution per mashup component explicitly, i. e., there is an association between the description of the needed mashup component and the description of the required distribution context. The distribution context spans several aspects, e. g., device characteristics like the display resolution or context characteristics like the maximum distance between the initial device (that

represents the user's position) and the needed execution context of the bound mashup components. The characteristics of the available CRTEs are determined from the registered runtime descriptors. They include runtime environment aspects, e. g., the supported in- and output modalities and the software platform aspects, such as provided APIs or available rendering capabilities (audio and video player features, etc.). With the help of this information the SRTE can derive what application component is executable within the context of which CRTE.

Matching & Ranking. The main goal of this phase is the computation of the final distribution set that describes the distribution of those mashup components, which correspond on one side with the functional requirements and on the other side with the distribution requirements that are both part of the application model. In our approach the algorithm to compute the distribution set consists of the following steps. First, the interpreted information of the application model is used to determine the set of usable CRTEs. Therefore, the SRTE needs to consider the registered runtime environment descriptions, which characterize the functional and non-functional properties of the corresponding CRTE. In other words, the CRTE descriptors are matched against the descriptions of the required distribution context in the application model. In our approach, each distribution context description represents one required CRTE per mashup component. After the corresponding CRTE was determined, the associations between the distribution contexts and the functional component requirements are used to create a matrix that comprises the mapping between a CRTE and a mashup component requirement. In the second step of our algorithm, each entry of the previously mentioned matrix is used to compute the best matching application component implementation. The result of our distribution algorithm is a refined matrix (*distribution set*) that describes the mapping between the entries of the set of usable CRTEs and the entries of the set of available mashup component implementations.

Apportionment. In our approach a distributed mashup application is characterized as a network of communicating mashup components, which are executed in the distributed context of several runtime environments (CRTEs). This involves the execution of multiple application components in the context of a single CRTE. To integrate these components successfully, the communication description of the application model is needed during the integration process. That is, the model has to be apportioned into several model fragments, to reduce the communication overhead between the SRTE and CRTE. These fragments have to be bound to these mashup components, which are part of the previously mentioned distribution set. Hence, the set has to be extended with modal data, for example, communication channel descriptions per mashup component. The result of this phase is an extended distribution set that contains several entries and each entry incorporate the following information units: a unique identifier of the mashup component, a component descriptor (functional and non-functional component properties) in the SMCDL format, the executable code of the mashup component, model fragment information and the identifier of the corresponding CRTE.

The first parameter is created in the context of the SRTE, because it is responsible to route all communication messages between the distributed mashup components. The second parameter incorporates all information that is necessary to integrate the component in a remote execution context, e. g., external software libraries. The third parameter covers all information that is necessary for instantiating and executing the application components. The fourth parameter contains the model data, e. g., the description of the communication behavior between the mashup components. The last parameter represents the CRTE, in which context the application components should be executed.

Integration. After the extended distribution set was created, the last phase of the loading process will begin – the distributed integration of all included mashup components. It covers the instantiation, initialization and the optional layout- and rendering-process of several mashup components in the remote context of different CRTEs. For this purpose the SRTE has created a bidirectional communication channel to every CRTE during the aforementioned discovery phase. In our approach we will use Web Sockets, which allow the SRTE to push application specific messages to the client-side. As input parameter the extended distribution set is used in this phase. First the set is sorted under the criteria of the CRTE, because this allows to group the distribution set and each group covers all mashup components that are executed in the same context. Second, for each group an integration command message is created, which causes the associated CRTE to start a client-side integration procedure, in that each delivered mashup component is instantiated and initialized. This message contains all information of each group member (component ID, component description, executable code and model fragment data) that is needed for the remote integration procedure.

5 Conclusion and Further Work

In this paper, we presented our approach to realize a distributed mashup application for multi-device scenarios. We used model-based development methods that are known from the mashup domain, e. g., modeling the composition and communication of application components. To distribute mashup components on heterogeneous devices further conditions must be fulfilled during the load-time. To address the problem of the dynamic availability of mobile devices every personal device has to provide a discovery feature that allows the server-side device registration and the client-side application selection. Modern web browser do not provide such discovery features, that means we need to extent the browser capabilities to use it as a universal runtime environment for mashup components. Furthermore, in our approach an application is provided and executed by the multi-device environment and not by the device itself. Hence, the personal device of the user has to communicate with a central server that provides all available applications.

Another problem that we address is the heterogeneity of multiple interactive devices. To overcome this issue, in our approach for every device a specific mashup com-

ponent implementation has to be registered in a public component repository service. Each mashup component is associated with a descriptor that contains functional and non-functional property information, such as supported operations and capabilities. Especially, the non-functional component description part contains requirement descriptions from the perspective of the component developer, e. g., needed device-specific sensors or in- and output resources. That means that the client-side runtime environment has to provide self-descriptive properties that condition the deployability of a mashup component. Hence, each potential runtime environment provides a generic interface that allows aggregating the runtime environment descriptors on the server-side during the discovery phase.

As further research, we will investigate the possibilities to express the distribution requirements per needed mashup component as part of the application model with the help of semantic technologies, e. g., SPARQL-queries[1]. We assume that this will help us to simplify the matching process, because these queries can be used directly in the matching phase of the application loading process. No additional interpretation and transformation operations have to be implemented. Furthermore, we will extend existing standards for device description, e. g., the UPnP standard, with additional non-functional properties, because we need to describe what device-specific (in- and output) and what software-specific conditions must be fulfilled to execute a mashup component. For example, a component developer wants to annotate that his mashup component is only executable on a smartphone with a camera and a Webkit-based browser. Therefore, we plan to describe these runtime environment properties in an ontological representation format, because we assume that semantic technologies will help us to reduce the implementation effort and improve the quality of the dynamic matching between required distribution contexts and the available runtime environments.

Acknowledgement

The work of Oliver Mroß is founded by the European Social Fund (ESF), Free State Saxony (Germany) and T-Systems Multimedia Solutions GmbH (Germany, Dresden) and is filed under ESF-080951831.

References

1. Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., Zukowski, D.: Challenges : An Application Model for Pervasive Computing. In: Proceedings of the 6th annual international conference on Mobile computing and networking - MobiCom'00. pp. 266-274. ACM Press, New York, New York, USA (2000).
2. Pietschmann, S., Radeck, C., Meißner, K.: Semantics-based discovery, selection and mediation for presentation-oriented mashups. In: Proceedings of the 5th International Workshop

[1] http://www.w3.org/TR/rdf-sparql-query/

on Web APIs and Service Mashups - Mashups'11. p. 1. ACM Press, New York, New York, USA (2011).
3. Pietschmann, S., Waltsgott, J., Meißner, K.: A Thin-Server Runtime Platform for Composite Web Applications. In: Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services. ICIW'10, Washington, DC, USA, pp. 390-395. IEEE (2010).
4. Demeure, A., Sottet, J.S., Calvary, G., Coutaz, J., Ganneau, V., Vanderdonckt, J.: The 4C Reference Model for Distributed User Interfaces. In: Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08), Washington, DC, USA. pp. 61-69. IEEE (2008).
5. Manca, M., Paternò, F.: Flexible support for distributing user interfaces across multiple devices. Proceedings of the 9th ACM SIGCHI Italian Chapter International Conference on Computer-Human Interaction Facing Complexity - CHItaly. p. 191. ACM Press, New York, New York, USA (2011).
6. Blumendorf, M., Roscher, D., Albayrak, S.: Dynamic user interface distribution for flexible multimodal interaction. International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction on - ICMI-MLMI'10. p. 1. ACM Press, New York, New York, USA (2010).
7. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R.M.: Four easy pieces for assesing the usability of multimodal interaction: the CARE properties, In Nordby, K., In Nordby, K., Helmersen, P.H., Gilmore, D.J., Arnesen, S.A., eds.: INTERACT. IFIP Conference Proceedings, Chapman & Hall (1995) 115–120
8. Togias, K., Goumopoulos, C., Kameas, A.: Ontology-Based Representation of UPnP Devices and Services for Dynamic Context-Aware Ubiquitous Computing Applications. 2010 Third International Conference on Communication Theory, Reliability, and Quality of Service. pp. 220-225. IEEE (2010).
9. UPnP Forum: UPnP Device Architecture, http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf (2008).
10. Nickelsen, A., Paternó, F., Grasselli, A., Schmidt, K.-U., Martin, M., Schindler, B., Mureddu, F.: Open: open pervasive environments for migratory interactive services. In: Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services - iiWAS'10. p. 639. ACM Press, New York, New York, USA (2010).
11. Ketfi, A., Belkhatir, N., Cunin, P.-Y.: Automatic Adaptation of Component-based Software: Issues and Experiences. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications - Volume 3. PDPTA '02, CSREA Press (2002) 1365–1371 1365-1371 (2002).