C. Stocklöw, A. Grguric, T. Dutz, T. Vandommele, A. Kuijper: **Resource Management for Multimodal and Multilingual Adaptation of User Interfaces in Ambient Assisted Living Environments.** In: C. Stephanidis, M. Antona: Universal Access in Human-Computer Interaction. Applications and Services for Quality of Life - 7th International Conference, UAHCI 2013, Held as Part of HCI International 2013, Springer, July 2013. ISBN 978-3-642-39193-4.

and Multilingual Adaptation of User Interfaces in Ambient Assisted Living Environments

Carsten Stocklöw[1], Andrej Grguric[2], Tim Dutz[1], Tjark Vandommele, and Arjan Kuijper[1]

[1] Fraunhofer Institute for Computer Graphics Research, Fraunhoferstr. 5, 64283 Darmstadt, Germany
[2] Research and Innovations Unit, Ericsson Nikola Tesla d.d., Krapinska 45, 10002 Zagreb, Croatia
{carsten.stockloew,tim.dutz,arjan.kuijper}@igd.fraunhofer.de, andrej.grguric@ericsson.com, tjark_vandommele@gmx.de

Abstract. Providing multimodal user interfaces in Ambient Assisted Living scenarios is a challenging task due to large variety of modalities and languages that can be used as well as impairments and preferences of end users. Creating an application that can cope with this multitude of presentation possibilities is highly complex. However, by separating the application from the presentation layer and representing the dialog in an abstract form, it is possible to perform adaptations according to the output parameters. In this work, we present the concept for a Resource Server for multimodal and distributed systems which is capable of storing different kinds of resources and associated metadata, and adapting abstract dialogs. We propose the introduction of a presentation identifier as placeholder for a set of concrete resources, a two-stage mapping between identifiers, and a selection algorithm to cope with the problem of multiple matching resources.

Keywords: Ambient Assisted Living, User Interaction, Resource Server.

1 Introduction

Ambient Assisted Living (AAL) comprises methods, technologies, products, and attendances applied to improve the quality of life for people of all ages. Considering predictions of the demographic changes in western societies, AAL particularly focuses on elderly and differently abled people. These user groups often suffer from a large variety of impairments, which makes it necessary to provide highly adaptive user interfaces [1]. By separating the application from the presentation layer and describing the user dialogs in an abstract and modality-independent way, it is possible to perform adaptations in accordance with a user's needs and preferences. By allowing descriptions of user interfaces in declarative and abstract terms much flexibility is gained, which especially becomes useful in multimodal systems. In such a scenario, the modality with which a dialog is

C. Stephanidis and M. Antona (Eds.): UAHCI/HCII 2013, Part III, LNCS 8011, pp. 97–106, 2013.

presented is not chosen by a specific application, but by an underlying system that has access to information about the user's profile and about the system's contextual state. A presentation of this dialog is then created by the system according to the selected modality. Thus, the developer of an application can entirely focus on the application's functionality and does not need to devote energy to the problem of how to handle different modalities.

However, this approach has one major drawback. The resources (such as images, texts, audio files, etc.) needed to create a presentation of an abstract dialog have to be provided by the application. As a software developer for the AAL domain, it is almost impossible to predict the available devices and the required modalities upfront. Therefore, the developer has two options. Firstly, he could offer large amounts of different resources and try to cover each modality, device, and language. Or secondly, he could focus on covering only a few, specific use cases and only provide the necessary resources for those; thereby reducing the addressed group of end users that can use the application. Both options are not suited for AAL environments, because they either require too much work on the developer's site, or might render the application practically useless.

As a consequence, we propose the introduction of a *Resource Server* for managing the individual resources of user interfaces in AAL systems. This new component is capable of storing different kinds of resources and associated metadata, integrating additional resources, and providing resources to components of the system. Furthermore, we propose the idea of using unique identifiers (*Uniform Resource Identifiers*, URIs) for elements in the abstract dialog description for referencing a set of resources independently of specific modalities. This identifier (*presentation-URI*) is used to adapt the dialog according to the concrete modality, possible access restrictions and preferences of the end user. A two-stage mapping is used to map the identifier to a set of concrete resources to provide the possibility to map the identifiers themselves in case that two or more different identifiers are used by different applications to describe the exact same resource. This way, our approach also provides the option to reuse resources of other applications easily in order to further simplify the development of new products. The mapping can be changed and extended at any time even while the system is running - the changes applied will then be taken into account for all subsequent dialogs.

Furthermore, we incorporate a selection algorithm to cope with the problem of more than one matching resource. This algorithm uses different classes of restrictions to compare them to the available resources for each presentation identifier and selects the most suitable of them. With those restrictions, it is possible to map the situation in which a dialog will be presented. Specifically the user's preferences, the chosen modality, the features of the selected output device and the capabilities of the component that creates the dialog's presentation can be modelled by those restrictions.

2 Related Work

The idea of a resource server for intelligent, multimodal and distributed systems was also proposed by Zimmermann and Wassermann and was implemented by the *URC-Consortium*[1] [2]. They define a resource server as "*a platform for the storage, processing and retrieval of information related to user interfaces for service environments*" [3]. To access the resources the software developer can either use a web front-end or a HTTP-Protocol, which was specifically created for this system [4]. However, this system is focused on the provision of complete user interfaces which limits the re-usability of individual resources.

Ponnekanti et al. [5] propose a system called *iCrafter* to automate the creation of user interfaces for distributed systems. For that purpose they use specialized *Interface-Generators*, that are capable of creating an interface for a previously defined functionality. In order to select which generators are used, the user has to specify the funtionalities that will be used. This is done with the help of a designated device that runs an *Interface-Manager*, which has knowledge of the available functionalities. Therefore, this Interface-Manager is the link between the hardware the user wants to use and the functionalities of the system. When the user has chosen the desired services, suitable Interface-Generators are selected and the corresponding interfaces are created. In that process a *Generator-Database* is used by the generators to obtain information about the functionalities, like the positions of the respective hardware.

The *Personal Home Server*, discussed by Nakajima et al. [6], is another concept for creating personalized user interfaces autonomously. This approach uses a beacon, such as a mobile phone or a special watch, to store information about user interaction related preferences. The user has to carry this beacon so that it can send information about the user's preferences to any device in the vicinity that might use a user interface. This way, the interface can be personalized for the user. Using this system as a resource server, however, is not feasible. As this system is decentralized, it is costly to integrate new resources into the system. Furthermore, because of the use of wireless technology, bandwidth shortages might occur. This can slow the system down if it is used in an environment that contains many output devices, such as an AAL-system.

3 Concepts

3.1 Architecture

Figure 1 shows the overall architecture of the system. By separating the application layer from the presentation layer and representing the dialog in an abstract form it is possible to provide adaptions to the dialog. The idea for such a separation is not new, the most prominent example is the World Wide Web which uses a webserver as UI Provider, a browser as a UI Handler, and HTML as format

[1] `http://www.myurc.org`

for dialogs. The Resource Server can then be integrated in this system as a component that can be called from the UI Handler because only the UI Handler has knowledge about the concrete capabilities of the output device. In the following sections we will describe the different components of this architecture and section 4 will detail the integration of the Resource Server in an AAL platform.

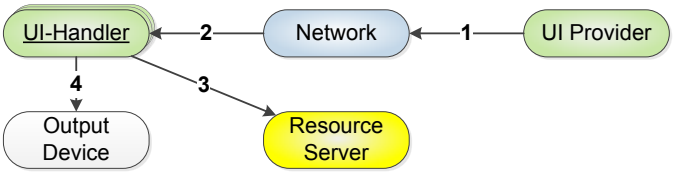

Fig. 1. Architecture

3.2 Presentation-URI

In a system without a Resource Server, the resources needed for a user dialog are being provided by its corresponding application. When developing such an application the programmer specifies which resource to use for which element of the dialog in the abstract description of that dialog. As these resources are not abstract but concrete, the dialog description is not truly abstract itself. As a consequence, such a dialog can only be used for output devices that support all the used resources, which greatly reduces its flexibility. Moreover, in a distributed system the software developer has no information about the location of the resources once they are managed by the Resource Server.

For these reasons we propose the usage of a **presentation-URI**. Such a presentation-URI is the abstract description of a resource which is associated with a set of concrete resources. This set contains resources that all represent the same data in different ways. For example there could be an audio file in which a person says 'sunny', several text files with the word 'sunny' in different languages or multiple image files of different resolutions with a sun and even images containing text in different languages.

When developing an application for a system with a Resource Server the developer uses the presentation-URIs in the abstract dialog description instead of concrete resources. As no concrete elements are left in the dialog description it is truly abstract and can be used for any output device. Which concrete resource is used to present a dialog has to be determined by the Resource Server, which uses information about the user, the chosen output device and the current context to find the most suitable resource for each presentation-URI.

In order to provide for an easy reuse of resources provided by other applications, the presentation-URIs can also be associated to one another. This way multiple applications that use the exact same resources but address them with different presentation-URIs can access the same resources. Therefore, it is sufficient to store each resource once and maintenance of the resources is simplified.

The mapping of resources to presentation-URIs as well as the mapping of presentation-URIs to each other can be changed and extended at any time while the system is running. These changes will then be taken into account for all subsequent use of the resources.

3.3 Server Component

The core of the Resource Server is its **Server Component**. It stores and manages the resources of all applications and distributes them to the elements of the system that intends to use them. For that purpose it receives requests by the system which it answers by sending either information about resources or the resources themselves. The requirements for the server component are:

Storing Resources: This is the core functionality of the server component. As it is not possible to predict which kind of resources will be used by the applications, the Resource Server must be able to store resources in all common formats as well as newly defined file types. Furthermore, the Resource Server has to prevent resources from accidentally being altered by other components of the system.

Storing Metadata: In addition to the resources themselves, the Resource Server also needs to store metadata that describe the contents of the resources. This metadata is important for determining which resource to use if there are multiple resources from the same file type for one presentation-URI. Due to the fact that it is not known which property distinguishes two resources of the same set, the system has to be designed in a way that allows the developer to define new metadata categories. However, for compatibility reasons some common categories like 'lang' for language should be defined by default.

Searching for Resources: In order to provide a maximum flexibility in the usage of the resources, the server component has to be able to search for resources. In other words it has to be possible to find a resource not by addressing it with its presentation-URI, but by defining a set of properties and filtering the metadata of all resources for those properties.

Aggregating Resources: For the purpose of reducing the delay caused by transmitting the resources to the different elements of the system, the Resource Server should be able to aggregate multiple resources before they are sent. Especially when many small resources like text snippets for buttons, labels etc. are requested, it is inefficient to deliver each resource separately. Preferably, the system would request all those resources at once and the resource component would aggregate them either in one file or with the help of an archive file. While combining several resources in a single file is simple and intuitive, it only works if all resources are of the same type and if that file type can be aggregated at all. More flexible is the use of an archive file, which may contain all kinds of resources. On the other hand, that approach needs some additional computing of the server component as well as the component that requests the resources. Therefore, there can be cases in which aggregating resources cannot reduce the transmission delay.

There are several server technologies that can be used to fulfill the requirements mentioned above. The easiest way would be to use the **file system** of the underlying operating system. The resources would be stored in a specific folder and for each resource a text file would be created which would contain the metadata of that resource. However, when searching resources by their properties, each of those text files would have to be opened and analyzed which is very inefficient. A better approach would be to use a **relational database** to store the metadata. In that case, a database with three tables would be created to provide a maximum of efficiency when searching for resources.

The first table, called *resources*, contains mandatory information about every resource stored on the server component. The primary key, which is needed in every table of a relational database, would be an automatically generated resource ID (*R_ID*). Furthermore, there are fields that save which application stored the resource (*bundle*), its file name (*filename*), the corresponding file extension (*fileextension*), the path in the file system where the resource is stored (*path*) and its presentation-URI (*uri*).

The second table contains the metadata of all resources and therefore is called *metadata*. In this case the primary key is a combination of the field's resource ID (*R_ID*) to specify which resource this property belongs to, the category of the property (*category*) and a field called (*value_no*). The third component of the key is necessary because it is possible that a category of metadata has several values simultaneously. The last field of this table is called *value* and contains the actual information about the resource.

If any resource is used by more than one application the different presentation-URIs for that resource are associated with each other, as mentioned above. The presentation-URI which is already connected to the resource will become the 'master'-URI while the other will be called 'slave'-URI. This information will then be stored in a table called *URI-mapping*, which consists only of the fields *slave* (primary key) and *master*.

Figure 2 shows an example of a resource and its metadata stored on the server component.

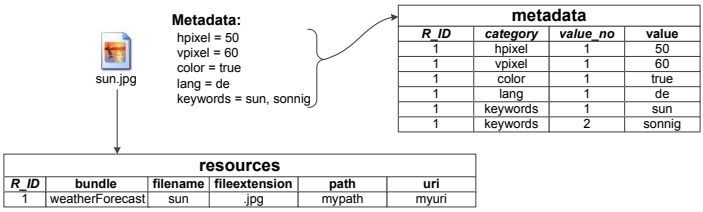

Fig. 2. Storing metadata of a resource in a relational database

3.4 Client Component

Although multiple servers are possible, the typical realization of this scenario contains only one server in an intelligent environment. If this environment is designed as an open distributed system then potentially multiple nodes can

access the functionalities of the server. To simplify the integration of different UI Handlers with the Resource Server, a client component can be integrated into the system which takes care of mapping the presentation-URIs to concrete URLs used by the UI Handler. It ensures the connection between the system and the server component and mediates the requests and responses between them. Thus, only little changes to the UI Handler have to be performed to include functionalities of the Resource Server.

Another benefit of this approach is that it is possible to use specialized client components. For example, if a component of the system is written in a different programming language, a client component that is capable of processing requests from this component can be used. This way access to the Resource Server is flexible and it can be used by every component in the system.

3.5 Selection Mechanism

In most cases each presentation-URI will be associated with more than one resource, which yields the necessity of a selection mechanism. Although other approaches are possible, this selection will take place in the client component, as mentioned above. This is due to the flexibility in the usage of specialized client components, which can be exploited to implement different selection strategies.

We propose a universal selection mechanism to select resources for a user dialog, which can be altered to meet the needs of different scenarios. This mechanism uses information about the current context, the capabilities of the UI-Handler and the output device to create restrictions. Additional restrictions may be supplied by the application that wants to display the dialog. Those restrictions are then compared with the metadata of each resource to determine if it is suitable for the dialog.

The most important information is which types of resources can be processed by the UI-Handler and its output device. This constraint is heavily dependent on the modality that was chosen for the dialog but can also be influenced by the UI-Handler and the output device. For example, if the dialog shall be presented visually, all acoustic resources can be ignored. In addition to that the UI-Handler might not be able to render HTML files, thus those can be omitted as well. Therefore, the UI-Handler has to submit information about which MIME-types can be processed (*whitelisted*) and which cannot (*blacklisted*). Due to the importance of this information, the server component uses the file extension of each resource to determine its *MIME-type* automatically and saves it as a property in the metadata table. This ensures that the MIME-Type is set for every resource and can be used for the selection.

To ensure an efficient comparison of the resources and the restrictions, the latter should be divided into three classes:

1. **Must have:** Those restrictions define properties that a resource has to contain in order to be selected. Otherwise, the resource will be ignored.
2. **Must not have:** Properties that are described in this class are not allowed to be present in the metadata of a resource.

3. **Nice to have:** Restrictions in this class do not reduce the number of candidates for a presentation-URI. Instead, they are used to rate the remaining resources if more than one meets the restrictions of the other classes. In that case a resource is awarded one point for each nice-to-have-restriction it fulfills. Afterwards the resource with the highest score is chosen unless there is more than one resource with a top rating, in which case one of those is randomly picked.

Although it is possible to select a resource with those three classes of restrictions, this system is very inflexible and cannot cope with the variety of properties that can be stored in the metadata of the resources. Therefore, an additional attribute that describes the type of a restriction has to be applied. Possible attributes are:

- **if set:** This attribute states that the restriction shall only be evaluated for resources that contain a value for the corresponding property. For example, a resource might not contain any information regarding its language, because it is an image that does not contain words in any language. Naturally, such a resource should still be selectable even if a restriction is given that requires the language to be English.
- **in range:** By using this attribute it is possible to define a restriction that does not expect a specific value of a property, but allows the value to be in an interval. This can be used to select images with a resolution in an acceptable range. In addition to the interval boundaries, information whether those boundaries are inside or outside the interval have to be given.
- **x of:** This attribute can be used if a set of properties is given and it does not matter which of those are met, as long as a certain amount is. For example, it is possible to ask for resources that contain three of five keywords.
- **no attribute:** Restrictions that do not contain an attribute are still possible and are evaluated as discussed above.

Further attributes can be defined and are particularly useful for more specialized selection mechanisms. Consequently those mechanisms can adopt to the diversity of different resources.

4 Implementation

The concepts described in this work have been applied to the open AAL platform *universAAL* [7] which is supposed to become a standardized general-purpose platform for AAL environments. The universAAL platform is a consolidated combination of prior work, not following a completely new approach but rather integrating approved concepts from a variety of projects in this area. By default, this platform uses an OSGi container (although different containers are possible), making it easy to integrate additional modules such as the resource client.

Figure 3 illustrates how the Resource Server is, in general, used to provide resources for an output dialog sent by an application.

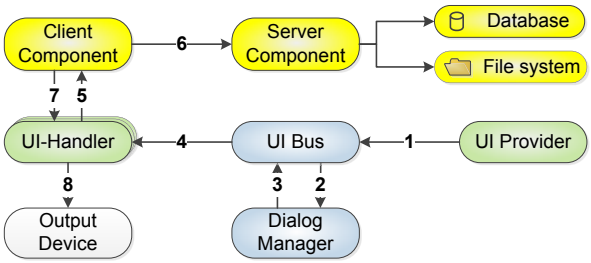

Fig. 3. Transmission of a dialog from application to output device

1. The application forwards an abstract dialog to the UI-Bus. Since the system contains a Resource Server, no concrete resource is used in this dialog. Instead, the corresponding presentation-URIs are used.
2. The UI-Bus transmits the dialog to the Dialog Manager. This component adds information about the current context to the dialog and assures that no other dialog has higher priority.
3. As soon as the Dialog Manager releases the dialog for presentation, it is returned to the UI-Bus. Using the attached context information, the UI-Bus decides which UI-Handler will create the output.
4. The UI-Bus sends the dialog including the context to the chosen UI-Handler.
5. The UI-Handler realizes that the dialog contains one or more presentation-URIs and thus forwards it and the context to the client component. Additionally the UI-Handler sends information about the capabilities of itself and its associated output device to the client component.
6. The client component needs four steps to process the request from the UI-Handler:
   (a) First it extracts the presentation-URIs contained in the dialog.
   (b) Then it requests the available information about all resources associated with those presentation-URIs from the server component, using a HTTP-request.
   (c) Afterwards, if any of the presentation-URIs is associated with more than one resource, the client component uses the information about the context and the UI-Handler to choose the most suitable resource.
   (d) Lastly it integrates the path of each selected resource into the dialog.
7. The client component submits the altered dialog to the UI-Handler, which creates a presentation of the dialog.
8. This dialog is sent to the output device, where it is presented to the user.

5 Future Work

Security: While the operating system can provide some security against attacks from outside of the system, no protection is provided against malicous applications that run inside the system itself. These applications can access or alter

any resources or metadata, which can result in various problems. Accordingly, a security protocol with access rights and encryption should be designed for the Resource Server.

Compression: The aggregation of resources, as discussed in chapter 3.3, has neither been implemented nor tested yet. This is due to the fact that this concept requires a major redesign of the components that use resources. In addition to those changes it has to be evaluated under which circumstances the compression of resources results in a reduction of the transmission delay.

Web-Frontend: For easy administration of the resources and their metadata a web-frontend for the server component should be developed. As the nature of such a component enables it to influence many aspects of a system, special attention should be paid to its security and privacy features.

Acknowledgements. This work is partially financed by the European Commission under the FP7 IST Project *universAAL* (grant agreement FP7-247950).

References

1. Hawthorn, D.: Possible implications of aging for interface designers. Interacting with Computers 12(5), 507–528 (2000)
2. Zimmermann, G.: URC Technical Primer 1.0 (DRAFT). Technical report, Universal Remote Console Consortium (2008)
3. Zimmermann, G., Wassermann, B.: Why We Need a User Interface Resource Server for Intelligent Environments. In: Schneider, M., Kröner, A., Alvarado, J.C.E., Higuera, A.G., Augusto, J.C. (eds.) Workshops Proceedings of the 5th International Conference on Intelligent Environments, pp. 209–216. IOS Press (2009)
4. Zimmermann, G.: Resource Server HTTP Interface 1.0 (DRAFT). Technical report, Universal Remote Console Consortium (2009)
5. Ponnekanti, S.R., Lee, B., Fox, A., Hanrahan, P., Winograd, T.: ICrafter: A Service Framework for Ubiquitous Computing Environments. In: Abowd, G., Brumitt, B., Shafer, S. (eds.) UbiComp 2001. LNCS, vol. 2201, pp. 56–75. Springer, Heidelberg (2001)
6. Nakajima, T., Satoh, I.: A software infrastructure for supporting spontaneous and personalized interaction in home computing environments. Personal and Ubiquitous Computing 10(6), 379–391 (2005)
7. Furfari, F., Tazari, M.R., Eisemberg, V.: universaal: an open platform and reference specification for building aal systems. ERCIM News (2011)