

Flexible Support for Distributing User Interfaces Across Multiple Devices

Marco Manca, Fabio Paternò

CNR-ISTI, HIIS Laboratory
Via Moruzzi 1, 56124 Pisa, Italy
{marco.manca,fabio.paterno}@isti.cnr.it

ABSTRACT

In this paper, we describe a solution to obtain flexible user interface distribution across multiple devices, even supporting different modalities. For this purpose we extend a model-based user interface language in order to address the specification of distribution at various user interface granularities. We also introduce how this solution works at run-time in order to support dynamic distribution of user interface elements across various devices.

Categories and Subject Descriptors

H.5 Information Interfaces And Presentation; H.5.2 User Interfaces

General Terms

Algorithms, Design, Human Factors, Languages,

Keywords

Distributed user interfaces, multi-device environments, model-based approaches, user interface software and technology.

1. INTRODUCTION

The current technological trends are determining a steadily increasing number of computers per person along with many sensors able to detect a wide variety of contextual events. The computers are becoming more and more variegated in terms of possible interaction resources and modalities, including interconnected embedded devices composed of small electronic components, which can interact with each other.

This implies that in the near future we will no longer access our applications through one device at a given time but we will rather use sets of collaborating devices available while moving, such as using the smartphone to control the content on a large screen. Distributed User Interfaces (DUIs) have recently become a new field of research and development in Human-Computer Interaction (HCI). The DUIs have brought about drastic changes affecting the way interactive systems are conceived. DUIs go beyond the vision that user interfaces are controlled by a single end user on the same computing platform in the same environment. Unlike existing user interfaces, DUIs enable end

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CHItaly 2011, September 13–16, 2011, Alghero, Italy.
Copyright 2011 ACM 978-1-4503-0876-2/11/09... \$10.00.

users to distribute any user interface element, from the largest one to the smallest one, across different computing platforms and physical environments [9]. Thus, emerging ubiquitous environments need Distributed User Interfaces, which are interfaces whose different parts can be distributed in time and space on different monitors, devices, and computing platforms, depending on several parameters expressing the context of use [3]. This has an impact on the user interface languages and technologies because they should be able to support the main concepts characterising interactions with an application through various combinations of multiple devices.

Model-based approaches have been considered in order to manage the increasing complexity derived from managing user interfaces in multi-device environments, since each device has specific interaction resources and implementation languages to execute such user interfaces. They are also currently under consideration for W3C for standardization purposes [4]. The basic idea is to provide a universal small conceptual vocabulary to support user interface design, which can then be refined into a variety of implementation languages with the support of automatic transformations without requiring developers to learn all the details of such implementation languages.

Some research effort to address distributed user interfaces with model-based approaches has already been carried out but with limited results and not able to support the many possible ways to distribute user interface elements. In HUID (High Level UI Description) [7] the user interface has a hierarchical structure and the leaves of the tree are Abstract Interactor Object (AIOs) describing high-level interactors. During the rendering process the AIOs are mapped onto Concrete Interaction Object (CIOs) associated with the current platform. In addition, they introduce a split concept for the groupings through an attribute that, when it is set to true, allows the distribution of the user interface elements without losing its logical structure. In our case we propose a different solution, still using a model-based approach. One difference is that we support the specification at the concrete level because at this level it is easier to generate the corresponding implementations and there is a better understanding of the actual effects that can be obtained. Vanderdonckt and others [8] have developed a set of primitives to manage user interface distribution but they only consider graphical user interfaces while our approach is able to support user interfaces exploiting also other modalities, such as voice. Blumendorf and others [1] address multimodal interfaces but they lack an underlying language able to support user interface distribution.

To overcome the limitations of previous work our starting point was the MARIA language [6], which in current version consists in a set of languages: one for abstract user interface description, and a set of concrete refinements of such language for various target platforms (Vocal, Desktop, Smartphone with touch, Mobile, Multimodal desktop, Multimodal mobile). Then, user interfaces generators for various implementation languages (XHTML, SMIL, VoiceXML, X+V, HTML 5) are available starting with such concrete languages. Tools for authoring user interfaces in MARIA and for reverse engineering Web pages into MARIA specifications are publicly available at <http://giove.isti.cnr.it/Tools/>

We have extended such language in order to be able to specify distributed user interfaces and we have also designed a solution to generate implementations of such distributed user interfaces, which can dynamically change how the user interface elements are distributed according to user requests or other events.

In the paper we first provide an overview of the solution that we have developed. Next, we provide some detail on the language supporting it. We discuss the corresponding architectural solution for providing support at run-time and show an example application. Lastly, we draw some conclusions and provide indications for future work.

2. THE APPROACH

The approach proposed has been developed aiming to satisfy two main requirements:

- flexible support able to address a wide variety of granularities in terms of user interface components to distribute;
- small and simple set of primitives to indicate how to perform the distribution.

Regarding the set of primitives we decided to use the CARE (Complementarity, Assignment, Redundancy, and Equivalence) properties [2], which were introduced to describe multimodal user interfaces, and have already been considered in the MARIA concrete language for multimodal interfaces [5]. In our case the idea is to use them with this meaning:

- *Complementarity*: the considered part of the interface is partly supported by one device and partly by another one
- *Assignment*: the considered part of the interface is supported by one assigned device
- *Redundancy*: the considered part of the interface is supported by both devices
- *Equivalence*: the considered part of the interface is supported by either one device or another.

Regarding the possible granularity levels to address we have started from the consideration that in MARIA a user interface is composed of presentations (in graphical interfaces they correspond to the set of elements that can be perceived at a given time, e.g. a Web page). Then, in each presentation there can be a combination of user interface elements and instances of composition operators. In MARIA there are three types of

composition operators: grouping (a set of elements logically related to each other), relation, a relation between groups of elements (e.g. in a form there usually are a set of interactive elements and a set of associated control elements to send or clear them), repeater (a group of elements that are repeated multiple times). Since we aim to obtain full control on what can be distributed we decided to consider also the possibility of distributing the elements within a single interaction element. For example, a text edit interactor can be distributed in such a way that the user enters the text in one device but receives feedback on what has actually been input in another device. For this purpose we provide the possibility to decompose interactive interface elements into three subparts: prompt, input, and feedback.

Then, by combining the set of four possible granularity levels (presentations, compositions, interface elements, interactive subparts) with the CARE properties we obtain a simple and powerful tool to indicate how the user interface can be distributed in a flexible way. Thus, we can distribute an entire presentation. For example, by associating the Redundancy property we indicate that one presentation should be completely rendered in two different devices. However, we can also distribute single interface elements. For example, distributing a textual object in a complementary way means that part of the text is rendered through one device and part through another one. As we mentioned, it is even possible to distribute sub-elements of a single interaction object. For example, a single selection object can be distributed in such a way that when the user selects one element then the feedback indicating what has been selected is rendered through another device. This means that the prompt and input components have been assigned to one device while the feedback sub-component to another one.

It is worth pointing out that the decomposition into prompt, input and feedback is meaningful only for interactive interface element, and cannot be applied for only-output elements.

In this way it is also possible to easily indicate how dynamically the user interface elements can be distributed. Thus, if we want to move one element from one device to another then it means that we have changed the device to which that element is assigned. While if we want to copy a user interface element from one device to another then it means that we have changed the corresponding CARE property from Assignment to Redundancy.

3. THE LANGUAGE

In order to formalise the concepts introduced in a language that can be used to design and generate the corresponding user interfaces we have extended the MARIA language.

In particular, we have introduced a language with the possibility of defining a concrete distributed user interface. In such language it is possible to indicate the types of devices on which the user interface can be distributed. Each device belongs to a platform for which already exists a corresponding concrete language. Such concrete languages refine the abstract vocabulary taking into account the interaction resources that characterise the corresponding platform. This allows designers to specify interface elements that better adapt to the devices in which they are rendered.

The user interface is hierarchically structured: the user interface is composed of presentations. Each presentation is composed of a combination of interface elements and composition elements, which can be recursively composed of interface and composition elements. When a CARE property is associated to one element of

this hierarchy then all the underlying elements inherit such association. Thus, if a grouping of elements is assigned to a device then all the user interface elements of the group will be assigned to it. This also simplifies the specification process by avoiding the need to indicate the value of the CARE properties to all the interface elements.

Below we can see an excerpt from an example of MARIA specification of a distributed user interface. We consider a grouping of interactor elements. The corresponding CARE property is complementarity, which means that the interface elements are distributed across various devices. In particular, there are two interactors (a video and a text) and four devices. For each device it is specified the corresponding platform, in this case we have one desktop, one vocal, one mobile, and one multimodal.

```
<grouping>
  <output care_value="complementarity">
    <bind>
      <interactor interactor_id="description_video"/>
      <device id="paterno" platform="desktop"/>
    </bind>
    <bind>
      <interactor interactor_id="description_text"/>
      <device id="sist" platform="vocal"/>
      <device id="iphone_lab" platform="mobile"/>
      <device id="manca" platform="multimodal"/>
    </bind>
  </output>
```

Afterwards we have the specification of the two involved interactors. Since the text is complementary over three devices, the specific attributes for each of them can be specified. Actually, since in one case one device is multimodal, we can again apply the CARE properties to indicate how the information is distributed across the two modalities of the same device. In the example, it is complementary again.

```
<description id="description_video">
  <description_desktop>
    <video src="video.flv" alt="alternative_text"/>
  </description_desktop>
</description>

<description id="description_text">
  <!--COMPLEMENTARY DISTRIBUTION -->
  <description_mobile>
    <text><string>Mobile Text </string></text>
  </description_mobile>
  <description_vocal>
    <speech><content>VocalText </content></speech>
  </description_vocal>
  <description_multimodal output="complementary">
    <!-- [graphical part] -->
    <text><string>Mobile Text</string></text>
    <!-- [vocal part] -->
    <speech><content>Vocal Text</content></speech>
  </description_multimodal>
</description>
</grouping>
```

A dynamic change of the distribution of the user interface elements is supported by adding a distribution event in the dialogue model in the MARIA specification. The dialogue model is composed of a number of event handlers and indicate the temporal relation among them. The distribution event can be triggered by a user action and the event handler indicates what user interface elements and what devices are involved by changing the corresponding CARE attributes.

In the following we can see an example of such events. It is generated by a button that when pressed activates a distribution of the input, prompt, and feedback components of one interactor in such a way that the input can be entered either through a desktop or a vocal device, the prompt is complementary across such two devices, and the feedback is assigned to only the desktop device.

```
<activator>
  <button><label>Distribute UI</label></button>
  <event>
    <distribution_event>
      <handler>
        <change_property interactor_id="multiple_id"
          phase="prompt" care_value="COMPLEMENTARITY">
          <device id="manca_pc" platform="desktop"/>
          <device id="sist_pc" platform="vocal">
        </change_property>
        <change_property interactor_id="multiple_id"
          phase="input" care_value="EQUIVALENT">
          <device id="manca_pc" platform="desktop"/>
          <device id="sist_pc" platform="vocal">
        </change_property>
        <change_property interactor_id="multiple_id"
          phase="feedback" care_value="ASSIGNMENT">
          <device id="manca_pc" platform="desktop"/>
        </change_property>
      </handler>
    </distribution_event>
  </event>
</activator>
```

4. AN EXAMPLE APPLICATION

In order to show how our approach works let us consider a concrete example, not too complex for sake of clarity. We consider an application to show slides, it allows users to go back and forth, and to annotate them. Figure 1 shows the initial user interface, completely rendered in a desktop graphical device.

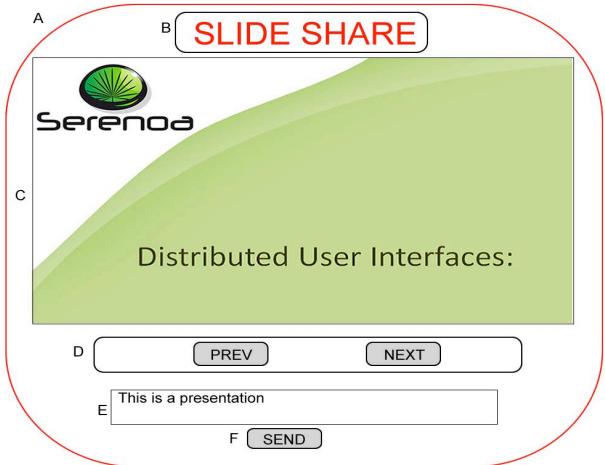


Figure 1. The Slide Share application.

More precisely Figure 2 shows the corresponding hierarchical structure: one presentation with a couple of output descriptive objects (the application title and the slide), a grouping composing two buttons to go back and forth, an interactive elements to write comments and a button to store them. Since the interface is completely shown in one single device, it is sufficient to associate the assignment property to the root.



Figure 2. The Structure of the Example.

Now, suppose that the user wants to distribute parts of the user interface to a multimodal mobile device as indicated in Figure 3.

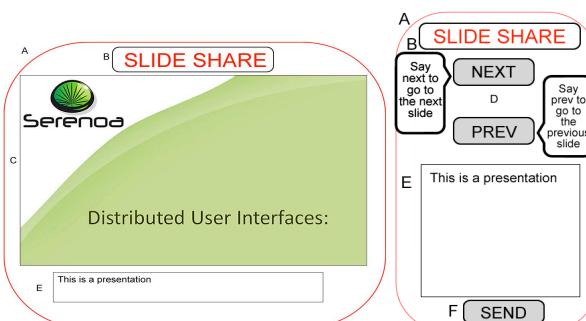


Figure 3. The Slide Share Distributed.

To obtain this example of distributed user interface a number of elements have been assigned new values of the CARE attribute as indicated by Figure 4. Thus, there is no longer a CARE attribute assigned at the presentation level. The title description is redundant while the slide description is assigned to the desktop device, because it has a larger screen that can better show its content. The grouping with the buttons for going forth and back is

assigned to the mobile device and it has a multimodal support: prompt is redundant with vocal and graphical modality, and input is equivalent and can be provided by either modality. The text edit interactor for entering comments is redundant in both devices but the button to store the comments is assigned only to the mobile device, for immediate activation by the user.

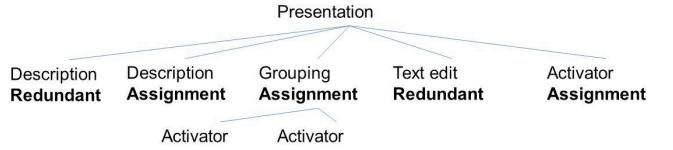


Figure 4. The Updated CARE Attributes.

5. RUN-TIME SUPPORT

Once the distributed user interface has been designed there is the issue of obtaining the corresponding implementation able to support the desired dynamic behavior. There are two possible ways to do this:

- Dynamically modify the CARE properties in the MARIA logical description and then regenerate the corresponding updated user interfaces;
- Initially generate all the possible user interface elements but initially presenting only those that should be perceivable according to the CARE properties, and when an event changes the value of such properties, change the perceptibility of the user interface elements accordingly.

We opted for the second solution because it is more efficient since it does not require generation of all the distributed user interface elements for each change in the CARE property. For this purpose the specification of the CARE properties is included in the data structure supporting the data model associated with the user interface.

At first the user interface generator analyses the devices involved in the MARIA specification of the distributed user interface, separates the concrete user interfaces for each of them, and generates the corresponding initial user interfaces, which are available in the associated application server. When a distribution event occurs it is forwarded to a specific component (Data Model and Property List in Figure 5), which updates the CARE properties according to the indications in the associated event handler, and then updates the user interfaces of the devices involved accordingly. For this purpose the Web socket mechanism is used because it provides an efficient way to push the updated content.

Indeed, in the MARIA language it is possible to include a data model to represent the data types handled by the user interface. The user interface elements (interactors) can be associated with data types through the data reference attribute. This dependency implies that at run-time if an element of the data model changes its value then all the associated interactors should be notified of the change in order to update their state accordingly.

The changes in values in the data model are described through event handlers that indicate when they should occur and the new value that should be applied to the data element in question. In addition, such event handlers can also communicate changes in the CARE properties of some user interface components. As

mentioned, in the case of distributed user interfaces, all the user interfaces generated for the various devices involved share the data model, indicating also the values for the CARE properties.

More in detail, Figure 5 shows how the software architecture works at run-time. We assume a case in which the user interface is distributed across a desktop and a mobile device. At the beginning the user accesses the application through the desktop device, the environment creates a data model and property list according to the MARIA specification and provides the requested user interface (3 in Figure 5) and the same process is followed for the mobile device. When the user changes a value of an interactor, this is communicated to the websocket server, which checks the CARE properties of that interface element to see whether changes are necessary and updates the other instances in other devices (9) accordingly.

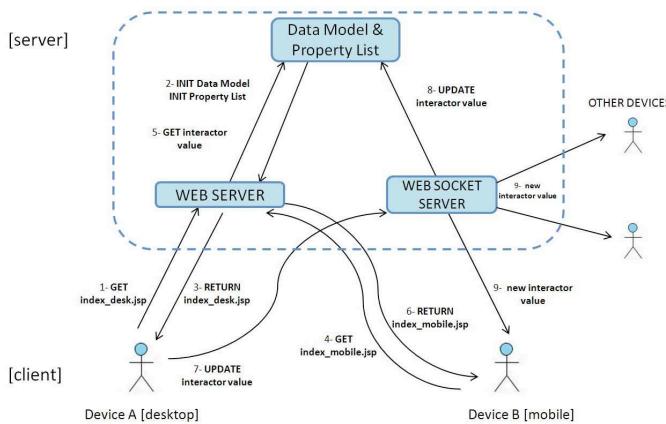


Figure 5. The Run-Time Architecture.

The current prototype environment takes Distributed MARIA specifications and generates JSP implementations, which are dynamic pages able to generate XHTML or HTML5 or VoiceXML or X+V implementations depending on the type of target device.

6. CONCLUSIONS and FUTURE WORK

Distributed user interfaces require novel languages and tools in order to obtain flexible support for user interface designers and developers. In this paper, we have presented an approach able to describe distribution at various granularity levels, even involving multimodal devices. We have also introduced how dynamic distribution can be achieved through such approach and the software architecture supporting the corresponding implementation.

Future work will be dedicated to engineering the supporting environment and to usability evaluation of both the authoring environment and the resulting distributed interactive applications.

ACKNOWLEDGMENTS

This work is supported by the EU ARTEMIS SMARCOS Project, <http://www.smarcos-project.eu/>.

REFERENCES

- [1] Marco Blumendorf, Dirk Roscher, Sahin Albayrak, Dynamic User Interface Distribution for Flexible Multimodal Interaction, Proceedings ICMI-MLMI'10, 8-12, 2010, Beijing, China
- [2] Coutaz J., Nigay L., Salber D., Blandford A., May J., Young R., 1995. Four Easy Pieces for Assessing the Usability of Multimodal Interaction: the CARE Properties. Proceedings INTERACT 1995, pp.115-120.
- [3] Demeure, A., Sotter J.S., Calvary G., Coutaz J., Ganneau V., and Vanderdonckt J.. The 4C Reference Model for Distributed User Interfaces. Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems 2008, pp. 61-69.
- [4] Fonseca J.M.C. (ed.), W3C Model-Based UI XG Final Report, May 2010, available at <http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/>
- [5] M. Manca, F. Paternò. Supporting Multimodality in Service-oriented Model-based Development Environments, Proceedings HCSE 2010, 3rd Conference on Human-Centred Software Engineering, pp.135-148, LNCS 6409 Springer, Reykjavik, October 2010.
- [6] Paternò F., Santoro C., Spano L.D.: MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16(4): (2009).
- [7] Vandervelzen C., Conix K., Towards Model-Based Design Support for Distributed User Interfaces, NordiCHI 2004: 61-70, 2004.
- [8] Vanderdonckt J., A Model-based Approach for Distributed User Interfaces, Proceedings ACM EICS 2011, Pisa.
- [9] Workshop on Distributed User Interfaces 2011. DUI2011 @ CHI2011 - Workshop held at ACM CHI Conference on Human Factors in Computing Systems (Vancouver, BC, Canada, 7-12 May 2011). University of Castilla-La Mancha, Spain; ACM, 2011.