Exploiting Web Services and Model-Based User Interfaces for Multi-device Access to Home Applications

Giulio Mori, Fabio Paternò, and Lucio Davide Spano

ISTI-CNR, HIIS Laboratory, Via Moruzzi 1,
56124 Pisa, Italy
{Giulio.Mori, Fabio.Paterno, Lucio.Davide.Spano}@isti.cnr.it

Abstract. This paper presents a method, and the corresponding software architecture and prototype implementation to generate multi-device user interfaces in the home domain. The approach is based on Web services and model-based user interface generation. In particular, it focuses on multi-device interfaces obtained starting with XML descriptions of home Web services, which are then mapped onto user interface logical descriptions, from which it is possible to then generate user interfaces adapted to the target devices. During use, the generated interfaces are able to communicate with the home Web services and can be dynamically updated to reflect changes in domestic appliances available and the associated state.

Keywords: User Interface Generation, Web Services, Logical Interface Descriptions, Home Applications.

1 Introduction

Our work takes into account current technological trends and research results and aims to provide integrated solutions able to allow users to flexibly access functionality important for their daily life. In particular, the approach is based on three main aspects:

- In recent years, model-based user interface generation has stimulated increasing interest because it can support solutions for multi-device environments exploiting XML logical descriptions and associated transformations for the target devices and implementation languages.
- Web services are increasingly used to support remote access to application functionalities, in particular in ubiquitous environments. They are described using WSDL (Web Services Description Language) files, which are XML-based descriptions as well.
- The home is becoming more and more populated by intelligent devices with the ability to communicate information, thus allowing remote access to their state in order to query or modify it.

The goal of our solution is to allow users to access their domestic appliances from anywhere using any available interactive device. This is obtained by supporting

T.C.N. Graham and P. Palanque (Eds.): DSVIS 2008, LNCS 5136, pp. 181 – 193, 2008.

automatic generation of user interfaces for home applications in such a way as to be able to handle dynamic configurations of home appliances. The resulting environment allows users to dynamically access their home applications involving access to domotic devices such as lights, alarm sensors, media players and so on. We aim to provide dynamic access through multiple interactive devices to multiple functionalities available through Web services (see Figure 1). Regarding the home appliances (such as lights, shutters, air conditioning, video recorders), they can communicate using various types of network protocols. We assume the existence of an intermediate middleware supporting interoperability among such home devices (for example, we have considered the open source environment DomoNet [9]), which provides access to the home devices through Web Services independently of the communications protocols. Thus, the devices can use their original protocol to communicate (examples are UPnP, Konnex, BTicino …) but then such communication goes through a home server, which makes their services accessible to any client through a unifying format. The goal is also to obtain an environment able to support access even when changes in the available home devices occur.

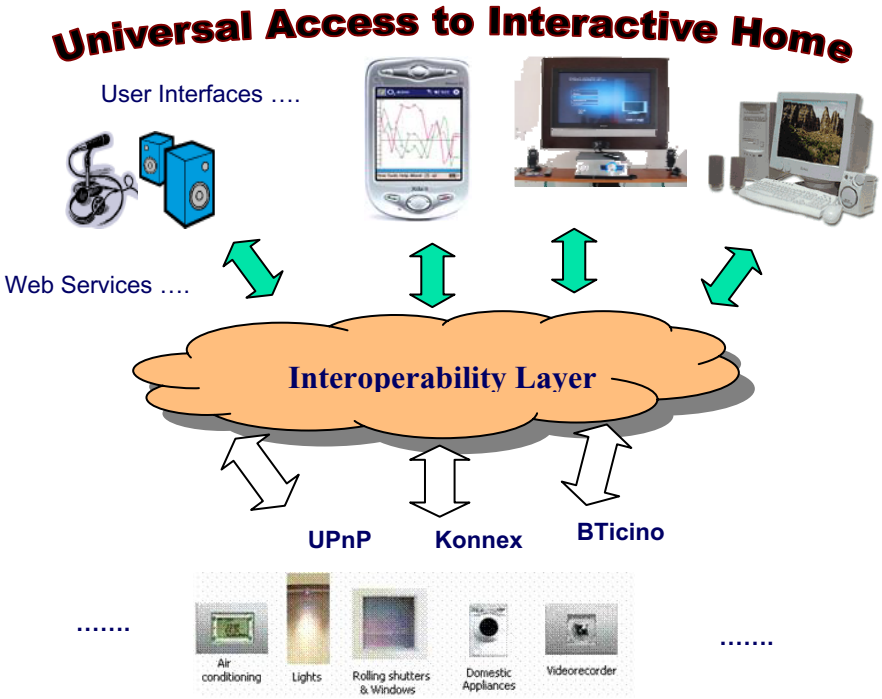

Fig. 1. The Overall Approach

The paper is structured as follows: we first discuss related work, next we provide some background information useful to make the paper self-contained, then we present the overall approach proposed, and show an example application. Lastly, some conclusions are drawn along with indications for future work.

2 Related Work

The increasing availability of interaction device types has raised interest in techniques able to support adaptation of the user interface. In Web applications, the adaptation process can take place in the application server, or in the proxy server or in the client device. Digestor [2] and Power Browser [3] have been solutions that use proxy-based transformations (as in our case) in order to modify the content and structure of Web pages for mobile use. However, they do not use logical descriptions of user interfaces in order to reason about page re-design or apply analysis of the sustainable costs of the target device, as happens in our case. Supple [6] is a tool able to support adaptation by applying intelligent optimization techniques.

One solution that has raised a good deal of interest is the model-based approach in which the logical user interface descriptions are usually represented through XML-based languages (examples are TERESA XML[12], UIML[1], USIXML[7]). In the CAMELEON project [4] a framework describing the various possible abstraction levels was refined based on the experience acquired in this area. A number of tools have been developed aiming to implement such framework (see for example, Multi-modal TERESA [12], ….).

Such logical descriptions have also been exploited in other environments. For example, in Damask [8] they are used along with a sketch editor and the possibility to exploit a number of patterns. PUC [10] is another interesting environment, which uses some logical description but focuses on the automatic generation of consistent user interfaces for domestic appliances (such as printers, copy machines, …). In PUC, logical descriptions of the device to control are downloaded by a mobile device in which the corresponding user interface is automatically generated.

In general, little attention has been paid to the use of user interface model-based approaches for the generation of applications based on Web services. Some work has been dedicated to the generation of user interfaces for Web services [13] [14] but without exploiting model-based approaches to user interfaces. In [15] there is a proposal to extend service descriptions with user interface information. For this purpose the WSDL description is converted to OWL-S format, which is combined with a hierarchical task model and a layout model. We follow a different approach, which aims to support the access to the WSDL without requiring their substantial modifications in order to generate the corresponding user interfaces, still exploiting logical interface descriptions. We aim to address this issue, with particular attention to home applications, which are raising increasing interest given the increasing availability of automatic domestic appliances.

3 Background

In this work we want to investigate solutions for the combined use of Web services and model-based user interfaces. Regarding the description of the logical user interfaces, we have extended TERESA XML [12]. Since this language has already been considered in other papers, herein we just recall the basic concepts in order to make this paper self-contained, highlight the more relevant parts, and indicate its evolution in order to better address the issues raised by this work.

TERESA XML is a set of languages able to describe the various abstraction levels for user interfaces. We consider the levels highlighted by the CAMELEON reference framework [4], which is based on the experiences of the model-based user interface community. There are two platform-independent languages, which means that they are able to describe the relevant concepts for any type of device. They are the language for the task model (which is the ConcurTaskTrees notation [11]) and the language for the abstract user interface description. Then, there is a set of platform-dependent languages, one for the concrete description of each platform considered. We mean for platform a set of devices and associated software environments that share similar interaction capabilities (e.g. form-based graphical desktop, vocal, …). Such concrete languages are implementation-language independent but depend on the interaction modalities associated with the considered platform (examples are: the desktop direct manipulation graphical platform, the form-based graphical mobile platform, the vocal platform, …). Each language part of TERESA XML is associated with an XML Schema. We initially used DTDs for this purpose, but their expressiveness is limited.

The abstract description is composed of presentations and connections indicating how to move from one presentation to another. The presentations can include composition operators and interactors. The composition operators are declarative ways to indicate how to put together groups of interactors, in particularly in order to achieve some communication goal, such as highlighting that a group of elements are semantically related to each other (grouping) or that some elements somehow control another group of elements (relation). Associated with groups of elements it is possible to specify the level of importance of the composing elements or whether there is any specific ordering among them. The interactors are declarative descriptions of ways to present information or interaction objects.

All the concrete description languages share the structure defined by the abstract language and refine it by adding elements indicating how the abstract elements can be better defined for the target platform. Thus, the concrete elements are mainly defined by adding attributes to the abstract elements, while still remaining independent of the implementation language. For example the form-based desktop description language can be used to describe user interfaces implemented in XHTML or Windows Forms or Java Swing.

Our work on the home case study has been useful to identify some of the abstractions missing in previous versions of TERESA XML, such as alarms, ,the possibility to enter numerical values within a range, the possibility to have activators associated with multiple functionality selectable by the user. One important modification has been the introduction of dynamic connections, which means the possibility of moving to a presentation dynamically, in such a way that the actual target presentation depends on some condition tested at run-time.

4 The Proposed Approach

In order to reach our goals, the proposed environment is based on a user interface generator (UIG) server, whose architecture is represented in Figure 2, which receives access requests from the user.

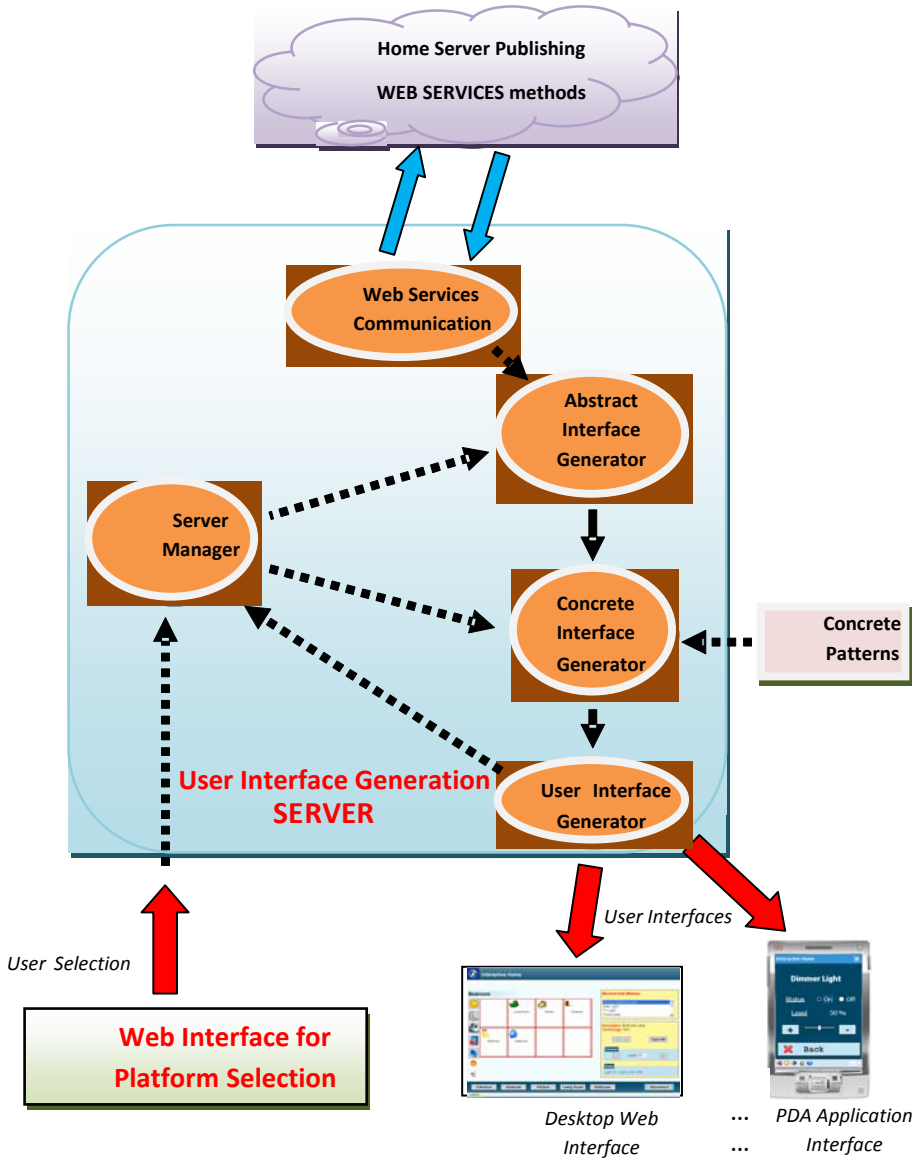

Fig. 2. The Architecture for Platform-dependent User Interface Generation

The first request is the selection of the environment that the user prefers to use in terms of type of device (mobile, desktop, vocal, …) and implementation language preferred (XHTML, Java, C#, …).

At the time of the request, the UIG server accesses the home server, which supports access to the home appliances. Such home server is accessible through Web services, which export the list of possible methods and their parameters through an

XML-based WSDL file. In particular, the Web services provide information regarding what home appliances are available, their location in the home, and their current state, as well as supporting state change requests. The UIG server contains a module, which is able to take the information from the Web services of the home server, and then pass it to the module in charge of building an abstract description of a user interface able to support access to the home devices. The information passed includes the list of methods supported by the functionality of the domestic appliances and the type of parameters they can accept.

The abstract description is a platform-independent description, which is then refined into a concrete, platform-dependent description. In order to complete the concrete description, the tool also uses some predefined presentation patterns for the considered application domain (home), which include some relevant content (icons, texts, …). At this point the server is ready for the generation of the final implementation, which is then uploaded to the current user device.

During the user session, the user interface software accesses the home Web services. For this purpose, if the user interface is implemented for a Web environments then a set of Java servlets are generated along with the user interface implementation, which become part of the server manager. They will be the elements supporting communication in both directions between the user interface and the home services. Thus, the user interface generated will include indications on what servlet to activate in case of generation of requests to modify the state of any home device, as well as on the servlets that can dynamically update the user interface content in order to provide dynamic information regarding the state of the domestic appliances.

5 Mapping Home Web Services onto Abstract User Interface Descriptions

In the module for mapping the Web services onto the abstract user interface description, we assume that the application refers to a home, which is composed of various rooms. In each room there is a number of devices, which belong to some device category (such as DimmerLightBulb, thermostat, media player, …). For each device category, the Web service provides a list of associated methods, which allow users either to access their state or to modify it. If we analyze the devices' functionality in detail, we can note that each device is associated with a set of functionalities that are independent of the specific model of the device, one parameter is the device id that is used to distinguish among various devices in the same category. We now discuss a subset of home devices considered in order to illustrate how our approach works. Other devices considered include media players able to support remote access to various types of multimedia files.

The *LightBulb* device is associated with the methods:

- *turnOnLight*: has no return parameters, it is a write-only method with a Boolean as second parameter. Thus, it is used to send two possible values (on and off), each of which can be associated with a specific button.
- *isLightOn*: has a Boolean return parameter, thus it is a read-only method. The representation of the value can be given by an output-only object

DimmerLightBulb is a device subclass of lightBulb, which adds the possibility to control the brightness value. Its methods are:

- *setDimmerValue*: has no return parameter, it is a write-only method that accepts as input as second parameter a short integer indicating the value that is set for the dimmer. Thus, the corresponding additional interaction element is a numerical_in_range_edit object, whose parameters are the min and max possible values and a Boolean indicating whether the range is continuous.
- *getDimmerValue*: has a return value indicating the state of the dimmer, which can be represented through an output-only object.
- *getDimmerRange*: is useful to know the limits of the possible values, which can be represented through two output-only objects.

The *thermostat* is associated with the methods:

- *setCurrentTemperature*: has no return parameter, thus it is a write-only method, and the second parameter is a short integer, which can be edited through a numerical_edit object.
- *currentTemperature*: has a short integer as return parameter, thus it is a read-only method, which can be associated with an output-only text object.
- *getcurrentTemperatureRange*: is useful to know the range of the possible values independent of the adopted solution for the control.

Sensor represents any type of sensor that can generate an alarm, and it has only one method:

- *getSensorStatus*: has a Boolean as a return parameter, thus it is a read-only method. When the Boolean is set to true then an alarm object is activated.

Alarm represents an alarm device and has the methods:

- *setAlarmState*: has no return parameters, thus it is a write method with the second parameter as integer. Usually three values are used ON/Off and an intermediate value.
- getAlarmState: is a read method, whose return parameter is an integer. The representations of such values can be (Total – Partial - Off) .

In the application of these mappings, we could obtain cases in which an interactive element to set the state of a device is separated from the output element that shows the device state. However, in some cases, for example a mobile user interface in which screen space is limited, it may be useful to have a single interactive element able to cover both aspects (possibility of changing the state and showing actual state). For example, a dimmer can have a slide bar control for both purposes: showing the current value, which can be received from the home device, but also allowing the user to change it sending the new value as result of the interaction. In order to identify such cases, we have developed a heuristic indicating that when in the WSDL we find two methods with complementary structures (such as set xxx value and get xxx value) associated to one device, then they are mapped onto one element able to support both methods instead of two separate interface elements. These mappings are exploited in the building of the abstract user interface.

In this approach, the goal is to obtain an abstract description of a user interface, which when it will be generated it will be able to directly communicate with the Web services. Through this communication, some parts of the user interface will be

dynamically filled in (in terms of data values), such as the list of available home devices, eventually filtered by type. Below we show an excerpt of the WSDL considered. At the beginning the types of home devices are defined. All of them are subclass of DomoDevice, which has the common basic attributes (such as room, name, …).

In the following WSDL excerpt, there is the Light Bulb, which has the methods TurnOnLight and IsLightOn; we can also note that the TurnOnLight has two parameters: the device (LightBulb) and a Boolean:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ... >
  <wsdl:types>
    ...
     <s:complexType name="LightBulb">
        <s:complexContent mixed="false">
          <s:extension base="tns:Lighting"/>
        </s:complexContent>
      </s:complexType>
     ...
      <s:element name="TurnOnLight">
        <s:complexType>
          <s:sequence>
<s:elementminOccurs="0" maxOccurs="1" name="bulb" type="tns:LightBulb"/>
<s:element minOccurs="1" maxOccurs="1" name="on" type="s:boolean"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      …….
      <s:element name="IsLightOn">
        <s:complexType>
          <s:sequence>
<s:elementminOccurs="0" maxOccurs="1" name="bulb" type="tns:LightBulb"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      ...
  </wsdl:types>
      ...
  </wsdl:portType>
  ...
</wsdl:definitions>
```

6 From the Abstract Description to the Concrete Descriptions and the Implementations

The abstract structure of the resulting user interface is structured as a vertical grouping of three grouping elements (see Figure 3): one dedicated to the header (containing a logo/title), one to the main area, and one to the footer (containing some controls that allow dynamic filtering of the device list, for example according to the type of room or to the type of device. The corresponding user interface is in Figure 4: in the footer grouping there are the buttons associated with the type of rooms available in the home and general controls, such as the Disconnect button. In the main area there are two grouping: one dedicated to the map zone (which provides a graphical representation

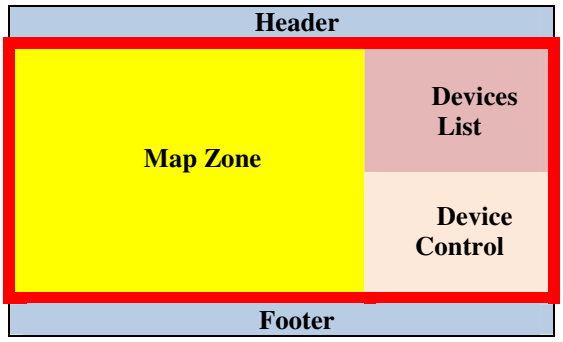

Fig. 3. The structure of the desktop interface

of the rooms available), and one to the device area, which is a vertical grouping of a grouping dedicated to the available devices' list and one to the controls for the currently selected device.

Below there is the excerpt of the abstract description that indicates how this presentation is structured through the composition operators:

```
<?xml version="1.0"?>
<!DOCTYPE interface PUBLIC …  >

<interface>
  <presentation name="Main_Presentation">
    <interactor_composition>
      <operator name="grouping_Application" />
            <interactor_composition>
            <operator name="grouping_Header" />
            …
            </interactor_composition>
            <interactor_composition>
            <operator name="grouping_Central_Zone" />
                  <operator name="grouping_Map_ZONE" />
                  …
                  </interactor_composition>
                  <interactor_composition>
                  <operator name="grouping_Device_Area" />
                  <interactor_composition>
                  <operator name="grouping_Devices_List" />
                  …
                  </interactor_composition>
                  <interactor_composition>
                  <operator name="grouping_Devices_Control" />
                  …
                  </interactor_composition>
            </interactor_composition>
          <interactor_composition>
          <operator name="grouping_Footer" />
            …
          </interactor_composition>
</interactor_composition>
  </presentation>
</interface>
```

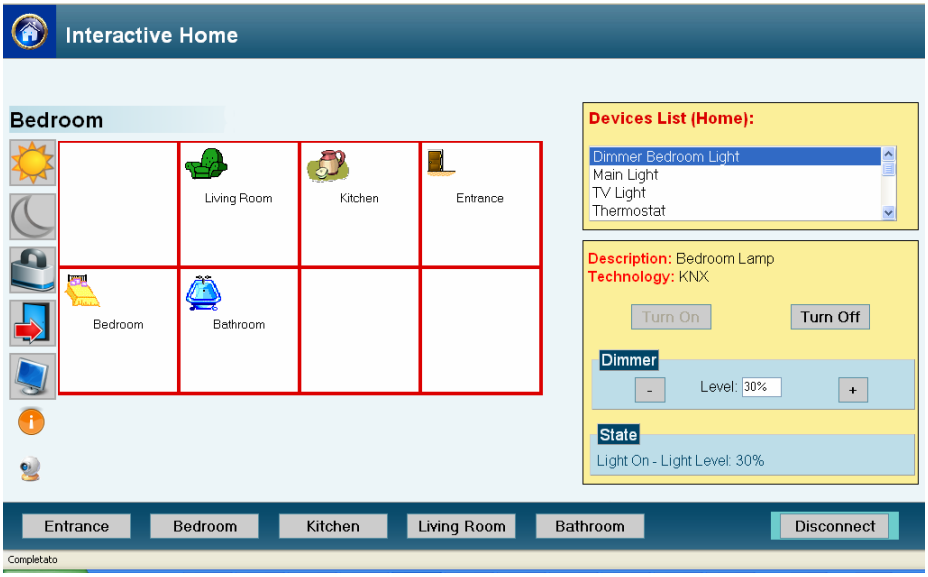

Fig. 4. The desktop interface

More in detail, each area contains specific interactors and composition operators. It is important to note that in the user interface there are some parts, which are dynamic (in particular the Device Control part). This means that, depending on the type of device dynamically selected at run time, different controls will be shown in this part of the user interface in order to operate with them. This has been obtained by extending the TERESA XML language in such a way to include dynamic connections, which means connections in which the target presentation changes depending on a value that is identified at run-time. Thus, through the analysis of the WSDL we are able to identify all the possible target presentations and the associated structure. In addition, we are also able to identify the values (in this case the device type value) which are associated with each of them.

In particular, while in the previous version of TERESA XML a connection was defined through the interactor triggering it and the corresponding target presentation:

interactor ---------- connection ----------> target presentation

Now, we also associate the interactor with a set of possible values, which are known through an analysis of the WSDL and the connection can have multiple target presentations:

Interactor (values) ---------- connection ----------> multiple target presentations

the actual value considered is generated at run-time depending on the user interaction with the interactor and determines which target presentation to activate. Thus, in our case at run-time, depending on the actual device selected by the user, different pages

with different controls will be shown. This was also obtained through another small extension in the TERESA XML language, for which the activator element can be associated with multiple functionality rather than only with a single one, as it happened in the previous version. This feature is used when the user selects a device from the available device list and depending on the selection a different Web service functionality will be activated. Below we can see the abstract user interface description of the Device Control in the case of a Dimmer Light Bulb type of device:

```
<interactor_composition>
<operator name="grouping_Basic_Device_Control" />
                <interactor id="TurnOn_Button">
                   <interaction category="interaction">
                    <control type="control">
                   <activator object="activator" />
                    </control></interaction></interactor>
                <interactor id="TurnOff_Button">
            <interaction category="interaction">
                    <control type="control">
                <activator object="activator" />
                     </control></interaction></interactor>
      </interactor_composition>
      <interactor_composition>
   <operator name="grouping_Advanced_Control" />
        <interactor id="Decrease_Level_Button">
            <interaction category="interaction">
                <control type="control">
                <activator object="activator" />
                 </control></interaction></interactor>
                 <interactor id="Level_Light">
                 <interaction category="interaction">
                 <edit type="edit">
                 <text_edit object="alphanumeric" />
                 </edit></interaction></interactor>
          <interactor id="Increase_Level_Button">
            <interaction category="interaction">
                  <control type="control">
                  <activator object="activator" />
                  </control></interaction></interactor>
              </interactor_composition>
          </interactor_composition>
```

We can note that the `grouping_Basic_Device_Control` represents the set of basic controls (associated with a Light Bulb), while the `grouping_Advanced_Control` represents the additional controls (associated with a Dimmer Light Bulb), which provide the additional possibility of choosing the brightness level. Figure 4 shows the resulting user interface for a desktop platform.

The creation of the mobile version is obtained by applying a cost-based semantic redesign transformation in the process of building the concrete description. The starting point is still the same abstract user interfaces. The first version of the concrete description created preserves the same structure but is associated with content for the mobile device in this case. This means smaller icons and, generally, more simplified representations. Then, the concrete description is transformed to better match the currently available resources. Thus, it takes information regarding the screen size of

the current device and depending on this it splits the original presentations into presentations more suitable for the current target device. The splitting is based on the logical structure of the user interface. This means that the resulting cost of the composed elements is calculated and if it is too expensive for the device then a new presentation is allocated for this set of elements and the connections to support navigation with it are automatically generated. In our example, simplified versions of the header and footer are generated. Then, the grouping associated with the room list has a cost sufficient to fill in a mobile presentation. The grouping associated with the list of available devices, which is dynamically filled in at run-time is associated with another specific mobile presentation. Also the grouping associated with the device controls has a cost sufficient to fill in a presentation. Lastly, the corresponding user interfaces are generated, Figure 5 shows three presentations for the mobile version.

Fig. 5. The interface for the mobile device

7 Conclusions and Future Work

We have reported on a work that aims to bridge the use of Web services and model-based user interface generation for home applications. We have discussed the method developed for this purpose and the corresponding software architecture and prototype implementation. In the paper we have also discussed how TERESA XML has been extended in order to support a more flexible set of interaction techniques and dynamic pages, whose interactive elements depends on information-generated dynamically at run-time in the communication between the user and the home Web services.

Future work will be dedicated to testing the usability of the automatically generated user interfaces for the various interactive devices, considering the use of ontologies for richer semantic descriptions and analysis, and the application to other case studies (such as remote elderly monitoring and assistance).

References

1. Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J.: UIML: An Appliance-Independent XML User Interface Language. In: Proceedings of the 8th WWW conference (1999)
2. Bickmore, T., Girgensohn, A., Sullivan, J.: Web-page Filtering and Re-Authoring for Mobile Users. The Computer Journal 42(6), 534–546 (1999)
3. Buyukkokten, O., Kaljuvee, O., Garcia-Molina, H., et al.: Efficient Web Browsing on Handheld Devices Using Page and Form Summarization. ATOIS 20(1), 82–115 (2002)
4. Calvary, G., Coutaz, J., Thevenin, D., Bouillon, L., Florins, M., Limbourg, Q., Souchon, N., Vanderdonckt, J., Marucci, L., Paternò, F., Santoro, C.: The CAMELEON Reference Framework, Deliverable D1.1 (2002)
5. Florins, M., Vanderdonckt, J.: Graceful degradation of user interfaces as a design method for multiplatform systems. Intelligent User Interfaces, 140-147 (2004)
6. Gajos, K., Christianson, D., Hoffmann, R., Shaked, T., Henning, K., Long, J.J., Weld, D.S.: Fast and robust interface generation for ubiquitous applications. In: Beigl, M., Intille, S.S., Rekimoto, J., Tokuda, H. (eds.) UbiComp 2005. LNCS, vol. 3660, pp. 37–55. Springer, Heidelberg (2005)
7. Limbourg, Q., Vanderdonckt, J.: UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. In: Engineering Advanced Web Applications. Rinton Press, Paramus (2004)
8. Lin, J., Landay, J.: Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces. In: Proceedings, C.H.I. (ed.) Proceedings CHI 2008, Floremce (April 2008)
9. Miori, V., Tarrini, L., Manca, M., Tolomei, G.: - An open standard solution for domotic interoperability. IEEE Transactions on Consumer Electronics 52(1), 97–103 (2006)
10. Nichols, J., Myers, B.A., Rothrock, B.: UNIFORM: Automatically Generating Consistent Remote Control User Interfaces. In: CHI 2006, pp. 611–620. ACM Press, New York (2006)
11. Paternò, F.: Model-based Design and Evaluation of Interactive Applications. Springer, Heidelberg (1999)
12. Paternò, F., Santoro, C., Mantyjarvi, J., Mori, G., Sansone, S.: Authoring Pervasive MultiModal User Interfaces. International Journal of Web Engineering and Technology (2) (2008)
13. Song, K., Lee, K.-H.: An automated generation of xforms interfaces for web services. In: Proceedings of the International Conference on Web Services, pp. 856–863 (2007)
14. Spillner, J., Braun, I., Schill, A.: Flexible Human Service Interfaces. In: Proceedings of the 9th International Conference on Enterprise Information Systems, pp. 79–85 (2007)
15. Vermeulen, J., Vandriessche, Y., Clerckx, T., Luyten, K., Coninx, K.: Service-interaction Descriptions: Augmenting Services with User Interface Models. In: Proceedings Engineering Interactive Systems 2007. Springer, Heidelberg (2007)