Designing Usable Applications based on Web Services

Fabio Paternò, Carmen Santoro, Lucio Davide Spano
HIIS Laboratory – ISTI-CNR
Via Moruzzi 1
56126 Pisa, Italy
{fabio.paterno, carmen.santoro, lucio.davide.spano}@isti.cnr.it

ABSTRACT
One trend in software development is to implement application functionalities through Web services. This eases the possibility of developing interactive applications exploiting functionalities implemented by others. In this paper we discuss the issues raised when designing user interfaces for these types of applications. In particular, we describe a possible approach to address them based on the use of model-based user interface descriptions with the possibility of obtaining versions adapted to different types of interactive devices. The development of the final user interface is supported by a semi-automatic process in which at first the designers take benefit of an authoring tool able to automatically generate the first version of the user interface and then, after an evaluation phase of of the resulting user interface, they can manually make further modifications and refinements in order to obtain highly usable user interfaces.

Categories and Subject Descriptors
H5.m. Information interfaces and presentation (e.g., HCI).

General Terms
Design,

Keywords
Model/based design. Usability, Web services.

1. INTRODUCTION
One current trend in software development is the use of Web services. They have been introduced to better support software–to–software communication. This is achieved through the WSDL (Web Service Description Language) description associated with each service, an XML-based description of the possible operations, and input/output parameters. The basic idea is to ease the development of applications based on the SOA (Service–Oriented Architecture) approach in which often applications developers have to design access to services and their compositions developed by others.

Meanwhile, recent years have also assisted to a renewed interest in model-based user interface design and development because logical descriptions allow designers to better manage the complexity of multi-device environments (see for example [1], [4]). This is usually obtained by exploiting XML logical descriptions and associated transformations for the target devices and implementation languages. Having the possibility of specifying a user interface at different levels has several advantages: it helps designers because the separation in different abstraction levels provides different "views" of the same user interface, and the selection of the most appropriate view is performed by the designers depending on the specific aspects they are currently interested in, and/or on their specific skills. In addition, it is worth pointing out how not only designers can be involved in the approach, but also other stakeholders can play a relevant role in the process. Indeed, as the method is supposed to be iterative and refinement-based with a semi-automatic approach there is enough room for even an early intervention of evaluation in the process, to the aim of identifying usability problems and include the design of their solutions as soon as possible in the user interface software lifecycle.

In addition, the information contained in the models can be exploited both at design and at run time. Therefore, the use of models does not pose any particular constraints to when and how the models should be used. Maintaining links among the elements in the various abstraction levels enables e.g. linking semantic information (such as the activity that users intend to do) with more concrete levels, up to the implementation levels, and this can be exploited in many ways. For instance, such links can be automatically supported by suitable transformations, which are useful for obtaining a description in a specific abstraction level, once a description in a different level is available, not forcing designers to build all the different descriptions or to use any specific model.

If we consider the abstract level, it is generally recognised that the main benefits in using an abstract description of a user interface is for the designers of multi-device interfaces, because they do not have to learn all the details of the many possible implementation languages supported by the various devices. Thus, one advantage of using the abstract levels of a user interface is that designers can reason in abstract terms without being tied to a particular platform/modality/implementation language. In this way, they have the possibility to focus on the 'essence' of the interaction (e.g.: what is the intended effect the interaction wants to achieve/support?), regardless of the details and specificities of the particular environment considered. In addition, considering the abstract level of the user interface appears to be particularly useful when the user interfaces are aimed at handling Web Services. Indeed, WSDL files provide a description of the operations supported by the Web Services. The relationships of such descriptions of the operations (contained in WSDL files) with the

I-USED'08, September 24, 2008, Pisa, Italy

abstract user interface objects expected to support them in the resulting user interface (contained in the abstract user interfaces) is an interesting issue to investigate, and appears to be promising in helping to solve the problem of generating user interfaces for Web Services.

This is a novel problem. Indeed, most model-based approaches have not addressed the specific issues related to Web-service based applications. Work on generating user interfaces for Web services but without using model-based approaches has been carried out at Dresden [6] and Yonsei [5] universities. The limitation is that such works usually consider direct mappings between Web services functional interface and an implementation language for user interface, which cannot be exploited when devices supporting different implementation languages are considered.

is that designers and developers have to create interactive applications accessing application functionalities developed by others. Thus, they have to focus their effort on how to take into account the specific requirements that the application interface of the existing Web services pose. In addition, they also have to indicate how to compose functionalities implemented in different Web services.

Our approach (see Figure 1) is to have first a bottom-up step in order to analyse the Web services providing functionalities useful for the new application to develop. In this phase an analysis of the operations (OP1, .. OP4 in Figure 1) and the data types (DT1, .., DT4 in Figure 4) associated with input and output parameters is carried out in order to associate them with abstract interaction objects suitable to support presentation of their values and their modification.

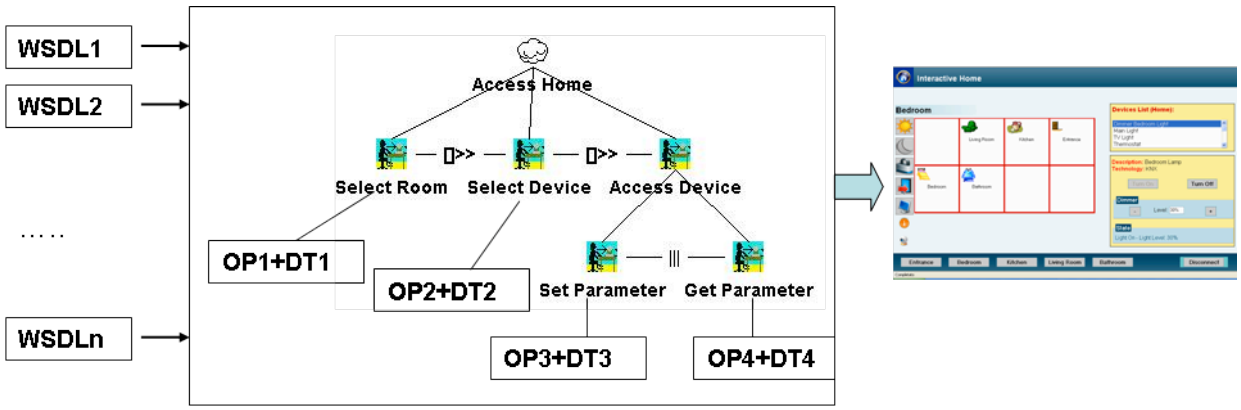

Figure 1. The Proposed Approach.

Work by Vermeulen et al. [7] aimed to solve such issues by extending Web services with OWL-S combined with task and layout model. This approach requires a lot of manual work by the designers.

In general, the type of issue that we have to solve is how to associate information regarding the data types and information on the user interface. Indeed, while the semantic Web has mainly focused on the data semantics through the use of ontologies and languages that allow for more intelligent processing, user interface models allow designers to consider the semantics of interaction, which is related to the tasks to support in order to reach the users' goals. Thus, we need to link these two types of information.

In this paper we discuss how to address the issues introduced by proposing a specific approach and a language and the associated environment, which builds on our previous experiences but aims to provide better support when designing interactive applications based on Web-services, we also report on how the approach can be applied in an application in the home domain and how it supports the interplay between software development and usability evaluation.

2. METHODOLOGICAL APPROACH

A traditional top-down approach going through the various abstraction layers that have been considered useful in HCI (task, abstract interface, concrete interface, implementation) does not seem particularly effective in this context for various reasons. One

For example, a Boolean can be represented by a button, an enumeration type by a list or a radio button depending on its cardinality. Thus, for each Web services we can have a corresponding abstract description of the user interface.

Then, we can use the task model expressed in ConcurTaskTrees (CTT) [3] for describing the interactive application and how it assumes that tasks are performed. This notation is a standard de facto in the area of task model representations, and it also under consideration for standardization in the W3C consortium. CTT provides a first classification of tasks depending on the agent performing them: the user (in case of only internal cognitive activity, such as making a decision about how to carry on a session), the system (completely automatic task) or interaction (involving both the user and the system). Web services are application functionalities, thus they are associated with system tasks. Another issue is what level of granularity to reach in the task decomposition. There are mainly two possibilities: associating the system basic tasks to the web services or reach a further detail in order to associate each system basic task with the operations of the web services. Thus, if a Web Service supports three operations, then there would be three basic system tasks. The latter solution allows for a more detailed and flexible specification.

The next step is to obtain first an abstract, and then a concrete, platform-dependent, user interface description of the user interface. To this end we have to consider information derived

from the task models and the various pieces of abstract interface associated with the various Web Services. The information coming from the task model is useful in order to identify how to structure the presentations of the interactive applications and define the navigation model through them. The information coming from the abstract interface excerpts are mainly useful to identify the interface elements to include in each presentation and their type. Defining the structure of a presentation mainly means to identify the logical groups of elements inside it, and whether there are particular relations among some of such groups The structure of the Web services can be useful for this purpose because we can think of 'groupings' associated with each operation (indicating how to represent their input and output parameters), and higher level groupings associated with the Web services.

The use of an automatic tool and, consequently, automatic transformations, has several advantages: it allows for generating usable and consistent user interfaces by incorporating already in the transformation rules some design guidelines/rules for obtaining usable interfaces. In addition, it also allows for ensuring that some minimal consistency overall the pages is automatically kept (eg: ensuring the consistency of the title label or the style of the presentations overall an entire user interface application). However, very often the results of fully automatic authoring tools for user interfaces are not very satisfactory from a usability point of view, even when some good design rule have been incorporated in the transformations.

To this aim in our approach a semi-automatic process has been proposed. Such a process provides also space for (a manual) intervention, an evaluation phase carried out on an initial version of the user interface that has been automatically generated. Therefore, in order to improve the usability of the final results, an evaluation feedback from a HCI expert can be envisaged so that a consequent manual refinement and modification of the user interface which has been automatically obtained with the authoring tool can be carried out accordingly.

3. MARIA

In order to obtain a more powerful description language able also to satisfy the new requirements posed by service/oriented architectures, and modelling the new forms of human-computer interaction, we are developing a new UI specification language, which will take also into account the new technical requirements raised by the issue of generating usable interfaces for Web Services. The new language name is MARIA (Model-bAsed descRiption of Interactive Applications) XML and it can be used for the abstract and concrete user interface definition. Its development takes into account our previous experiences with a previous language (and the associated tool) for designing multi-device user interfaces, TERESA XML [4].

There are many differences between TERESA XML and MARIA XML. For example, MARIA supports also an abstract description of the underlying data model of the application. The interactors (namely, the elements of the abstract or concrete user interface) which compose an abstract (resp.:concrete) user interface, can be bound to a type or an element of a type defined in the abstract model. In this way, a change of application status is modelled as a change of one or more values in the abstract data, which will be reflected on the interface (abstract or concrete) status. This is a powerful feature that can be used to express in a natural manner aspects such as correlation between the value of interface elements, conditional presentation connections, conditional layout of interface parts, etc.

The data model is described using the XSD type definition language. In MARIA there is the possibility to define the data manipulated by the user interface both at the abstract level (through an abstract data model) and at the concrete level (a concrete data model, which is a refinement of the abstract one). The introduction of a data model at the abstract level also allows for having more control on the operations that will be done on the different data types. In addition, the data model is also useful for specifying the format for values: the format specification for a value can be expressed in MARIA by bounding the concrete data model with the editing interactor used for getting the input value from the user: if the editing object is bound with a date, the underlying implementation will have the needed information for validating the value that will be provided by the user. The MARIA data model can be the same as the types part of the WSDL description of the service, or it can be mapped on a more UI oriented description using an XSLT style sheet. The new authoring environment for MARIA XML is currently being developed to support this operation. The problem of mapping fields to services parameters is also supported by the MARIA environment: the mapping is obtained by performing the inverse operation of the process described before: another XSLT style sheet performs the mapping from the AUI data model to the WSDL one.

Figure 2 shows a graphical representation of an abstract interactor of type *only_output* in MARIA XML abstract user interface description (the description has been unfolded only for the higher levels).

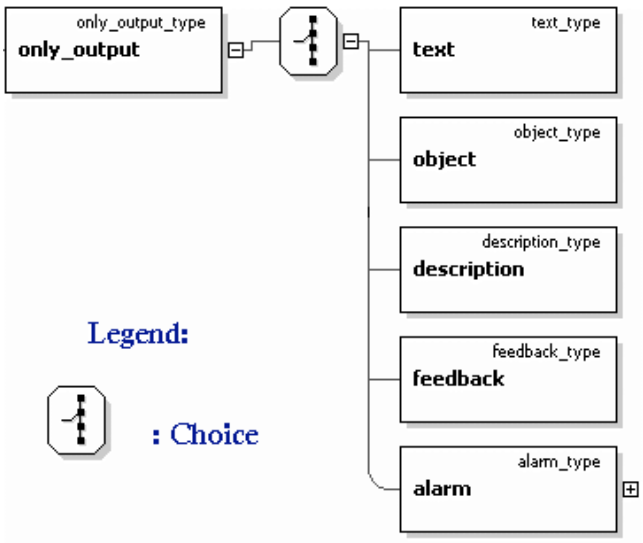

Figure 2. The specification of the *only_output* element

The *only_output* interactor models the possibility for the user to receive information from the application, and, depending on the type of information received (text, object, description, feedback, alarm), suitable interactors should be used.

In the same way, within MARIA XML an abstract interactor allowing the user to interact with the underlying application is refined into different objects depending on the type of activity which is supported: selection (select an object within a set of objects), edit (modifying an object), control (activating a functionality), and interactive description (a combination of both *only_output* and interaction objects).

Figure 3 presents a graphical representation of an abstract interactor of type *interaction* in MARIA XML abstract user interface description (the description has been unfolded only for the higher levels).

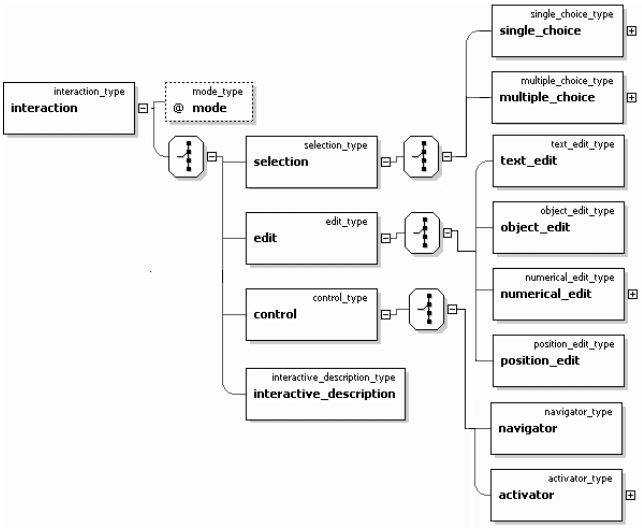

Figure 3. The specification of the *interaction* element

The corresponding excerpt of MARIA XML Schema for the abstract user interface description of the abovementioned interaction object (only for the first level) is visualised, together with the specification of the two possible types an interactor can assume (interaction and only_output):

```
<xs:complexType name="interaction_type">
  <xs:choice>
    <xs:element name="selection" type="selection_type"/>
    <xs:element name="edit" type="edit_type"/>
    <xs:element name="control" type="control_type"/>
    <xs:element name="interactive_description"
type="interactive_description_type"/>
  </xs:choice>
  <xs:attribute name="mode" type="mode_type" fixed="input"/>
</xs:complexType>

<xs:complexType name="interactor_type">
  <xs:choice>
    <xs:element name="interaction" type="interaction_type"/>
    <xs:element name="only_output" type="only_output_type"/>
  </xs:choice>
          …
</xs:complexType>
```

Figure 4 shows how it is possible, with MARIA XML, modelling a concrete user interface object (for the desktop platform) allowing for editing a textual value. More in detail, in the figure it is visualised the hierarchy of concrete interactors unfolded only for the branch of textfield objects, which allow editing text-based values. Textfields have a number of attributes, label (the label of the interactor), length (the length of the field), and the information about whether the field is aimed at accepting passwords (therefore the object should have a special behaviour in the feedback -eg: in a graphical platform it will not visualise the inserted value).

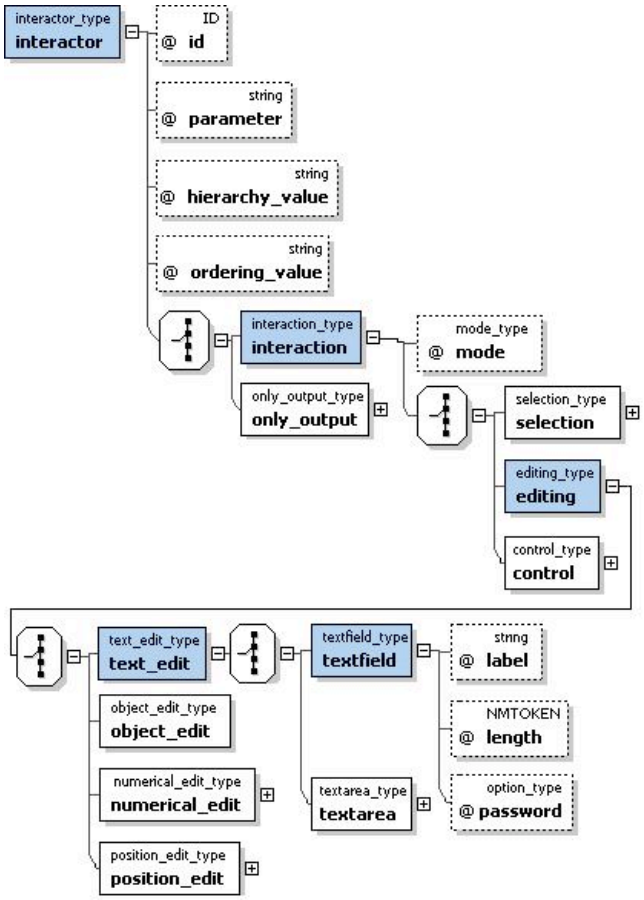

Figure 4. The specification of the *textfield* element

In Figure 4 below the objects derived from refining the *interactor* object down to the *textfield* object have been highlighted with a different colour, in order to make clearer to the reader such decomposition.

Below there is the corresponding MARIA XML specification excerpt for the textfield object

```
<xs:complexType name="textfield_type">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="label" type="xs:string" use="required" />
<xs:attribute name="length" type="xs:NMTOKEN"
use="required" />
<xs:attribute name="password" type="option_type"
use="required" />
</xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

4. AN EXAMPLE APPLICATION

We have applied our approach to the design of a home application. This is an application domain that is raising an increasing interest because our houses are becoming more and more populated with interactive, intelligent devices. In this case we have used a home server able to support interoperability among home devices supporting various communication protocol (X10, UpnP, Konnex, …). The functionalities of the domestic appliances are made available through a standardised set of Web services exposed by a home server [2]

This case study has also provided us with the possibility to define an algorithm for generating a default user interface for accessing a Web service operation. The idea is that the generated UI can then be refined by a designer with an authoring environment, but we want to create heuristics to minimize the need of human intervention.

For the sake of simplicity, in order to illustrate the algorithm we take into account a single service with a finite set of operations and data types.

The algorithm has a first step that aims to extract an object oriented model of the operations: each operation is associated to a data type (the type "owner" of the operation) defined in the types part of the WSDL file, checking the operation parameters.

After that the constructed model is reviewed for identifying the input and output operations that read or write the same properties. For instance we can take into account a Sensor data type that contains an element *status*. The WSDL has two operations :

- SetSensorStatus(Sensor s, boolean status)
- Boolean GetSensorStatus(Sensor s)

These two operations are bound to the Sensor data type, the first one is an input operation that writes a value in the *status* field and it is marked as input, while the second is a read operation and it is marked as output (it delivers as a result a Boolean data, as you can see from its specification). When the parameter that represents the value of the input operation matches with the read value of an output operation, their names are checked using the following heuristic: if the names are similar enough, they are merged into a single input/output operation: as a consequence, the same interactor will be used for supporting the input and output operations. Otherwise the two operations will remain distinct and different interactors will be used for accessing the two operations.

As an explanatory example of the above concept, we could consider a mobile user interface in which screen space is limited, and therefore it may be useful to have a single interactive element able to cover both aspects (possibility of changing the state and show actual state). In order to identify such cases, we have developed a heuristic indicating that when in the WSDL we find two methods having complementary structures (such as *set*<value> and *get*<value>, like e.g. *setSensorStatus* and *getSensorStatus* before) associated to one device, then they are mapped onto one element able to support both methods instead of two separate interface elements.

Enumeration data type, with high cardinality	

Abstract User Interface	`<selection>`
	`  <single_choice`
	`cardinality="high"/>`
	`</selection>`

Concrete Desktop	`<selection>`
	`  <single`
	`  cardinality="high">`
	`  <list_box`
	`  alignment="…">`
	`  <choice_element`
	`  label="[elementName]">`
	`  elementName`
	`  </choice_element>`
	`  [Other elements]`
	`  </ list_box >`
	`  </single>`
	`</selection>`

Concrete Mobile	`<selection>`
	`  <single`
	`  cardinality="high">`
	`  <drop_down_list`
	`  alignment="…">`
	`  <choice_element`
	`  label="[elementName]">`
	`  elementName`
	`  </choice_element>`
	`  [Other elements]`
	`  </drop_down_list>`
	`  </single>`
	`</drop_down_list>`

Concrete Vocal	`<selection>`
	`  <single`
	`  cardinality="high">`
	`  <message_menu`
	`  message="…"`
	`  nomatch_event="[nomatchmsg]"`
	`  noinput_event="[noinputmsg]"`
	`  help_event="[helpmsg]" >`
	`  <message>`
	`  [elementName]`
	`  </message>`
	`  [Other elements]`
	`  </message_menu>`
	`  </single>`
	`</selection>`

Figure 5. Examples of mappings

The next step is the creation of the abstract user interface using the collected operation information. The table shown in Figure 5 describes an example of the main mapping rules in the case of an enumeration data type with high cardinality, by showing an abstract *single_choice* element and the corresponding concrete elements for the desktop, mobile, and vocal platforms.

In this way it is possible to obtain an application able to support access through multiple types of devices. For example, it is possible to generate versions for a PDA and a desktop system, but other device types can be considered as well. In the case of the PDA access, we consider the possibility of generating an application in C#, even able to support libraries for vocal and multimodal access. This application has to be downloaded and installed in the mobile device. In the case of a desktop access, we consider the generation of a Web application able to support access, through some servlets, to the web services associated with the domestic appliances. Whatever interaction device is actually used, then the user can freely choose one domestic device and perform the desired information, usually check the state of some parameters (such as temperatures or alarms) or change some of their values.

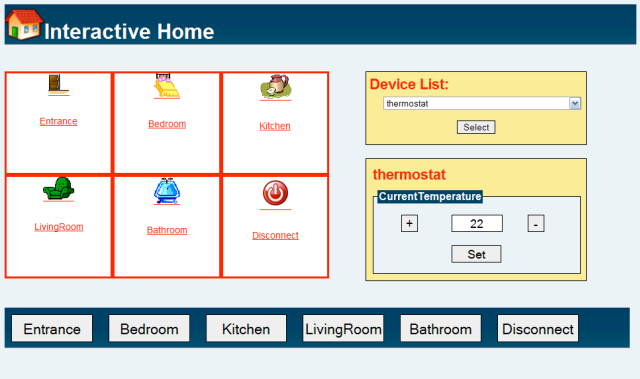

Figure 6. The Desktop User Interface.

The different screen space implies substantial differences in the generated user interfaces. In the desktop interface (see Figure 6) it is possible to show the various rooms, select a device and access the associated controls in *one single* presentation.

Figure 7. The PDA User Interface.

All these possibilities are still available in the PDA interface (see Figure 7) but they require multiple presentations and the addition of navigation capabilities among them.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed a method for the model-based design of interactive applications based on the use of Web services. We have also briefly reported on the development of a new XML specification language, and the associated authoring environment, to better support the method, and, more generally, to provide more flexible support to UI designers. We have also illustrated the approach with a specific example in the home domain.

In addition, we pointed out how our approach leverages an easy coupling between on the one hand software design and development and, on the other hand, usability evaluation. Indeed, the approach is supported by a semi-automatic process in which an initial version of the user interface is expected to be obtained through the use of automatic tools in which some guidelines for good UI design are already incorporated (eg within the transformation rules). Therefore, the initial results automatically obtained should already be compliant with principles of good design if they have been incorporated in suitable transformations (which, if not hard coded in the automatic tool can be even subject of an usability evaluation as well). Afterwards, the preliminary versions of the user interfaces so obtained are supposed to be analysed and evaluated by HCI experts: the feedback of such an evaluation can be included through a manual refinement which can affect (and, hopefully improve) the result not only at the final UI level but also at more abstract UI levels, depending on their skills.

Future work has been planned for applying the presented approach to more complex case studies, in order to test the generality and the flexibility of the method.

6. ACKNOWLEDGMENTS

This work is partly supported by the ServFace EU ICT project.

7. REFERENCES

[1] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. UIML: An Appliance-Independent XML User Interface Language, Proceedings of the 8th WWW conference, 1999.

[2] Miori V., Tarrini L., Manca M., Tolomei G. - An open standard solution for domotic interoperability. In: IEEE Transactions on Consumer Electronics, vol. 52 (1) pp. 97-103. IEEE, 2006.

[3] Paternò F., Model-based Design and Evaluation of Interactive Applications, Springer Verlag, November 1999, ISBN 1-85233-155-0.

[4] Paternò F., Santoro C., Mantyjarvi J., Mori G., Sansone S., Authoring Pervasive MultiModal User Interfaces, International Journal of Web Engineering and Technology, N.2, 2008

[5] Song, K., Lee, K.-H., 2007. An automated generation of xforms interfaces for web services, Proceedings of the International Conference on Web Services, 856-863.

[6] Spillner, J., Braun, I., Schill, A., 2007. Flexible Human Service Interfaces, Proceedings of the 9th International Conference on Enterprise Information Systems, 79-85.

[7] Vermeulen J., Vandriessche Y., Clerckx T., Luyten K. and Coninx K., Service-interaction Descriptions: Augmenting Services with User Interface Models, Proceedings Engineering Interactive Systems 2007, Salamanca, Springer Verlag.