A Model-based Approach to Address the Design of Web 2.0 Applications based on Web Services

Fabio Paternò Carmen Santoro Lucio Davide Spano

CNR-ISTI – HIIS Laboratory
Via Moruzzi 1, 56124, Pisa, Italy
{fabio.paterno, carmen.santoro, lucio.davide.spano}@isti.cnr.it

ABSTRACT
Creating an interactive application based on pre-existing functionalities poses a number of novel issues in the design process. We discuss a method, associated with a new model-based language and a supporting tool, which aims to address such issues in multi-device contexts. One specific aspect of this method is the possibility of supporting composition of user interfaces associated with different services. In addition, the possibility to specify dynamic user interfaces, Web services accesses and scripts allows designers to declaratively describe Rich Internet Applications as well.

Keywords
Model-Based Design, Multi-device Environments, User Interface Design, Web Services.

1. INTRODUCTION
Model-based approaches for UI design aim at supporting user interface design by means of some representations (models) of the aspects that are supposed to be relevant in the UI software lifecycle. This involves the identification and representation of the characteristics that are meaningful at each design stage, and mainly highlights one of the most difficult parts of the work: identifying what characterizes a UI without having to deal with a plethora of low-level implementation details that can distract the designer from the most important issues. After having identified such characteristics, the next issue is specifying them through suitable languages that can enable simple integration within software environments, so as to facilitate the work of the designers.

The design of interactive multi-platform systems has further stimulated interest in model-based approaches in HCI. In the design and development of such systems the use of model-based approaches has revealed to be useful, especially through the capture and modelling of different levels of abstractions in which it is possible to gradually move from aspects that are technology-neutral to more concrete, platform-dependent detailed aspects. In such a way it is possible to start with a general abstract vocabulary and then obtain concrete languages for each type of platform by just refining the abstract language.

However, recently, the design of such systems has become even more challenging. Indeed, not only it is required that the same interactive application be accessible from different devices, within different contexts of use, but in addition the way in which such interactive applications are built/created has changed, since there is the need to reuse existing code for reducing development time and effort. An example of this can be seen from the role that Web Services are playing in the development of interactive applications. Indeed, the increasing availability of functional units within Web Services has driven the need to develop methods that are able to exploit such pre-existing functionalities by including them into more composite interactive applications. In particular, some heterogeneous issues have to be faced by the designers in this case. First, the need for exploiting some (generally small) legacy functionalities that were developed without accounting for human interaction, since they were basically intended to support computer-to-computer (service-to-service) communication. Therefore, the first issue is how to obtain the UI for such functionality, possibly in a semi-automatic way, so that it can ease the work of the designer. Secondly, even when a UI for such portions of functionalities is available, there is the issue of including and integrating pre-existing user interfaces associated with functionalities into new, composite ones, and possibly support the designer during such composition.

In the paper, after discussing some related work we describe the main features of our approach for designing user interfaces for Web Services. We also introduce the dimensions of a design space for composing user interfaces in such context. Afterwards, we express the requirements that have driven the development of MARIA, an XML-based language for describing user interfaces at various abstraction levels. Then, we introduce an authoring environment aiming to support the model-based approach discussed in the paper. Lastly, some conclusions and directions for future work are provided.

2. RELATED WORK
Several model-based approaches have been put forward in the field of multi-device UIs. A sign of the maturity of this area can be seen by the recent interest in defining international standards connected with it (e.g.: new W3C Group on Model-based User Interfaces: http://www.w3.org/2005/Incubator/model-based-ui/)

and their adoption in industrial settings (e.g.: Working Group in NESSI NEXOF-RA IP, http://www.nexof-ra.eu/).

In particular, a number of approaches have been proposed to support descriptions of logical user interfaces. UIML [1] was one of the first model-based languages targeting multi-device interfaces. It structures the user interface in: structure, style content, behaviour, even if it has not been applied to obtain rich multimodal-user interfaces.

XForms [http://www.w3.org/MarkUp/Forms/] is a W3C initiative, and represents a concrete example of how the research in model-based approaches has been incorporated into an industrial standard. XForms is an XML language for expressing the next generation of Web forms, through the use of abstractions to address new heterogeneous environments. However, the language includes both abstract and concrete descriptions (control vocabulary and constructs are described in abstract terms, while presentation attributes and data types in concrete terms). XForms supports the definition of a data layer inside the form. XForms is mainly used for expressing form-based UIs and does not seem particularly suitable for supporting other interaction modalities, such as voice interaction. UsiXML (USer Interface eXtensible Markup Language) [3] http://www.usixml.org is an XML-compliant markup language developed at University of Louvain-la-Neuve, which aims to describe the UI for multiple contexts of use. UsiXML is decomposed into several meta-models describing different aspects of the UI. There is also a transformation model that is used to define model to model transformations between the different models. The authors use graph transformations for supporting model transformations, which is an interesting academic approach with some performance issues. TERESA XML [5] defines several abstraction levels for expressing the characteristics of a user interface. Among such levels, one (the concrete interface) is specified through a number of platform-dependent languages, which are refinements of the abstract level, which the user interface using a platform-neutral vocabulary: interactors (describing single interaction objects), composition operators (indicating how to compose interactors), presentations (indicating the elements that can be perceived at a given time). Various modalities have been supported through this approach. However, it does not support data and event models.

One issue with such model-based approaches is that they have not explicitly addressed the recent increasing trend in software design aiming to build atomic software components, called Web services, that are available in a distributed settings. Thus, applications have to be assembled starting from such pre-existing building blocks. Especially for enterprises this has represented several advantages in terms of reusing the code, augment productivity and leveraging integration processes. Some work has been dedicated to the generation of user interfaces for Web services [7, 8] but without exploiting model-based approaches. Previously, there have been approaches investigating the possibility of automatic generation with model-based support for applications based on Web services [4] but such approaches work well only with not too complex cases and when the application domain is well-known. In [9] 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. Therefore, model-based approaches have to cope with further requirements. There is less need to design an application from scratch, but they have to support interactive application development starting from small functionalities (services) that are already available, even if these were not built having in mind that particular application. In addition, the need of accessing the same service through an increasing number of device types (in particular mobile) available in the mass market, sometimes able to exploit a variety of sensors (e.g. accelerometers, tilt sensors, electronic compass), localization technology (such as RFIDS, GPS) and interaction modalities (multi-touch, gestures, camera-based interaction) have further urged the identification of suitable universal declarative languages able to address such composite number of aspects in a comprehensive specification.

3. THE APPROACH

A top-down approach essentially consists in breaking down and progressively refine an overall system into its sub-systems, thus it is particularly effective when the design starts from scratch. In such cases the designer has an overall picture of the system to be designed and s/he can refine it gradually and without any particular constraints.

However, when the designer wants to include already existing pieces of software like services, this necessarily requires that a bottom-up approach is considered in the design process in order to include and exploit in the design not only such legacy, fine-grained functionalities, but also composite and higher level functionalities that can result from assembling the elementary ones. Therefore, what seems the best option is a hybrid solution in which a mix of bottom-up and top-down approaches is used.

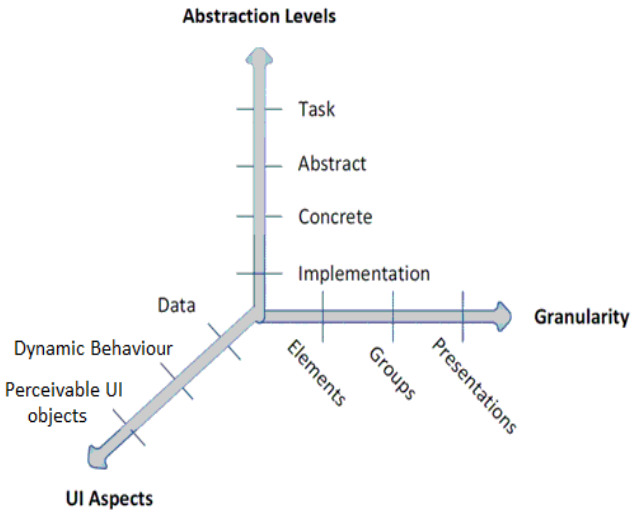

Figure 1. The Design Space for UI Composition

The problem of automatic or semi-automatic composition of existing Web services is one important issue in this context. Indeed, the design and development of an interactive application based on pre-existing Web services is by definition driven by a composition-oriented approach. Not only application logic

functionalities have to be composed (for this purpose various approaches already exist, e.g. BPMN) in order to provide arbitrarily complex behaviour but also the corresponding user interface specifications associated with the elementary services (which can be provided through specific annotations) can be composed as well.
In order to better understand how this user interface composition activity can be carried out we have identified a design space for this specific activity (see Figure 1).
Three main aspects have been identified as important in order to compose user interfaces: the abstraction level of the user interface description, the granularity of the user interface considered, and the types of aspects that are affected by the UI composition. Regarding the abstraction level, since a user interface can be described at various abstraction levels (task and objects, abstract, concrete, and implementation), it is straightforward that the user interface composition can occur at each of them. The granularity refers to the size of elements to be composed: indeed, we can compose single user interface elements (for example a selection object with an object for editing a value), groups of objects (for instance a navigation bar with a list of news), we can compose various types of interface elements and groups to obtain an entire presentation, and we can compose presentations in order to obtain the user interface for an entire application. It is worth pointing out that with the term 'presentation' we refer to the set of user interface elements that can be perceived at a given time, a common example is a graphical Web page.
Lastly, we have to distinguish the compositions depending on the main UI aspects that they affect, which are: i) the dynamic behaviour of the user interface, which means the possible dynamic sequencing of user actions and system feedback (e.g.: when some elements of the UI can appear or disappear depending on some conditions); ii) the perceivable UI objects (for example in graphical user interface we have to indicate the spatial relationships among the composed elements); iii) the data that are manipulated by the user interface.

More specifically, in the proposed approach first a bottom-up step is envisaged, in order to analyse the Web services providing functionalities useful for the new application to develop. This implies to analyse the operations and the data types connected with input and output parameters in order to associate them with suitable elementary abstract UI objects. Then, there is a step aiming to define the relationships occurring among such elements. Such relationships will allow the designer to compose such elementary objects in composite abstract expressions. We envisage for this step the use of task model, expressed in ConcurTaskTrees (CTT) [6], for describing the interactive application and how it assumes that tasks are performed. In this case, the Web services can be seen as a particular type of task (system task, namely task whose performance is entirely allocated to the application), and the temporal relationships that are specified in a task model will indicate how to compose such functionalities. This process (specifying the task model) should be driven by the user requirements and it also implies some constraints on how to express such functionalities. Indeed, in order to be able to address the right level of granularity, not only a Web service will be associated with an application task, but it is required that each operation specified within the same Web service be associated to a specific system task. Thus, if a Web Service supports three operations, then the designer should associate them with three basic system tasks (first arrow in Figure 2), with the parent task being another application task (corresponding to the Web service itself).
After having performed this step, we have obtained a first level of composition applied to the functionalities associated with the Web Services, and this has been done through the use of task models. Once we have obtained the task model it is possible (through a top-down step) to generate the various UI descriptions, and then refine them up to the implementation, by using the MARIA language (second arrow in Figure 2).

4. MARIA

Based on the lessons learned from the analysis of the state of the art and previous experiences conducted by various groups with TERESA (see [2] for a test in an industrial setting), we have identified a number of requirements for a new language suitable to support user interfaces in ubiquitous environments.
In particular, the following requirements have been identified for the new language, which were not well supported in TERESA and similar languages:

- the need to provide the designer with higher control of the user interface produced, through also an event model;
- the need for a more flexible dialogue and navigation model, supporting also parallel interactions;
- the need for a flexible data model, which allows the association of various types of data to the various interactors;
- the need to support recent dynamic techniques, such as Ajax scripts;
- The need to streamline the specifications of the abstract and concrete languages, in order to make the specifications shorter and more readable.

4.1 Main Features

A number of features have been included in the language:

a) Data model

We have introduced an abstract description of the underlying data

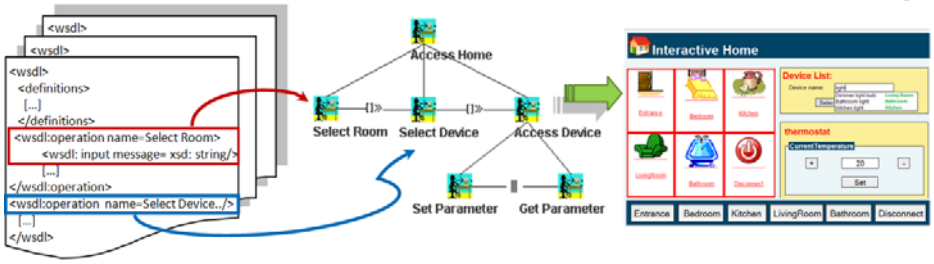

Figure 2. The Approach

model of the user interface, needed for representing the data (types, values, etc.) handled by the user interface. Indeed, by means of defining an Abstract Data Type/data model, the interactors (the elements of the abstract or concrete user interface) composing an abstract [concrete] user interface, can be bound to a specific type or an element of a type defined in the abstract [resp.:concrete] data model. The introduction of a data model also allows for more control on the admissible input that can be provided to the various interactors. In MARIA XML, the data model is described using the XSD type definition language. Therefore, the introduction of the data model can be useful for: doing some correlations between the values of interface elements (for instance, the value of one element can vary depending on the value of the another element), conditional presentation connections (triggering the activation of a presentation depending on a certain value associated to an interactor), conditional layout of interface parts (showing or not a portion of a presentation depending on the value associated to UI element), specifying the format of the input values (depending on the data type it is possible to specify a certain acceptable template for input values associated with that data type), application generation from the interface description (having information on the values associated with a UI description enables the actual generation of a working application).

b) Event model

In addition, an event model has been introduced, at different abstract/concrete levels of abstractions. The introduction of an event model allows for specifying at different abstraction levels how the user interface is able to respond to events triggered by the user. In MARIA XML two types of events have been introduced: i) Property change events: events which change the status of some UI properties (e.g. colours, fonts, …). ii) Activation events: some interactors can raise events with the purpose to activate some application functionality (e.g. access to a database or to a Web service). This type of event can have both change_properties or script handlers (which have associated a generic script).

c) Support for Ajax scripts, which allow continuously updating of fields

Another aspect that has been included in MARIA is the possibility of supporting continuously updating of fields at the abstract level. We have introduced an attribute to the interactors: continuosly-updated= "true"["false"]. The concrete level has the duty to provide more detail on this feature, depending on the technology used for the final UI (Ajax for Web interfaces, callback for standalone application etc.). For instance, with Ajax asynchronous mechanisms, there is a behind-the-scene communication between the client and the server about what has to be modified in the presentation, without explicit request from the user. When it is necessary the client redraws the relevant part rather than redrawing the entire presentation from scratch, thus it allows for quicker changes and real-time updates.

d) Dynamic set of user interface elements

Another feature that has been included in MARIA XML is the possibility to express the need of dynamically changing only a part of the UI. This has been specified in such a way to be able to affect both how the UI elements are arranged in a single presentation, and how it is possible to navigate between the different presentations. The possibility to change only a part of a presentation has been introduced. Therefore, the content of a presentation can dynamically change (this is also useful for supporting Ajax techniques). In addition, it is also possible to specify a dynamic behaviour that changes depending on specific conditions: this has been implemented thanks to the use of conditional connections between presentations.

In the next sections we provide a more detailed description of concepts/models that have been included in MARIA, both for the Abstract UI and the Concrete UI.

4.2 MARIA – Abstract Level

The advantage of using an abstract description 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 semantic of the interaction (namely: what the intended goal of the interaction is), regardless of the details and specificities of the particular environment considered. In our approach an interface is composed of one data model and one or more presentations. The presentation includes a data model and a dialog model, which contains information about the events that can be triggered by the presentation in a given time. The dynamic behaviour of the events is specified using the CTT temporal operators. When an event occurs, it produces a set of effects (such as performing operations, calling services etc.) and can change the set of currently enabled events (e.g. an event occurring on an interactor can affect the behavior of another interactor, by e.g. disabling the availability of an event associated to another interactor). The dialog model can also be used to describe parallel interaction between the user and the interface. A connection indicates what the next active presentation will be when a given interaction is performed and it can be either an elementary connection, or a complex connection (when a Boolean operator composes several connections) or a conditional connection (when various conditions on connections are specified).

There are two types of interactor composition: grouping or relation, the latter has at least two elements (interactor or interactor compositions) that are in relation each other. An interactor (see Figure 3) can be either an interaction object or an only_output object. The first one can assume one of the following types: selection, edit, control, interactive description, depending on the type of activity the user is supposed to carry out through such objects. An only_output interactor can be object, description, feedback, alarm, text, depending on the supposed information that the application provides to the user through this interactor. The selection object is refined into single_choice and multiple_choice depending on the number of selections the user can perform. The further refinement of each of these objects can be done only by specifying some platform dependent characteristics, therefore it is specified at the concrete level. All the interaction objects have associated events handlers in order to manage the possibility to describe how to react after the occurrence of some events in their UI. The events differ depending on the type of object they are associated with.

4.3 MARIA – Concrete Level

The concrete description is aimed at providing a platform-dependent but implementation language-independent description of the user interface. Thus, it assumes that there are certain available interaction resources that characterise the set of devices belonging to the considered platform. It moreover provides an intermediate description between the abstract description and that supported by the available implementation languages for that platform.

In order to enhance the readability of the language and also for consistency reasons (cross-references between different models enabling more consistency because they avoid to replicate the same data in two different places), we decided to furnish the concrete user interface only with the details of the concrete elements, leaving the specification of the higher hierarchy in the abstract meta-model. At this level differences associated with the specific characteristics of the platform will be modelled. For instance, when focusing on a iPhone platform the concrete user interface language has to express the fact that interaction is carried out through the use of not only a simple touch-based interface (which is also to some extent available on PDA), but it has also to handle multi-touch events. Therefore, on this platform, there is the need of introducing and modelling a different group of events, the so-called touch property events, which includes touch start (activated when a finger tap the screen surface), touch move (triggered when a finger moves on the surface), touch end (activated when a finger leaves the screen surface). In addition, the zoom gesture event (which is done through a multi-touch interaction) notifies that a zoom command has been recognized by the system and contains the scale factor that should be considered for zooming. Another peculiar characteristics of the iPhone is the existence of an accelerometer. In this case, the concrete user interface language has to support the specification of the current screen orientation and also to support the associated events. A more detailed description of MARIA is in [10].

5. TOOL SUPPORT

The authoring environment supporting MARIA and the associated method is a tool composed of three main sub-environments.

The first one, “Tasks-Services Binding Editor”, is aimed at supporting the associations between the tasks included in the model corresponding to the application to be developed and the Web services that the designer wants to include. In order to do this, the designer has to access the repository or the editor of task models and to access the URI where the Web services are made available. In addition, within this module, it is also possible to import some annotations associated to the Web services, and which provide further information about how the functionality included in the Web service will be finally rendered in the UI. Once such task-Web service association has been carried out, a dedicated module (“UI Composer/Transformer”), also through the use of possible annotations, is then be able to produce a first draft of the corresponding Abstract/Concrete User Interface (AUI/CUI) description in which all such various pieces of information (tasks, Web services, annotations) are exploited.

The AUI/CUI thus obtained are the output of the first module and, in turn, the main input to another module (the “User Interface Editor”) which is specifically aimed at supporting designers in refining the AUI/CUI depending on the specific needs and requirements of the application considered. Such User Interface Editor module exploits the “Transformation engine” module to obtain a CUI from an AUI, and then a user interface implementation (FUI) from a CUI description.

The rules included in the Transformation engine are defined in a specific model, the “Transformation model”, which allows for specifying the transformations that enable passing from a UI description to a more concrete one. The usefulness of having a Transformation Editor as a separate module lies in the enhanced

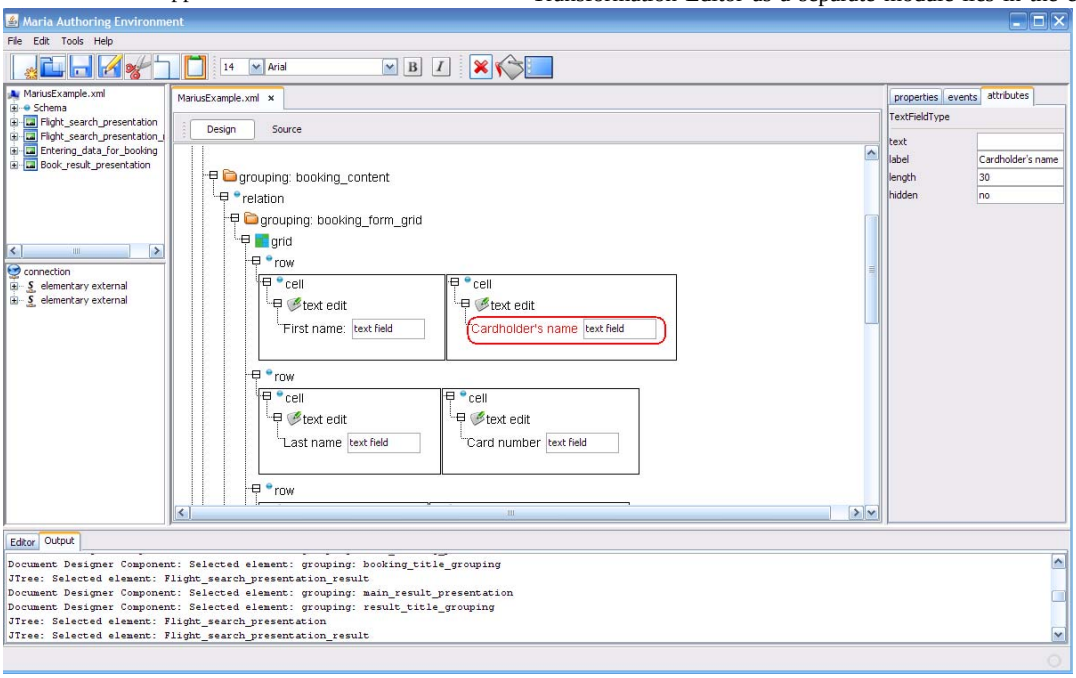

Figure 3. The Environment Supporting the Model-based Approach.

flexibility for designers to easily specify the transformations to be supported from time to time and avoid having them hardwired in the code.

The environment contains a set of generators, under development, each of which implements the transformations into specific platform-dependent description languages.

Figure 3 shows the interface for editing a CUI model: the left part contains an interactive tree diagram of the interactors and interactor compositions defined in the model. The central part is a direct manipulation interface for editing the model, where each interactor composition is a container for different interactor representations. All the elements of the model are classified according to their interaction semantics taking into account the target platform. For instance, a choice with low cardinality in a desktop CUI will be represented as a radiobutton, showing all the possible choices with the default option selected. All the interactor representations, which reflect their semantics, can be dragged from one container to another.

The right part of the interface is a toolbox for adding new interactors to the model (it shows only the allowed elements for the currently selected interactor).

The user can also edit the interactor attributes through the attribute list on the second tab, or set the event handlers through the event tab.

The environment also includes the Tasks-Services Binding Editor. In this case, the main part contains the CTT model using a hierarchical tree representation: the children of a node are the decomposition of the parent. The nodes at the same level are connected using different temporal operators. Each task is categorized as Abstraction, User, Interaction and System. The System tasks can be bound to a Web service operation from the repository on the right part, where different services with their operations and data types are listed. The CTT model enhanced with the Web service binding and annotations, which are represented on the left part, is used to generate the first AUI model for the application, which can be then modified by the designer using the AUI/CUI editor

6. CONCLUSIONS AND FUTURE WORK

In this paper we present our method for developing UIs for applications that are built by accessing Web Services. The described approach exploits a multi-layer framework of languages for describing UIs through a mix of bottom-up and top-down phases. We have introduced the corresponding authoring environment to ease the use of MARIA and associated transformations, which is under development. We have also discussed how the new MARIA language is able to support specification of flexible interactions exploiting such Web services and scripts, for then generate implementations for different types of devices.

Future work will be dedicated to extending the design space of user interface composition in order to address a wider set of cases, carrying out a usability test of the authoring environment under development to support the proposed approach, and applying the MARIA language in the OPEN EU project (http://www.ict-open.eu), which we coordinate, in order to support a richer set of migratory user interfaces. Such interfaces have the ability to follow the user across various interaction devices, adapting to the changing context and preserving their state even when users change devices.

7. ACKNOWLEDGMENTS

We gratefully acknowledge support from the EU ServFace Project (http://www.servface.eu).

8. 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] Chesta, C., Paterno, F., Santoro, C. (2004): Methods and Tools for Designing and Developing Usable Multi-Platform Interactive Applications. In Psychnology, 2 (1) pp. 123-139

[3] Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Lopez-Jaquero V. USIXML: A Language Supporting Multi-path Development of User Interfaces. EHCI/DS-VIS 2004: 200-220

[4] Mori G., Paternò F., Spano L. D.: Exploiting Web Services and Model-Based User Interfaces for Multi-device Access to Home Applications. Kingston, Canada, July 2008, DSV-IS 2008, Springer Verlag, LNCS, pp.181-193.

[5] Paternò F., Santoro C., Mantyjarvi J., Mori G., Sansone S., Authoring Pervasive MultiModal User Interfaces, International Journal of Web Engineering and Technology, Inderscience Publishers, 4(2) pp.235-261, 2008.

[6] Paternò F., Model-Based Design and Evaluation of Interactive Applications, Springer Verlag, 1999.

[7] Song, K., Lee, K.-H., 2008. Generating multimodal user interfaces for Web services, Interacting with Computers, Volume 20, Issues 4-5, September 2008, Pages 480-490

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

[9] 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.

[10] Paternò F., Santoro C., Spano L.D., MARIA: A Universal Declarative Language for Service-Oriented Applications in Ubiquitous Environments, ISTI Report, January 2009.