User Task-based Development of Multi-device Service-oriented Applications

Fabio Paternò, Carmen Santoro, Lucio Davide Spano
CNR-ISTI, HIIS Laboratory
{fabio.paterno, carmen.santoro, lucio.davide}@isti.cnr.it

ABSTRACT

In this paper, we discuss a method and the associated tool support able to exploit Web services in model-based user interface development, starting with the results of a task analysis phase, and using the content of Web service annotations. The resulting environment is a powerful support for developing multi-device interactive applications based on Web Services, since it is able to generate usable service front ends specified in a variety of implementation languages. This is achieved through connecting pre-existing Web services with the task model of the interactive application that has to be built. Then, the task model is used as a starting point for the generation of corresponding user interfaces descriptions at different abstraction levels, through a number of transformations that aim to preserve the usability of the corresponding models or implementations.

Keywords

Model-based user interface design, Web services, Annotations, Task Models.

INTRODUCTION

Interactive applications are making more and more use of application functionalities implemented through Web services. The clear distinction between the front-end and the Web services makes it possible to reuse such services across many interactive applications and supports a development model where the application and the service developers are different people. However, this distinction can make harder and longer the development of the interactive parts because their developers need to understand the Web service functionalities and the best way to interact with them.

In particular, we present a design and development environment, which supports the various possible abstraction levels for interactive systems (tasks, abstract and concrete user interface) [1] and it is able to derive service front-ends for various interactive platforms (desktop, mobile, vocal, multimodal). One of its main innovations with respect to previous model-based tools is the specific support for the development of interactive applications based on Web services, with the capability to also exploit associated user interface annotations, when available.

Damask [3] provides support for multi-device user interfaces using patterns and layers (which indicate what should be available in all platforms and what is specific to a given platform) but it does not provide any specific support for applications based on Web services and related user interface annotations. Model-driven design and deployment of service-enabled Web applications using WebML has been proposed as well (see for example [4]). Our work has a different focus since we propose an environment based on HCI models for generating usable service front ends, which can be implemented in a variety of implementation environments and not only for the Web.

In service-based applications the composition of Web services can occur at three levels (service, application, user interface). Our approach can potentially support all these levels. However, since the composition at the service level is already supported by well-known standards such as BPEL, we have so far focused our work on providing support for composition at the user interface and application levels.

Using annotated services can better support the development of interactive, service-based applications than traditional approaches. Annotations are hints provided by service developers in order to facilitate the development of service front ends, e.g. suggestion, form completion, validation, and synchronous field update. One proposal for a metamodel for user interface annotations is in [2]. However, the authors have not been able to indicate how to exploit such information when various abstraction levels are considered in the development of the service front ends.

This work is also useful from an End User Development (EUD) viewpoint because it allows the development of multi-device, service-based applications without having to know any of the many possible implementation languages. This is possible through the use of models that provide a conceptual view of the human-computer interaction hiding the complexity derived by the plethora of low-level implementation details. Thus, the end users that can benefit from the approach should have a capability to develop and understand models, which seems a reasonable assumption in the case of domain experts that often use their domain models in their work.

THE METHOD

In this Section we discuss the various parts characterising our approach. We first introduce the language that we use for logically specify user interfaces and then describe the method to derive user interfaces starting with task analysis and models.

The MARIA Language

MARIA [6] is a recent model-based language, which allows designers to specify abstract and concrete user interface languages according to the CAMELEON Reference framework [1] (an instance of the framework is in Figure 1). This language represents a step forward in this area because it provides abstractions also for describing modern Web 2.0 dynamic user interfaces, Web service accesses and novel interaction techniques such as those touch-based. In its current version it provides an abstract language independent of the interaction modalities and concrete languages for desktop, mobile, vocal, and multimodal (graphical and vocal composition) platforms. In general, concrete languages are dependent on the typical interaction resources of the target platform but independent of the implementation languages.

In MARIA an abstract user interface is composed of one or multiple presentations, a data model, and a set of external functions. Each presentation contains a number of user interface elements (interactors) and interactor compositions (indicating how to group or relate a set of interactors), a dialogue model describing the dynamic behaviour of such elements, and connections indicating when a change of presentation should occur. The interactors are classified in abstract terms: edit, selection, only_output, control, interactive description, .. Each interactor can be associated with a number of event handlers, which can change properties of other interactors or activate external functions.

While in graphical interfaces the concept of presentation can be easily mapped on that of a set of user interface elements perceivable at a given time (e.g. a page in the Web context), in the case of a vocal interface we consider a presentation as a set of communication between the vocal device and the user that can be considered as a logical unit, e.g. a dialogue supporting the collection of information regarding a user.

The Starting Points

Our approach has two distinct starting points, which usually are developed by different people: the task analysis and the Web services. The task analysis is the result of an interdisciplinary group involving end users, and its results are then formalized in a task model. The Web services are provided by software and service experts that want to make available some functionalities to interactive application developers. The connection between these two elements (task model and Web services) is obtained by the identification of the Web services that can implement some of such tasks. The task model is developed for providing a high-level description of the interactive application. It allows designers to obtain more refined descriptions of the interactive activities than workflow models, such as BPMN. In our work we use the ConcurTaskTrees notation (CTT) [5], which is an engineered notation for representing task models widely used, also for the public availability of editing and analysis tools. This notation explicitly represents through different icons whether a task is carried out by the user or the system or their interaction.

The novel design environment that we present is able to automatically access Web services, download and graphically present their WSDL description and, if available, their user interface annotations in the format previously introduced. Knowledge of the operation, parameters, data types of the Web services can be useful in the refinement of the task model as well. In particular, the Web service operations are functionalities automatically performed, and thus should be associated with basic system tasks: basic means that they are no longer decomposed in the task model, while system means that their execution is completely automatic (they are represented by a computer icon in the task model). Figure 1 shows the environment for supporting interactive association between tasks and services. The designers can first automatically import the

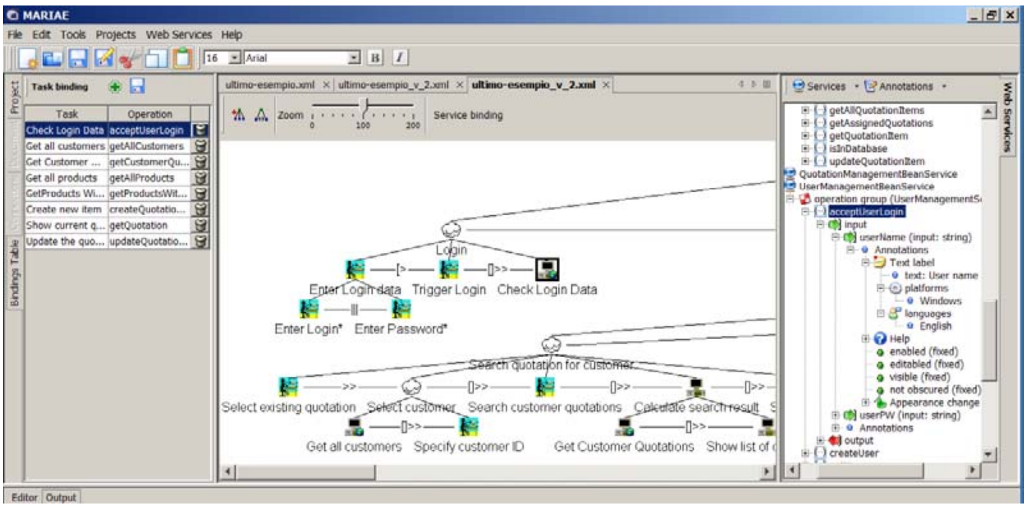

Figura 1: The Environment for Associating Tasks and Services.

service descriptions in the right side. The central part is dedicated to the editing of the task model. Then, they can interactive associate tasks and operations in the Web services. The results are shown in the two column table on the left side. For example, in Figure 1 it is possible to see that the Check Login Data task is associated with the AcceptUserLogin operation of the Web service called UserManagementBeanService. Such operation has also associated annotations concerning label, platform, and language.

The Transformations

Once the task model has been finalized, after the bindings of tasks with the relevant Web services by the designer, it can be the starting point for a series of transformations aiming to obtain the implementation of the corresponding interactive application. Figure 2 provides a graphical representation of the overall method in the case that the final results are two versions of the interactive application (one for desktop systems and one for IPhones), which will access the indicated Web services.

The information contained in the Web service annotations can be exploited in this transformation process at various abstraction levels. At the abstract user interface level, the annotations can specify groupings definition, input validation rules, mandatory/optional elements, data relations (conversions, units, enumerations,), languages.

At the concrete user interface level, the annotations can provide labels for input fields, content for help, error, warning messages, and indications for appearance rules (formats, design templates etc.).

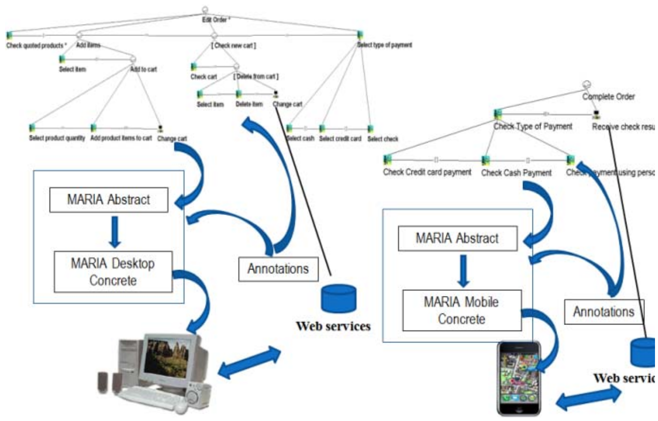

Figure 2: The Approach proposed.

One of the advantages of going through various abstraction levels is that the environment can be extended with limited effort to obtain the derivation of interactive applications adapted to different target interaction platforms (e.g. desktop, mobile, vocal, …) since in this case we have to apply different transformations only for the abstract-to-concrete and the concrete-to-implementation cases.

The Task-to-Abstract Interface Transformation

The Task-to-Abstract Interface Transformation is not trivial since it has to move from one representation in terms of tasks to one in terms of (abstract) user interface elements. The main aspects that are considered in the task model for this purpose are the hierarchical structure, the temporal operators, the task allocation, and the task types. Since the user interface is structured into presentations, the first step is to identify them from the task model. For this purpose the algorithm first builds a binary representation of the task model with each node annotated by the corresponding temporal operator. Then, the goal is to identify the presentation task sets (PTSs), which are the set of basic tasks that should be associated with a given presentation. The basic idea is that they are a set of tasks enabled in the same period of time. Thus, the binary tree is visited for this purpose taking into account the formal semantics of the CTT temporal operators.

After the identification of the abstract presentations, the interactors and the dialogue models associated to them have to be generated. For this purpose, three types of rules are applied to the task model description. Temporal relations among tasks indicate requirements for the UI dialogue model because the user actions should be enabled in such a way to follow the logical flow of the activities to perform. Task hierarchy provides information regarding grouping of UI elements: if one task is decomposed into subtasks, it is expected that the interactions associated with the subtasks are logically connected and this should be made perceivable to the user, thus a corresponding grouping composition operator should be specified in the abstract specification. Type of task provides useful information to identify the most suitable interaction technique for the type of activity to perform.

This transformation also exploits information specified by the associations between the Web services operations and the system tasks. In particular, usually the typical access to a Web service is modelled through three tasks: one interactive task for entering the user request, one system task for the Web service execution, and one system task for presenting the result of such execution.

In the abstract user interface, the interactors corresponding to interactive tasks, which provide the input information for the Web service operation, should contain a data model entity in their state for storing the value entered by the user. The type of such data entity is derived by the analysis of the WSDL, which also indicates the types of the Web service input data in XML format. Then, we need to include an activator interactor (this is the type of interactor that activates functionalities) for modelling the actual access to the Web service, and lastly we need an output interactor, which takes the result of the Web service execution and presents it in the user interface. Likewise, this interactor should contain a data model entity, whose type is derived from the WSDL, in this case by analyzing the data types of the output parameters.

The Other Transformations

The Abstract-to-Concrete Interface Transformation is much simpler than the previous one. Depending on the target platform, the specification is converted into the corresponding concrete description through an appropriate XSLT. Since the concrete language shares the structure of the abstract one and adds elements to it, which refine their description for the target platform, this transformation mainly consists in identifying which refinement, among the possible ones, to associate with each abstract element. It can happen that the language allows that one abstract element can be refined into multiple elements. In these cases the transformation has some selection rules to indicate which one to use depending on the value of a certain attribute. For example, depending on the cardinality of the possible choices a single choice can be refined either into a radio-button or into a pull-down menu. If even with the selection rules there are multiple possible target elements then the transformation selects one according to configuration properties, which can be modified by the designer.

The Concrete Interface -to- Implementation Transformation is a bit more complex than the previous one since implementation languages have a considerable amount of detail that needs to be provided. If we consider the case of a transformation from desktop concrete interface to XHTML, we have obtained it through XSLT as well. The transformation has been implemented by creating a template for each element of the source language, whose purpose is to create the corresponding code for the target element, and then to provide similar information for all the attributes that have been defined.

CONCLUSIONS and ACKNOWLEDGMENTS

In this paper we have presented a method, and the associated authoring environment for the model-based design of interactive applications based on Web services exploiting associated annotations. The approach allows development of multi-device service-based applications without knowing the various associated implementation languages. It is sufficient to design the interactive applications through the graphical representations of the models describing their features.

Future work will be dedicated to further facilitating the use of the design environment for non-professional software developers through more intuitive representations.

The current version of the tool can be downloaded at http://giove.isti.cnr.it/tools/Mariae/.

We thank the EU ICT STREP ServFace Project (http://www.servface.eu) for supporting this work.

REFERENCES

1. Calvary, G., Coutaz, J., Bouillon, L., Florins, M., Limbourg, Q., Marucci, L., Paternò, F., Santoro, C., Souchon, N., Thevenin, D., and Vanderdonckt, J. 2002. The CAMELEON reference framework. CAMELEON Project. Deliverable 1.1.
2. Janeiro J., Preußner, A., Springer, T., Schill, A., and Wauer. M. Improving the Development of Service-Based Applications through Service Annotations. Proceedings of IADIS WWW/Internet, 2009.
3. Lin J., Landay J.: Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. CHI 2008: 1313-1322.
4. Manolescu I., Brambilla M., Ceri S., Comai S., Fraternali P.: Model-driven design and deployment of service-enabled Web applications. ACM Trans. Internet Techn. 5(3): 439-479 (2005).
5. Paternò F.. Model-Based Design and Evaluation of Interactive Applications. Springer-Verlag, 2000.
6. Paternò F., Santoro C., Spano L.D., "MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment", ACM Transactions on Computer-Human Interaction, Vol.16, N.4, November 2009, pp.19:1-19:30.