Exploiting Web Service Annotations in Model-based User Interface Development

Fabio Paternò, Carmen Santoro, Lucio Davide Spano
ISTI-CNR, HIIS Lab, Via Moruzzi 1,
56124 Pisa, Italy
{Fabio.Paterno, Carmen.Santoro, Lucio.Davide.Spano}@isti.cnr.it

ABSTRACT
In this paper we present a method and the associated tool support able to exploit the content of Web service annotations in model-based user interface design and development. We also show an example application of the proposed approach.

Categories and Subject Descriptors
H5.2. Information interfaces and presentation (e.g., HCI).

General Terms
Design, Human Factors, Languages

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. Web service annotations can provide useful support in order to overcome this problem. They have been provided for many purposes, often to support the specification of the semantics of service operations, parameters and return values based on knowledge represented in ontologies. In this work we consider annotations whose purpose is to provide hints for the user interface development. Thus, they represent suggestions that Web service developers provide to facilitate the work of developers of service-based interactive applications.

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]. One of its main innovations, with respect to previous model-based approaches, is the specific support for the development of interactive applications based on Web services, with the capability to also exploit associated user interface annotations.

In general, the composition of Web services can occur at three levels: *service level*, the output of a Web service is directly the input for another one; *application level*, some application code, integrated in the service front-end, accesses a Web service, takes its input, processes it and then accesses another one; *user interface level*, the user can access multiple Web services directly from the user interface without the support of any application logic.

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.

Some work has been dedicated to the generation of user interfaces for Web services but without exploiting model-based approaches, see for example [7]. In [8] there is a proposal to extend service descriptions with model-based user interface information. For this purpose the WSDL description is converted to OWL-S format, which is augmented with a hierarchical task model and a layout model. We follow a different approach, since the service can be used in different applications and contexts we do not want to associate a fixed interactive application model to them. We prefer to use Web service annotations providing cues regarding the possible corresponding user interface, still leaving to the front-end developer the possibility to choose how to exploit such cues. 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. The use of logical user interface languages to support service composition and user interface generation is explored in [2]. Our solution is able to provide a more systematic solution to such issues because it exploits task

EICS'10, June 19–23, 2010, Berlin, Germany.

models that provide an integrated view of interaction and functional aspects.

In the paper, we describe the method and the tool for supporting model-based development of interactive applications based on Web services. The approach proposed exploits the various possible abstraction levels (task, abstract interface, concrete interface), and such abstractions can also exploit some information taken from the Web service annotations. In particular, we provide background information useful to better understand how our approach works: the main features of the modelling language used and of the user interface annotations exploited. Then, we describe the method proposed and the associated transformations, which are followed by the description of the authoring environment, and an example application. Lastly, we draw some conclusions and provide indications for future work.

BACKGROUND

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]. This language represents a step forward in this area because it provides abstractions also for describing modern Web 2.0 dynamic user interfaces and Web service accesses. In its first version it provides an abstract language independent of the interaction modalities and concrete languages for graphical desktop, mobile, and vocal 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, etc.. 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 defined as 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 interactions 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.

In general, using annotated services can provide useful information. For example, they can provide additional layout information, such as Group & Order and Mandatory Field. This increases the readability of a UI, since important UI elements are displayed first, semantically related elements are arranged nearby, and unimportant fields can be omitted. Annotations provide content awareness and field semantics, e.g. MIME Type, Special Data Type, which lead to an improved presentation of the content, e.g. a date can be presented as a calendar, an address on a map, or a video in a video player.

Annotations limit the need of programming while still being able to support complex communication patterns and additional functionality, e.g. suggestion, form completion, validation, and synchronous field update. Annotations also make relations between service elements explicit, e.g. semantic data type relation or bundle.

The annotation meta-model that we have considered in this work [3] contains 22 annotations, which can be logically grouped into three categories visual, behavioural, and relational. Visual annotations influence the appearance of the generated UI, they are: Label, Units, Contextual Help, Multimedia Content, Format, Special Data Type, Visual Property, Enum, Design Template, and Mime-Type. Behavioural annotations provide information that can influence the user interface behaviour and are Group & Order, Validation, Suggestion, Default Value, Example Data, Synchronous Field Update, Mandatory Field, and Form Completion. Relational annotations describe the relations between annotated elements and are: Semantic Data Type Relation, Bundle, Conversion, Authentication, and Appearance Change Rule. However, Janeiro et al. [3] have not proposed solutions to exploit such annotations when various abstraction levels are used to support design and development of user interfaces. How to fill this gap is addressed in this paper.

THE METHOD

Our approach exploits all the abstraction levels considered in the CAMELEON Reference Framework (task, abstract interface, concrete interface). The starting point is the identification of the tasks to support and the Web services that can implement application functionalities. Then, a task model is developed for providing a high-level description of the interactive application. Task models allow 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.

In addition, 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. Our solution is able to extract semantic information from this type of description so that knowledge of the operations, 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).

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. 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,), and 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.).

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

This 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. Such a procedure can include the same task in more than one PTS, the rationale is that the same task may be supported at different points during the application execution. In addition, the first splitting can produce sets with very low cardinality. In order to avoid having such presentations supporting a very limited number of user interface elements, the designer can apply some predefined heuristics in order to obtain better results.

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.

Compositions and elements are included in presentations and determine their structure. In addition, entire presentations can be composed through connections, which specify the interactions that trigger the navigation and the corresponding target presentations.

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 this 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 contains a data model entity, whose type is derived from the WSDL, in this case by analyzing the data types of the output parameters.

The Abstract-to-Concrete Interface Transformation

This 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

This 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.

It is also possible to specify mappings working only on attributes of the source language and associating them with corresponding elements in the target implementation language. For example, the connection element in the abstract interface has an attribute that specifies the target presentation. This attribute is mapped to the href attribute of the a tag in XHTML.

TOOL SUPPORT for EXPLOITING ANNOTATIONS

The MARIAE tool provides many features for editing specifications at various abstraction levels and transform them. In addition, it provides support for the method for designing interactive applications based on Web services. There is a part of the tool dedicated to the management of Web services and associated notations. It allows the designer to specify the URI of Web services and it automatically downloads its WSDL description and graphically represents it in the right side of the user interface (see Figure 1).

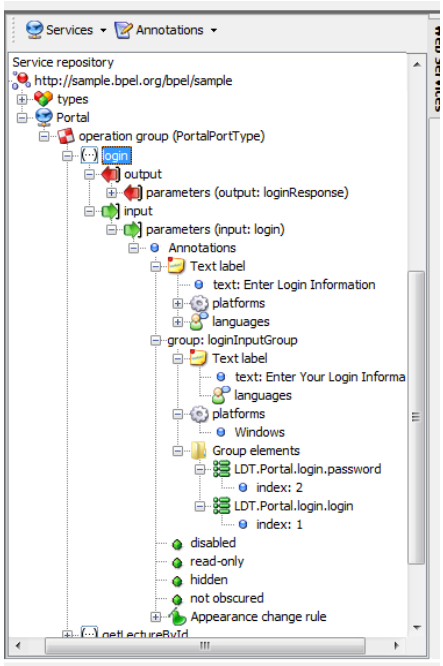

Figure 1: The part of the tool dedicated to showing information related to the Web services and their annotations.

If an associated user interface annotation file exists then the information that it contains is used to enrich the WSDL representation. In the example in Figure 1 the annotation provides indications for labels for some parameters, grouping of others, and others. When both the task model and the Web service are available, the designer can interactively map operations with tasks in the model. During this work it may be necessary to refine the task model in order to reach a level of decomposition in which each operation of the Web services is associated with a basic task. Obviously, one task model can refer to multiple Web services, thus providing an abstract description of how they should be composed in the interactive application under development.

When the tool is in editing mode, it is possible to have multiple tabs in the central area, one for each open specification. On the left side there is the indication of the presentations and the associated elements in an interactive tree-like view. On the right side there is the list of elements that can be imported in the specification by drag-and-drop depending on the element selected. The representation in the central area aims to be more immediately understandable of the XML textual description. Thus, it represents the interface elements with icons, while in the right area it is possible to visualize on request the attributes or the events associated with the selected element in the central part.

AN EXAMPLE

In order to show how the proposed approach works, we consider an example in which a sales representative user has to access the company system for handling orders/quotes in order to update the price quote for a certain customer. Thus, s/he specifies his/her username and password for accessing the system by calling the related Web service operation (acceptUserLogin of the AccountManagementBeanService WS, we will see further details later on). Then, s/he specifies the customer ID in order to get the list of quotations currently associated with that customer from the system. This list is delivered by a Web service having the associated operation getCustomerQuotations. Then, the user has to select the quotation that s/he plans to update.

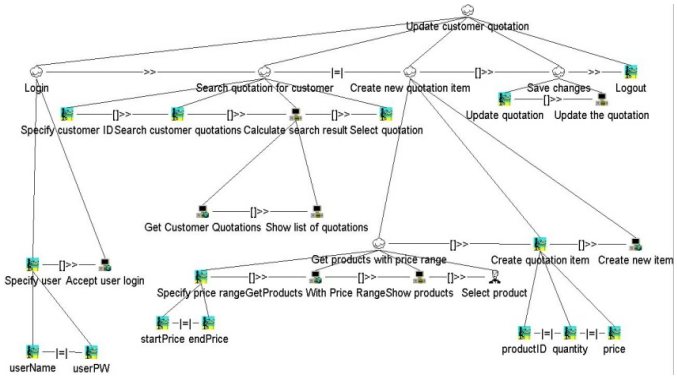

Figure 2: The task model of the example

Indeed, s/he wants to add another product to that quotation and, for this purpose, s/he specifies a certain range of prices in which to search the product of interest to be added to the quotation. The system delivers a list of products in that range (getProductsWithPriceRange operation) and displays

this list on the user interface, so that the user is able to select the product of interest. With the gathered information the user is now ready to create another quotation item for updating the current quotation. In order to do this, s/he has to specify the productID, the quantity and the price. At this point the system can create another quotation item (this is done by calling the associated WS operation createQuotationItem of the WS QuotationItemManagementBeanService) and then the user can update the quotation and finally logout from the system.

The associated task model is illustrated in Figure 2. In this example the system tasks that are connected with Web Services (Accept user login, GetCustomerQuotations, GetProductsWithPriceRange, CreateNewItem) are identified by the icon used for system tasks modified by the addition of a small circular shape. The system has to manage a number of functionalities, which are provided by operations that are part of different services that could be offered by different providers and were not originally designed to work together. These services are:

- *UserManagementBean Service*: allows the user to access the system. It supports the creation and handling of a user profile in the system (e.g. by specifying a username and a password).
- *AccountManagementBean Service*: manages all information about business partners. This service allows handling the creation, retrieval, update and deletion of customer data with associated payments and orders.
- *ProductManagementBean Service*: supports the management of the product portfolio, acting as a connection module towards the production process (e.g. the inventory). Typically, this service supports the creation, retrieval, update and deletion of product data.
- *QuotationItemManagementBean Service*: manages processing of quotations to customers and supports typical operations like the creation, retrieval, update and deletion of quotations and quotation items.

The application developer binds the system tasks to the Web services operations using the Web service panel of the editor (in particular the AcceptUserLogin operation is bound to the Accept user login task, the GetCustomerQuotations operation is bound to the Get_Customer Quotations task).

Regarding the association between the input parameter of the operations and the interactive tasks, the tool is able to detect that a specific interaction task provides the input for a WS operation (in this example the username and password should be specified to allow the user to access the system).

The authoring environment supports such specifications through the Web service browser where the developer can specify the URL of the WSDL file (which can be available remotely or locally) for inspecting operations (with input and output parameters) and data types defined for invoking the service. Then, s/he can load annotations for the selected Web service (if any) for supporting the logical user interface generation process.

In this example we use a number of annotations for the operations, annotations generally regarding visual aspects like the format and label for the various input fields, as shown for the case of the AcceptUserLogin operation:

AcceptUserLogin [input] userPWD: string (label= Password)

AcceptUserLogin [input] userName: string (label= User Name)

The first transformation creates a first draft of the AUI by calculating the Presentation Task Sets, which are the sets of tasks enabled over the same period of time according to the temporal relations defined in the task model. Such presentation task sets can be used to identify the corresponding presentations in the user interface when we transform the task model into an abstract user interface description. In our example the result of this transformation generates more than ten Presentation Task Sets. The designer can decide to reduce this number (as happens in our example) by applying some heuristics which are supported by the tool. In this case the designer, by using a heuristic making it possible to combine task sets having only one element into other PTSs, is eventually able to obtain three presentation task sets, which are then associated with abstract user interface presentations. In particular, the first obtained abstract presentation allows the user to access the system, by specifying the username and password and then logging into the system. Another presentation allows the user to do the search for a list of products in a certain range of price, one of these products will then be selected by the user to be added to a certain quotation that will be updated. The third presentation allows for performing a search in the list of quotations currently associated with a specific customer: the user will select the desired quotation and then add to this quotation a new item related to the product previously selected.

Figure 3: The Presentation_3 in the Abstract UI

Figure 3 shows the last presentation (Presentation_3) as it is automatically generated by the tool, before the designer edits it in order to customise it according his/her

necessities. As you can see, this presentation is composed of three groupings and one navigator object for logging out of the system. The three grouping expressions are the following: one (*Search_quotation_for_customer*) is devoted to specifying the customer and getting the associated quotations; one structured grouping (*Create_new_quotation_item*) contains two other groupings to i) search the products in a certain price range (Get_products_with_price_range) and ii) create a new quotation item depending on the characteristics of the product (Create_quotation_item); the third grouping (*Save_changes*) expression is devoted to saving the changes to the quotation (namely, adding the new item to the quotation).

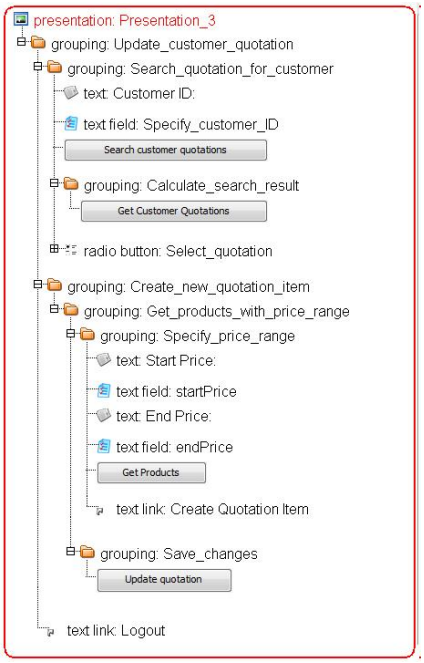

Figure 4: Concrete UI: Presentation_3 after the designer editing

Once we have obtained the abstract user interface description, we can move to the next transformation and then obtain a concrete UI for a specific platform (the desktop in this case). This phase is carried out by a number of mappings describing how the various abstract UI objects should be mapped at the concrete level. Such mappings are implemented by an XSLT file. The XSLT transformation can specify that, for example, a single choice selection object should be translated in the desktop platform to a radiobutton. In addition, the annotations can be used in this phase for generating a suitable CUI since the tool is able to exploit them. For instance, the tool is able to automatically include the text labels associated with the text input fields according to the annotations.

Of course, the designer can further change these labels. In addition, in this case the designer has made further modifications to the page: now the button not only activates the Web service operation associated for logging into the system, but also allows the user to move to the next presentation. This has been done in the tool by specifying that the activator element also has to act as a connection between two presentations. Indeed, also at this point the designer can further edit the concrete interface that is automatically generated. Figure 4 shows the Presentation_3 after the designer has performed some editing.

From the concrete description it is possible to automatically generate the Web pages corresponding to its implementation.

CONCLUSIONS and FUTURE WORK

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. First evaluation exercises have given positive results but we cannot describe them for lack of space.

More systematic evaluation is planned in the near future in order to improve its usability and make it suitable also for non-professional developers.

The tool is publicly available for download at http://giove.isti.cnr.it/tools/Mariae/

We thank the EU 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. Dery-Pinna A.-M., Joffroy C., Renevier P., Riveill M., and Vergoni C., ALIAS: A Set of Abstract Languages for User Interface Assembly. Proceedings Software Engineering and Applications (SEA 2008). November 16 – 18, 2008. Orlando, Florida, USA.
3. 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.
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.
7. Song, K., Lee, K.-H., 2008. Generating multimodal user interfaces for Web services, Interacting with Computers, Volume 20, Issues 4-5, September 2008, pp. 480-490.
8. 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, LNCS 4940, pp. 447-464, Springer Verlag.