Improving Task-driven Software Development Approaches for Creating Service-based Interactive Applications by using Annotated Web Services

Marius Feldmann, Gerald Hübsch, Thomas Springer and Alexander Schill
Technische Universität Dresden, Department of Computer Science,
Institute for Systems Architecture, Computer Networks Group
Dresden, Germany
{marius.feldmann, gerald.huebsch, thomas.springer, alexander.schill}@tu-dresden.de

***Abstract*—The aspect of human interaction becomes more and more relevant for service-oriented applications. On one hand, current development approaches support the composition of services but neglect the creation of user interfaces. On the other hand, task-based modeling of user interactions is well established but poorly integrated into service-oriented development. In this paper we propose an integrated modelling approach for the task-driven development of interactive service-oriented applications. Our central concept is the introduction of annotations as part of the Web service description. These annotations cover information relevant for the semi-automatic generation of task models and user interfaces. We present the annotation concept as well as concepts for the integration of the annotations into a task-driven software development methodology.**

I. Introduction

Due to the increasing complexity of applications and the large number of user interface (UI) technologies and toolkits that exist today, designing interactive applications is still a challenge for application developers. A promising approach to handle this complexity is task-driven software development [1], [2]. The concept of task-driven software development is to describe an application based on a task model. A task model consists of tasks which the user (user tasks), the system (system tasks), or the user and the system (interaction tasks) must perform in order to achieve a particular goal. It also captures the order in which these tasks must be performed. In a fine-grained task model, each system task corresponds to a single function call and each interaction task to a single man-machine dialog. The strength of task modeling is its technology-independence and its ability to capture user interaction and application logic in one model which can eventually be transformed into executable code.

In the terminology of model-driven architecture, a task model is a domain-specific language for describing interactive applications. Three artifacts can be derived from a task model through transformations: a *control flow* with hooks for dialogs and function calls, an *initial version of the application's UI*, and a *code skeleton* of the application logic [2]. Only the control flow can be transformed into executable code without further modifications. The initial version of the UI lacks layout definitions, usability optimizations and the selection of appropriate UI controls. This information has to be added by the developer in a cumbersome and time-intense UI refinement process. The code skeleton is only a partial implementation of the application logic which must be completed by writing code for algorithms, database queries, etc. In service-oriented applications, this application logic is implemented by service operations. They could provide the implementation of system tasks if task models would be extended towards support for binding service operations to system tasks.

Task-driven software development can furthermore profit from *service annotations*. Service annotations enrich service descriptions for example written in WSDL [3] or WADL [4] with information beyond a purely functional specification. Approaches for generating suitable UIs for web services based on service annotations already exist, for example WSGUI [5] and Dynvoker [6]. They implement an inference mechanism to generate a form-based UI from WSDL files and annotations that contain hints to generate layout, UI controls and labels.

Since the transformation of interaction tasks into an initial version of the UI is based on a similar inference mechanism, service annotations could substantially increase the quality of the initial UI and reduce the manual effort for the UI refinement. The concept of service annotations is by far not limited to aspects of UI generation. For example, annotations can also be used to check the completeness of task models based on dependencies between service operations.

In the following, we present work-in-progress that aims to extend task-driven software development towards annotated services based on the task model and the model transformations developed in the EMODE project [7]. We first discuss our concepts for service annotations. Afterwards, we show how service annotations can be exploited in task modelling and UI generation. The paper concludes with a summary and an outlook on future work.

II. Service Annotations

The central idea behind Web services is the reuse of functionality. In most development methodologies for building interactive applications upon a service infrastructure, reusability is still reduced to the functional core encapsulated within the services. The concept of Web service annotations aims to include further reusable information fragments that are not

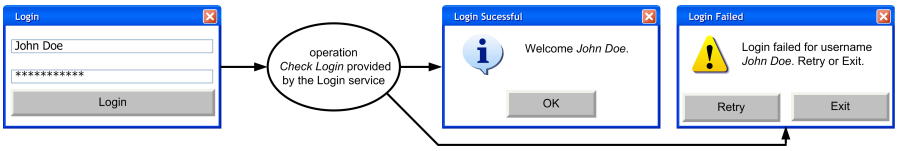

Figure 1. Dialogs related to the input and output of a login service

covered by the functional interface description and thus - when applied to the apropriate methodology - to profit from faster and simplyfied application development.

The mechanism for service annotations has been build upon the approach used within the Dynvoker project. In this approach a service annotation file contains UI related information that is associated to operations and parameters within the service's functional interface description by using XPath expressions. Due to the fact that the annotation language used in Dynvoker (GUIDD) only offers basic UI related information, it has been extended. In order to find apropriate annotations, several service-based interactive applications have been analysed to detect reusable patterns. During our investigations more then 25 different reusable types of information fragments and thus of annotations have been identified.

The annotations introduced by our approach are categorized into UI related annotations and annotations describing pre- and postconditions of a service operation. The UI related annotations semselfs are classified into annotations describing the layout and structure of the UI and annotations describing the behavior of the UI. In general, each operation of a service can be associated with one input dialog for collecting the input parameters and several output dialogs which are presented depending on the result value returned by the operation. The UI related annotations are used to give a reusable description of these dialogs. Due to the fact that the annotations are created on the level of Abstract User Interfaces, these description can be reused for developing applications for different platforms and devices. Thus for example the layout and structure definitions do not cover concrete position or size specifications but the capability to group input fields or determine their order.

Fig. 1 shows a simple example of a login service operation. The login service has got an input dialog to request credentials and depending on their validity one of two possible output dialogs is displayed. The dialog for requesting the credentials can not be derived completely from the functional interface description of the service operation. It can be determined only that two input parameters for values of type String are necessary. Annotations for *data type definitions* are used in the example to specify that the first parameter should only include characters and that the second one is of the semantic type *password*. Furthermore a button with the label *Login* is associated with the operation. A *ordering definition* that is one of several *layouting definitions* is used to indicate that this button should be displayed after the two input fields. The return value of the service operation is annotated by *return value dependend output dialogs*. Both dialogs contain a text indicating the result of the login operation. In the case of a failed login two buttons are indicated within the annotations. The button with the label *Retry* contains a *navigation definition* pointing to the login operation thus resulting in a further invocation of the first dialog when the button is pressed. Based on the described information within the annotations a valid UI can be created automatically.

The annotations within the behavior definition are mainly intended to increase the usability of the created UI. A very often applied annotation is the "validation annotation" that provides a rule expressed by a regular expression or by the Object Constraints Language (OCL). They can be used to provide direct feedback to a user if the input does not conform to one of these given rules. The decision to use OCL rules has been made to also describe dependencies between parameters. Thus it is for example possible to point out that the value of a variable "Date dateFlightTo" has to be smaller then a second variable's value "Date dateFlightBack" - and thus the first date input has to be before the second date. A further example for a behavior annotation is the "suggestion" information. It enables to associate an input parameter of a Web service with another service that returns a concrete proposal for an input value. It is obvious that such a suggestion service is only available in rare cases. Thus our research also aims to provide guidlines for extending Web service infrastructures to improve the development of interactive applications based on such infrastructures.

Besides the annotations used in the example to describe UIs for single services, annotations have been identified to define relations between parameters or return values of different operations. The usefulness of these annotations is explained in the next two sections that discuss how the UI related annotations and the annotations describing pre- and postconditions of a service operation can be used to derive a task-model and associated UIs (semi-)automatically.

In order to attach service annotations in a simple manner to their associated service, a specific editor for creating them is currently under development. It offers the possibility to import a functional interface description of a Web service and to assign values for the various annotations to the interface description itself, to the service operations, operation parameters and defined data types. The values can be assigned on one hand via a simple key-value mechanism and on the other hand in a visual mode. After parsing the functional interface description all UI interactors (e.g. text fields) are displayed and offer the possibility to instantiate associated annotations such as labels, validation information (e.g. by an regular expression), default values or MIME type information. For expressing the vocabulary of the annotations, an Ecore model has been introduced.

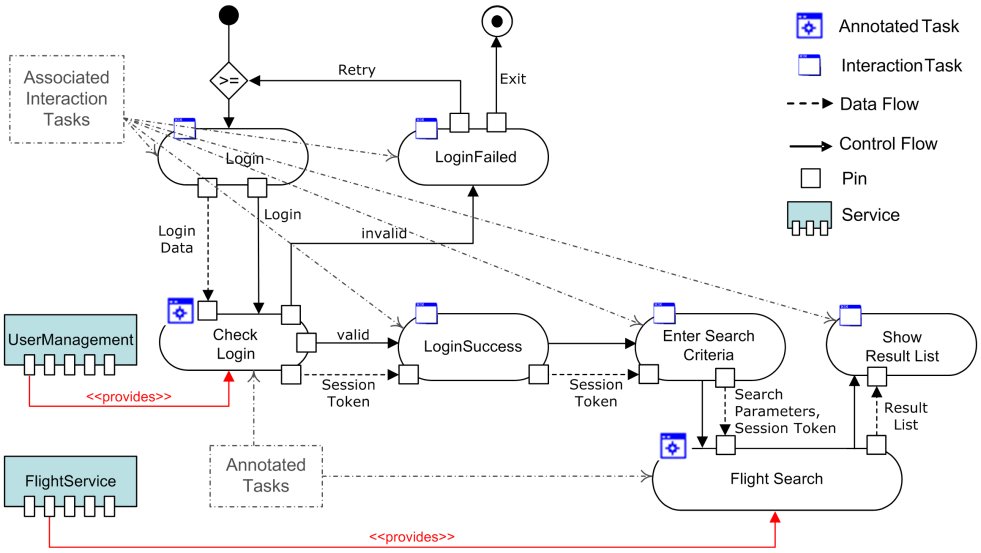

Figure 2. A task-model derived from annotated tasks

The annotations are serialized by using the XML Metadata Interchange (XMI) representation. The annotation file holds a referece to the associated service interface description. The annotation editor offers the possibility to store this file locally or to publish it to an annotation repository where annotations for various services are collected.

III. Exploiting Annotations in Task Modeling

Annotated services can be integrated seamlessly into a task-driven development approach by introducing the concept of *annotated tasks* into task models. An annotated task is a system task that is implemented by an operation of an annotated service. It is created when the operation of an annotated service is imported into a task model. After the import step, interaction tasks are *automatically* added to the task model and associated with the annotated task. They represent user interactions that must be performed before the annotated task can be invoked and for the presentation of the result returned by the annotated task. The association is established through data flows (specifying the exchange of data between tasks) and control flows (specifying the order and the conditions for task execution). The necessary information for the creation of the interaction tasks and their association is contained completely in the annotations of the service operation and therefore requires no additional action from the developer of the task model.

Fig. 2 provides a comprehensible example to demonstrate how the automatic creation of interaction tasks based on annotations increases the efficiency of modeling an interactive application. The example shows a simple excerpt of the task model for a flight booking application that allows the user to search flights after successful login. The user must first enter login information (Interaction Task *Login*). The login authentication (Annotated Task *Check Login*) is implemented by an operation of the User Management service. If the authentication failed, an error message is shown (Interaction Task *Login Failed*). Else, a session token is returned and a 'Welcome' message (Interaction Task *Login Success*) is displayed. The session token is required to invoke the flight search operation (Annotated Task *Flight Search*) provided by the Flight Service. After entering the search criteria (Interaction Task *Enter Search Criteria*), Flight Search returns a result list which is displayed to the user (Interaction Task *Show Result List*). With the help of the service annotations for the flight search operation, the task model in Fig. 2 can be created with minimal effort. The developer just needs to import the flight search service operation into the task model. Note that the login operation does not need to be imported manually. The flight search operation is automatically transformed into the annotated task *Flight Search*. Its annotation is parsed and the interaction tasks associated with Flight Search (*Enter Search Criteria*, *Show Result List*) are added automatically. Their associations through data flows and control flows are also added automatically.

Preconditions for annotated tasks can be utilized to simplify and automate task modelling beyond the generation of interaction tasks. The fulfillment of preconditions can be evaluated at design time to check the completeness of the task model. If a task model is evaluated as being incomplete, it can be completed automatically by hints in the annotations. In our example, the flight search operation is annotated with a precondition. The precondition specifies that a session token is required to invoke the operation. It furthermore specifies that the required session token is provided by the login operation of the User Management service. Based on this information, the task model can be completed without a developer's intervention by automatically importing the login operation. The interaction tasks associated with *Check Login* and their associations can be derived from the annotations, ultimately producing the task model shown in Fig. 2.

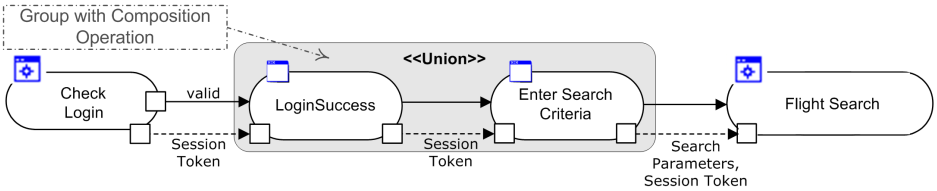

Figure 3. Group with Union operation in a fragment of the flight booking task model

IV. Generating User Interfaces from Task Models

Service annotations can be exploited to significantly enhance the initial UI which is generated by the tranformation of interaction tasks into dialogs. Without annotations, the initial UI generated from a task model lacks suitable layout definitions, usability optimizations and the selection of appropriate UI controls. This is due to the fact that the task model does not contain the necessary information. It must be added manually by a time-intense refinment of the initial UI. In case of annotated tasks, this information is contained in the UI related service annotations (see sect. II). It can therefore be automatically evaluated in the transformation process. This reduces the need and the time for a manual refinement of dialogs generated from interaction tasks associated with annotations to a minimum.

Another application area for annotations in the area of UI generation from tasks models is a better support for the grouping of interaction tasks. In our approach, one interaction task is transformed into one dialog by default. The default behaviour can be overwritten by combining interaction tasks into groups. The UI controls that are generated for all interaction tasks in a group are added to one and the same dialog, giving the developer control over the transformation result. This simple merging approach produces dialogs that can contain duplicates, i.e. UI controls controls with identical functions, that have to be removed manually in the refinement process.

A solution to avoid the manual removal of duplicates is to apply the composition operation *Union* proposed in [8] for groups of interactions tasks. The Union operation composes two dialogs into one and avoids duplicated UI controls, given that duplicates can be identified. In our case, duplicated UI controls can be identified based on annotations that describe relations between parameters of different operations. We are therefore able to provide the Union composition operation for groups of interaction tasks.

In the example in Fig. 3), the Union operation is added to the group containing *Login Success* and *Enter Search Criteria*. It guarantees that the dialog generated by the combination of the 'Welcome' message 'search criteria' form is free from duplicates. Again it shows that the annotations can reduce the effort for manual refinement significantly.

V. Conclusion and Outlook

In this work-in-progress paper the combination of the concept of annotated services and the task-driven development approach has been proposed. We introduced the notion of annotated tasks to integrate annotated services in task models and to derive and associate interaction tasks automatically. This approach has been evaluated by extending the EMODE task model and transformations. Based on examples it has been shown how service annotations can be exploited to increase the efficiency of task modelling significantly. Furthermore, ideas have been presented to reduce the efforts for UI refinement steps by including hints for UI creation within the service annotations.

In the future we will extend our approach to complete the list of annotations usefully in the scope of our concepts. We will also improve the integration of our concepts into the EMODE tools. An in-depth analysis to qualify the amount of source code generated automatically in typical service-based interactive applications by applying the service annotation concept is currently under consideration. The application of the service annotation concept to end-user development methodologies is currently a second parallel research focus.

References

[1] F. Paterno, C. Mancini, and S. Meniconi, "ConcurTaskTrees: A diagrammatic notation for specifying task models," in *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*, London, UK, 1997, pp. 362–369.

[2] M. Heinrich, M. Winkler, H. Steidelmüller, M. Zabelt, A. Behring, R. Neumerkel, and A. Strunk, "MDA applied: A task-model driven tool chain for multimodal applications," in *Proceedings of the 6th International workshop on TAsk MOdels and DIAgrams (TAMODIA 2007)*, 2007.

[3] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL) 1.1," W3C, Tech. Rep., 2001.

[4] M. Hadley, "Web application description language (WADL)," Sun Microsystems, Report TR-2006-153, 2006.

[5] M. Kassoff, D. Kato, and W. Mohsin, "Creating GUIs for Web Services," *IEEE Internet Computing*, vol. 7, no. 4, pp. 66–73, September/October 2003.

[6] J. Spillner, M. Feldmann, I. Braun, T. Springer, and A. Schill, "Ad-hoc usage of web services with dynvoker," in *ServiceWave 2008*, ser. LNCS 5377, P. Mähönnen, K. Pohl, and T. Priol, Eds., NESSI. Madrid, Spanien: Springer, 11 2008, pp. 208–219.

[7] M. Winkler, M. Heinrich, A. Behring, J. Steinmetz, and W. Dargie, "EMODE - ein Ansatz zur werkzeugunterstützten Modellierung multimodaler, adaptiver Benutzerschnittstellen," ser. Lecture Notes in Informatics, K.-H. R. M. R. Rainer Koschke, Otthein Herzog, Ed. Gesellschaft für Informatik e.V. (GI), 9 2007, beiträge der 37. Jahrestagung der Gesellschaft für Informatik e.V. (GI).

[8] S. Lepreux, J. Vanderdonckt, and B. Michotte, "Visual design of user interfaces by (de)composition," in *Interactive Systems. Design, Specification, and Verification*. Springer, 2007, pp. 157–170.