Rückert J, Nguyen D-K, Paech B (2007) "An adaptive dialog core for user interfaces"

Presented at:
ACOMP 2007 - International Workshop on Advanced Computing and Applications, Ho Chi Minh City/Vietnam, March 14-16, 2007

An adaptive dialog core for user interfaces

Jürgen Rückert, Dinh-Khoa Nguyen, Barbara Paech

Institute of Computer Science, University of Heidelberg, Im Neuenheimer Feld 326
69120 Heidelberg, Germany
Rueckert@informatik.uni-heidelberg.de
Dinh_Khoa.Nguyen@urz.uni-heidelberg.de
Paech@informatik.uni-heidelberg.de

Abstract. User interfaces are the bridge between users and usually one application core at runtime. In order to be integrated with new application cores the user interface must be recompiled, repackaged and redeployed so that it works correctly with the new cores and their specific technology. In this article we outline our solution and present a detailed design model of a user interface which can be configured dynamically and can integrate required application cores at runtime.

Keywords: user interface, application core (back-end, server, web service), dynamical configuration and adaptation, dialog core.

1 Introduction

The service-oriented architecture (SOA) becomes more and more popular in the field of business applications. However so far, developers have discussed mostly the construction of web services and their interoperability with other web services, namely the back-end of business applications. But there are a lot of complementary technologies that could by applied as back-ends for service-oriented applications. In this paper we introduce the idea of an adaptive dialog core, which allows runtime integration of several back-end technologies at the same time.

In section 2 we outline the concept of our adaptive dialog core. In section 3 we detail the concept by presenting the design model and some design decisions. In section 4 we summarize the article and give a short outlook in which areas the concept might be useful and should develop in future.

2 Outlining the Adaptive Dialog Core

In this section we present an application scenario in which the adaptive dialog core can be applied (2.1), we explain the general role of a dialog core in user interface architectures (2.2), we present the components of our adaptive dialog core (2.3) and we finally specify the targeted flexibility of the adaptive dialog core (2.4).

2.1 Application Scenario

The motivation of this paper comes from the lecture *Software Engineering of modern applications – component-based, service-oriented or mobile syste*ms given by Prof. Barbara Paech at the University of Heidelberg [2] [3]. During this lecture the students realize the application scenario *Auctions* with four different architectures. The *Auctions* application scenario (similar to eBay) enables users to take part in auctions in order to buy or sell items. The web application consists of a user interface (UI) and an application core. In contrast to the mostly constant UI, the application core is to be realized with servers and web services of four different technologies:

- JSE [5] Server: foreseen to hold application data in memory only
- JEE [6] Server: foreseen to persist application data in a database
- AXIS [8] Web Service: foreseen to hold application data in memory only
- JEE Web Service [6]: foreseen to persist application data in a database

In the lecture the students create four shipments each of them containing both the UI and the application core. As expected, most of the UI is reused in all four shipments and only a small piece of the UI, the one that communicates with the servers and web services, is exchanged, namely the dialog core.

2.2 Dialog Core

The UI is built based on the reference architecture of UIs introduced in [2], which is shown in figure 1. The most important component in this model is the dialog core, which is the interface component to the application core. The dialog core offers an interface DE for receiving dialog events from the presentations (views). It uses the interface AF of an application core for consuming offered functionalities.

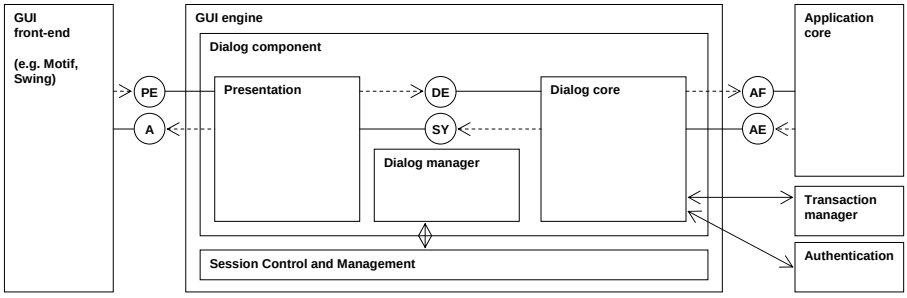

Fig. 1. Reference architecture of a user interface [4]

2.3 Adaptive Dialog Core

In a SOA environment we expect to have several application cores in different technologies (e.g. distributed Enterprise JavaBeans or web services). We therefore need to design a dialog core, which integrates and adapts to new servers/web services at runtime without having to recompile, repackage and redeploy the UI.
Figure 2 visualizes the usage of such an adaptive dialog core. The adaptive dialog core provides four functionalities F1 to F4 technology-independently through the interface DE (see figure 1) to the Presentation component (see figure 1). Each functionality Fx is offered by a technology-dependent application core, e.g. a JSE server, a JEE server (like JBoss), an AXIS web service or a JEE web service (like JBoss web service). Generic integrator components (technical adapters) are responsible for communicating with the application cores and offer the functionality Fx to the dialog core by hiding the specific technology.

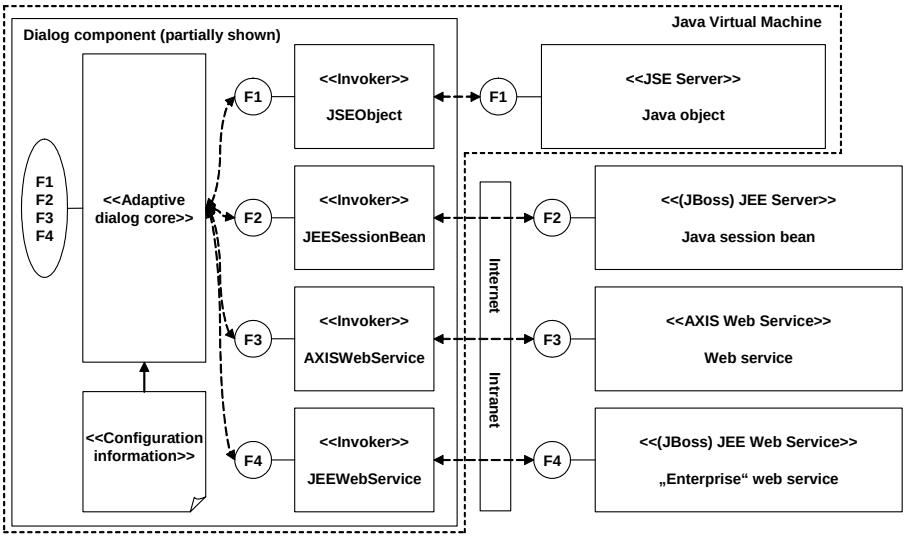

Fig. 2. The adaptive dialog core and its technical adapters as new components of the UI

2.4 Flexiblity

Configuration of connection parameters. Application cores are realized by different technologies and require therefore adapted UI engines. In order to communicate with web services, it is e.g. necessary to generate stubs (as technical adapters) and package them into the UI in order to create, send and receive SOAP messages. By using our approach of an adaptive dialog core, we would offer a generic, configurable web service invoker that allows switching to another application core at runtime without having to recompile and repackage the UI.

Configuration of functional mapping. Moreover, application cores can also provide different functionalities. In case of an eShop, we can imagine that shopping cart functionality (F1), user registration (F2), credit card check (F3) and purchase order (F4) are provided by different application cores. However, all these four functionalities should be made available to users through the UI. The adaptive dialog core can fulfill this demand by configuration.

Configuration of QoS. In case a JEE web service drops out suddenly and hence the data cannot be persisted any more, a pre-configured JEE server could replace it immediately. We imagine that the adaptive dialog core knows about redundant application cores and switches automatically to one of these in case a breakdown occurs. For other quality of service (QoS) demands, like security (encryption, signatures and authentication), we imagine that users may want to choose the appropriate application core in the UI by themselves, in order to fulfill their quality demands.

3 Designing the Adaptive Dialog Core

In this section we present the design class diagram of the adaptive dialog core and give some details of the design decision we took during creating the class diagram.

3.1 The Model

The model of our adaptive dialog core is presented in figure 3 as UML class diagram. The `DynamicAdaptionModel` enables us to configure several dialog cores. It offers several `Operations` to the dialog core, which actually correspond to the functionalities F1 to F4 from figure 2. These operations are named e.g. like “register”, “login”, “observeAuction” and “bid”. An `Operation` has multiple `OperationInputParamter`, one `OperationOutputParamter` and can throw several `OperationExceptions`. Each `Operation` can be called and executed by an `Invoker`, or more precisely a subclass of the abstract class `Invoker` which is a customization for a specific technology. In case there is no `Invoker` that integrates your application core, you have to add such by yourself. A `JSEObjectInvoker` is instantiated in the same Java Virtual Machine where the UI lives and is able to call local operations. A `JEESessionBeanInvoker` is able to call operations of distributed JEE Session Beans, which are hosted by a JEE Application Server. A `WebServiceInvoker` is able to send and receive SOAP message to and from Web Services. The libraries needed for the invocations of local and remote operations can be received via `RemoteJar`. At last, according to which technology we want to integrate, the `Operations` of the appropriate `Invoker` will be mapped to those of the dialog core at runtime. Although the input and output parameters can be defined using complex data types, it is sufficient to apply a `SimpleMapping` when mapping the input and output fields of the dialog and application core operations.

3.3 Design Decisions

Java platform. The UI of the application *Auctions* should be implemented as a web application by using Java Servlets [6], JavaServer Pages [6], JavaServer Faces [6] or web frameworks (Struts, Tiles etc.). The adaptive dialog core, as a component of the UI, is however independent from the presentation technology and can be implemented in pure JSE [5].

Remote libraries. Because our adaptive dialog core targets the Java platform, application cores have to offer Java libraries that allow communication. Libraries must contain Java classes. Because these libraries are necessary they have to be downloaded in the UI through Internet from the application core provider at runtime. Because the UI is implemented in JSE, we solve this problem by using the JSE class `URLClassLoader` and `JarURLConnection` for loading a class dynamically over HTTP. In order to be integrated with e.g. Web services, our adaptive dialog core must be able to access jar files of SOAP libraries like *soap.jar*, *saaj.jar* etc. For integration with a SAP system we need a SAP client adapter for Java called *jco.jar* (Java Connector) in order to perform a remote function call (RFC).

Single operation output parameter. We decided to model operations with only one output parameter because we target the programming language Java. Fortunately this assumption does not limit the capabilities of web services cause several web service message parts can be handled by one complex XML Schema type (like WSDL 2.0 [10] assumes).

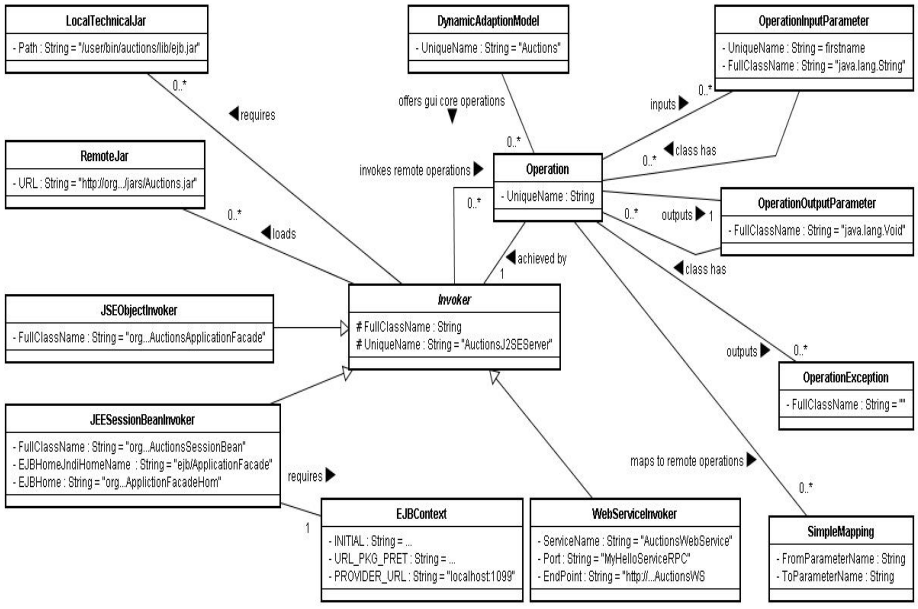

Fig. 3. Design class diagram of the adaptive dialog core

4 Summary and Outlook

In this article we outlined the idea of an adaptive dialog core for service-oriented user interfaces that allows technical and functional integration of multiple application cores at runtime (section 2). We proposed a model for the adaptive dialog core and explained important design decisions (section 3).

Development of *business applications* using service-oriented technologies and frameworks is taking place intensively and is becoming an important business strategy among rival software companies. In future, customers of these software companies will get more choices which products they will rely on, especially when enhancing their existing system landscape with new back-ends, then consisting of mixed installations, e.g. SAP NetWeaver platform plus Microsoft .Net platform plus a JBoss open source platform. Our approach enables software consumers to integrate UIs with back-ends in a fast and flexible way.

But our approach might be useful in the area of *service grids* as well. A grid service provider is someone who bundles several grid providers. The grid providers offer special features of their grid, e.g. high computing performance or high I/O performance. Thus, when the grid service provider receives a request from a grid service user, it will consider which grid provider would be most suitable to fulfill this request and then delegate the request to that grid provider. Therefore, we think of enhancing the adaptive dialog core with a quality of service component in future.

References

1. J. Rückert, J. Horvat, D.-K. Nguyen, S. Becker, B. Paech: **Modell for Adapting Dialog Cores.** Modellierung 2006, Workshop Quality of Models. Innsbruck, Austria, 2006.
2. B. Paech: Lecture notes "**Software Engineering of modern applications – component based, service oriented or mobile systems**", Institute of Computer Science, University of Heidelberg, Germany, 2006 (in German).
3. J. Rückert, B. Paech: **Software Engineering of modern applications**. SEUH 2007 (Education of Software Engineering in Universities). Stuttgart, Germany, 2007 (in German).
4. J. Siedersleben: **Moderne Software-Architektur**, Dpunkt Verlag, 2004 (book in German).
5. Java Platform, Standard Edition (JSE), Version 5. http://java.sun.com/javase/. Sun Microsystems.
6. Java Platform, Enterprise Edition (JEE), Version 5. http://java.sun.com/javaee/. Sun Microsystems.
7. JBoss open source JEE Application Server (JBoss AS), Version 4.0.5. http://www.jboss.org/. JBoss.org.
8. Apache AXIS 2, Version 1.1. SOAP implementation. http://ws.apache.org/axis2/. Apache.
9. SOAP 1.2. http://www.w3.org/TR/SOAP/. W3C Recommendation, 24 June 2003. World Wide Web Consortium.
10. Web Services Description Language (WSDL) 2.0. http://www.w3.org/TR/wsdl20/. W3C Candidate Recommendation, 27 March 2006.

Different approaches exist to describe user interfaces for interactive applications and services in a model-based way using user interface description languages (UIDLs). These descriptions can be device and platform independent and allow adaptivity to the context of use, although this adaptivity has to be predefined. In this paper we motivate the use of UIDLs in a broader view: As the basis for building adaptive, flexible and reliable systems using adaptive dialog management. The goal is to provide the user with the right service and the right interface at the right time.Â @inproceedings{Honold2011AdaptiveDM, title={Adaptive Dialogue Management and UIDL-based Interactive Applications}, author={F. Honold and M. Poguntke and Felix Sch{\"u}ssel and M. Weber}, year={2011} }. Add a custom trigger to the adaptive dialog to react to the event. Sign out a user. SignOutUser. Signs out the currently signed in user.Â Just like when invoking any dialog in the Bot Framework SDK, when calling BeginDialog to invoke an adaptive dialog, you can use the options parameter to pass input information for the dialog. EndDialog. Ends the active dialog by popping it off the stack and returns an optional result to the dialog's parent. 2 User Interface Architecture. Model-based UI software development has introduced concepts and techniques that assist in the process of UI development [3, 4]. These concepts give developers a better understanding of the eld of UI design. There are three phases in the process of UI design [5]: The semantic level design describes the tasks users should be able to perform using the application for which the UI provides the interaction means.Â Grouping UI elements that implement the dialog model into a hierarchical structure of windows is the essential step our adaptation process that leads to single authoring.Â Menkhaus, G.: An Architecture for Supporting Multi-Device, Client-Adaptive Ser-vices. User interfaces (UI) for the computing systems have changed in the last 20 years. The first UIs based on text that used a command prompt to access the operating system resources, have been replaced for user graphics interfaces (GUIs) that are manipulated through entry devices like keyboard and mouse. At present, user interfaces attempt to be more intuitive to the user by presenting graphic elements visually associated with real elements by the use of metaphors (Dix et al, 2003.)