Rapid Development of Composite Applications Using Annotated Web Services

Lars Dannecker[1], Marius Feldmann[2], Tobias Nestler[1],
Gerald Hübsch[2], Uwe Jugel[1], and Klemens Muthmann[2]

[1] SAP Research Center Dresden
Chemnitzer Str. 48, 01187 Dresden, Germany
{lars.dannecker,tobias.nestler,uwe.jugel}@sap.com
[2] Technische Universität Dresden, Department of Computer Science,
Institute for Systems Architecture, Computer Networks Group
{marius.feldmann,gerald.huebsch,klemens.muthmann}@tu-dresden.de

Abstract. Developing service-based interactive applications is time consuming and nontrivial. Annotating web services with additional information about the user interface and behavior of the service promises to ease and accelerate the development process. In this paper we present a model-driven development approach that utilizes the advantages of annotated web services during the service composition. It enables a rapid development of simple service-based applications in a graphical manner. Besides the description of the process and the models, our current service composition tool implementation, the "ServFace Builder" is introduced. The ServFace Builder infers graphical UIs from functional interface descriptions such as WSDL files which then can be combined in a presentation-oriented way to a service-based interactive application.

Keywords: Service Composition, Model Creation, Service Frontends.

1 Introduction

Nowadays web services are an important means for developing distributed applications. However, developing user interfaces for web services is still carried out manually by software developers for every new service. This process can be expensive and error prone. UI elements have to be created, parameters have to be bound to input and output fields, code for invoking web service operations has to be written for one or several of the application's pages, and also many other UI events, such as list selection or scrolling, have to be handled. Currently, many of these steps cannot be performed automatically due to missing tool support.

This paper describes the internals of the ServFace Builder - a Rich Internet Application (RIA) development tool that integrates a model transformation from an annotated service description to an executable web application. This tool is capable to automate most steps in the development process of service-based interactive applications. In this paper, we show how the ServFace Builder reduces the effort to execute reoccurring development steps and how the targeted user group of domain experts and other non-programmers is supported by the tool.

F. Daniel and F.M. Facca (Eds.): ICWE 2010 Workshops, LNCS 6385, pp. 1–12, 2010.

The main concept of the ServFace Builder is the inference of user interfaces from functional service interface description files such as WSDL documents. However, the information within service descriptions do not contain sufficient information for generating useable UIs. A simple example for missing UI information is a human-readable, localized label of an input field.

To represent such information, the ServFace project applies the concept of service annotations. By using a specific authoring tool, ServFace Annotations can be attached to any node (e.g. an operation or a parameter) in a functional interface description to provide explicit information for UI integration. The annotations themselves are sufficient to generate usable UI parts of an application, but to integrate the parts into a bigger context an application meta-model for storing the overall application structure, additional logic for the UI and relations between different pages are necessary. Therefore we have developed the Composite Application Model (CAM). The ServFace Builder allows to transform annotated service descriptions via a model-to-model transformation to a part of a CAM instance. The CAM is the storage format and the input model for model-to-code transformations.

In this paper, we discuss an approach for presentation-oriented design of rich internet applications. In Section 2, we explain the benefit of annotated service descriptions and in Section 3, we describe a generic meta-model for describing interactive service-based applications. The ServFace Builder uses those ideas to allow end-user friendly web application development. The tool's annotation-driven, automatic UI generation creates a visual representation for service operations, which can then be combined in a presentation-oriented way (see Section 4). Additionally, the ServFace Builder supports development for several target platforms by using our generic meta-model and a sophisticated model-to-code transformation, described in Section 5. We finally evaluate all concepts and the tool in Section 6, using a university portal web application as a complex scenario.

2 Service Annotations

Our service composition approach is based on services that are described via the Web Service Description Language (WSDL) and enriched with UI-related *service annotations*. These annotations are reusable information fragments, which are typically not available for an application developer or service composer. They provide extensive additional information covering aspects of the visual appearance of a service (e. g., add a *label* or *group* parameters, *enumerations* for predefined values) the behavior of UI elements (e. g., client-side *validation* rules, *suggestion* of input values) and relations between services (e. g., *semantic data type relation*). The annotations are created by the service developer to provide additional information about the web service. Thus, they can be seen as a kind of knowledge transfer from the service developer to the service composer to facilitate the understanding, to enhance the automated UI generation and to simplify the composition of web services. Once defined the additional information provided by the annotations are available for all application developers that intent to use the annotated service.

3 Meta-model Overview

For representing pieces of information, required to build interactive service-based applications, the following three domain-specific meta-models have been developed. An overview of these models is given in the next paragraphs.

Annotation Model. The Service Annotations are stored in an annotation model based on a formally defined meta-model that references the elements within the WSDL. An in-depth discussion of the annotation model can be found in [1].

Service Model. To infer a UI for a service operation as a basis for the presentation-oriented authoring process (compare sec. 4), a tool internal representation of an annotated web service is necessary. For this purpose a specific meta-model is used, which is instantiated by a WSDL parser. Besides the information contained within the functional interface description, the model instances hold references to the service annotations represented as instances of the ServFace Annotation Model. The service model and its structure is described in detail in [2].

Composite Application Model. The service model is an integral part of a domain-specific meta-model named *Composite Application Model* (CAM). It describes the integrated services, their connections in form of data flows as well as the navigation flow of the entire application. The CAM is used as the internal model of the ServFace Builder and furthermore serves as the input for the generation of executable applications. The meta-model is not bound to a specific platform and thus can be reused for various target platforms. Concerning the CAM, a service-based interactive application is formed by a set of interconnected pages. Each page is a container for service front-ends as described in section 4.2. The CAM instance is continuously synchronized with authoring actions performed by the user (e.g., add page, integrate front-end, define data flow). The CAM is described in detail in [2].

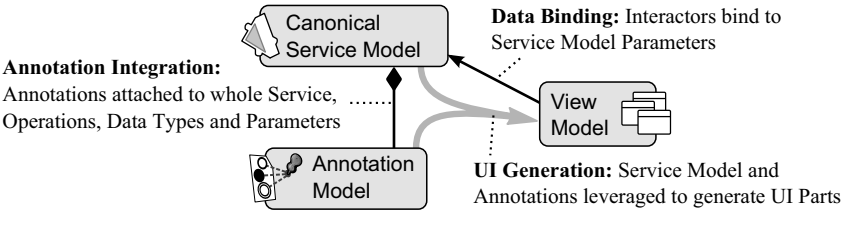

Fig. 1. Relationships between the models

4 ServFace Builder

The ServFace Builder (Fig. 2) is a web-based authoring tool developed in the course of the EU-funded project ServFace[1]. The graphical design-time tool utilizes

[1] `http://www.servface.eu`

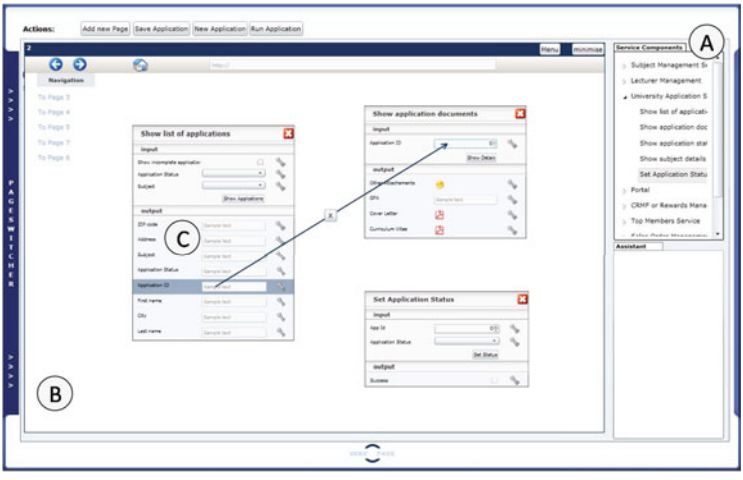

Fig. 2. The ServFace Builder

the described meta-models in order to realize a model-driven development of multi-page interactive applications for various target platforms and devices.

The tool applies the approach of service composition at the presentation layer [3] in which applications are built by composing web services based on their frontends, rather than application logic or data. It builds upon the concept of UI integration introduced by [4]. Our UI-centric composition approach aims to support non-programmers that are familiar with the semantics of their specific application domain. They understand the meaning of the provided service functionality without having a deep understanding of technical concepts like web services or service composition.

The goal of the graphical composition concept is to hide the complexity of the actual programming task by representing a service entirely by its corresponding UI and let the user only work with these visual representations in order to model all relevant aspects of the application development process. Thus, our approach fully abstracts from technical details in a presentation-oriented way. This enables people without substantial technical background to use our tool and create interactive applications based on web services.

The ServFace Builder consists of several components. The Service Browser (fig 2.A) shows all services available for the application creation process. The user creates the application on the Composition Area (fig 2.B) which represents a page of the final application. Each page contains the UIs of the added service operations (fig 2.C).

4.1 Composition Process

To create a multi-page interactive service-application with the ServFace Builder the user has to perform the following steps:

Select target platform: As the initial step, the user has to select the target platform from a set of predefined platform templates (compare sec. 4.4).

Integrate service operation: After the platform selection, a blank initial page is created and the actual authoring process begins. The user can integrate service operations from the Service Browser by dragging them to the Composition Area. The tool automatically creates the corresponding service frontend.

Define data flow: In order to define a data flow between two service frontends, the user has to connect UI elements of both frontends to indicate the relationship.

Define navigation flow: Frontends can be placed on one page or distributed over several pages to create a multi-page application. Pages can be connected in order to create a navigation flow.

Deploy application: After finishing the design process, the modeled application can be deployed according to the selected platform.

The following sections describe the particularities of the ServFace Builder in detail.

4.2 Automated UI Generation

During the design process each service operation is visualized by a generated UI that is called *service frontend*. The frontends consist of a nested container structure, which includes UI-elements like text fields or combo boxes that are bound to the corresponding operation parameters. Frontends are automatically generated by the *Visualization Engine* of the ServFace Builder when the user adds an operation to the application. The basis of the generation process is the associated service model that is retrieved from a remote repository containing all available annotated services. The engine parses the service model and analyses the elements of a particular service operation. For each service element that needs to be displayed in the service frontend, a proper UI element is inferred based on the class of the parameter (e.g. input or output), the basic data type, the data type configuration (e.g. enhancements, restrictions) and the cardinality. Complex data types are displayed as lists or tables in most cases. However, for complex types with a deeply nested or recursive structure the ServFace Builder provides a special widget. To enhance the visual appearance and the functionality of the service frontends, the ServFace Builder utilizes the information of the ServFace service annotations. Furthermore, we consolidated several HCI guidelines e.g. provided by companies such as Apple [5] or usability experts like [6] to form a set of UI design recommendations. They are used within the frontend generation to ensure a usable UI. Whenever the Visualization Engine finds annotations attached to an element of the service model, it uses an effect description for each annotation. This effect description specifies the effects the particular annotation has on the representation of a service operation. The effects are then incorporated into the UI element generation. During the generation process the current CAM instance is updated accordingly. Afterwards, the frontend is displayed to the user. The generation process that is depicted in figure 3 runs completely automatic in

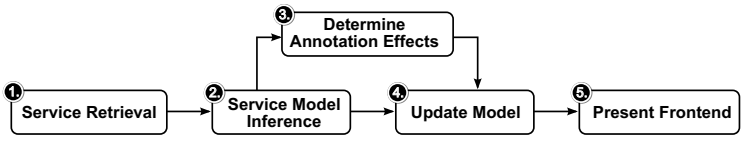

Fig. 3. The frontend generation process of the ServFace Builder

the background and without the necessity of further interaction, as the user just needs to add the service operation to the service composition. This simplifies and accelerates the design process of the application, since there is no need for a manual UI creation.

4.3 Presentation-Oriented Modification and Composition

After the frontend generation process has been realized, the user can modify several aspects of the frontends such as renaming or hiding elements. In addition, the frontends can be composed with other web service operations in a graphical manner to create the interactive application. This means that the user directly interacts with the input and output elements of the frontends to directly connect several service operations. A connection is created by a sequence of two clicks that identify the target UI element that is to be filled with data and the source UI element from which the data is to be taken. The data flows are visualized in form of arrows between the connected UI elements. The direct use of the UI elements benefits from the fact that form-based applications are well-known and most users are familiar with them. Furthermore, the user can create several pages to place the frontends on. All created pages are shown in a graphically structured overview and can be connected to model page transitions. The definition of the navigation flow is done in a graphical way as well. The user has to connect two pages in order to create a page transition. These presentation-oriented concepts distinguish our approach from other composition editors like Yahoo! Pipes [2] or the IBM Mashup Center [3] that often use abstract concepts like ports and wiring as a depiction for a connection between service operations. In addition they focus on the processing of data, rather than the creation of service-based interactive applications.

The ServFace Builder interprets the changes done by the user automatically in the background and adjusts the affected parts of the CAM accordingly. Thus, the user creates an application in WYSIWYG (What you see is what you get) style without a need to write any code or manually change the CAM. This lowers the barrier to use the editor, even for inexperienced users.

4.4 Support for Different Target Platforms

Our approach allows users to create applications for different platforms. The ServFace Builder supports this by offering a selection of several target platform

[2] `http://pipes.yahoo.com`
[3] `http://www.open-mashups.org/`

templates that adjust the CAM instance and the visualization of the UI to reflect the visual appearance of the application on the specific platform. This gives the user the possibility to directly adjust applications to the specifics of the target platform, e.g. limited space due to a smaller screen size. The templates conform to a pre-defined XML schema. This ensures the possibility to easily create new templates and therefore enhance the platform support of the ServFace Builder. An alternative to this approach would be to create an application in an abstract way, not knowing how it will look like on the target platform. Such applications can be transformed into executable files for any of the supported plattforms. Because the look and feel of an application strongly differs for different platforms, the creation of an abstract application can lead to unexpected or even unusable results. In addition we adopt the WYSIWIG principal which makes a platform specific representation a requirement.

While the CAM as a meta-model is platform independent, its instances contain specific aspects and configurations of the chosen platform like concrete UI elements. This is necessary to ensure a proper model-to-code transformation. It is planned to enhance the ServFace Builder in the future to allow a switch between several platform templates at design time which results in the creation of multiple CAM instances and applications at the same time.

5 Transforming the Application Model into an Executable Application

The final results of the ServFace Builder are deployable service based interactive application packages, each possibly for several different platforms. In our research we concentrated on two kinds: Packages for the mobil platform in the form of Google Android and packages for the Web case using Java Enterprise Edition. The following paragraphs describe at first the general steps to create a running application from a CAM instance and then delve into some details of the Android platform.

5.1 Model-To-Code Transformation

Two contrary approaches are possible to create a running application from a CAM instance. It is possible to either have a platform specific interpreter that reads the CAM instance during runtime or to generate a complete application with a model-to-code transformation. Because a generated application is faster than an interpreted one, the ServFace Builder uses the generation approach.

The model-to-code transformation as shown in figure 4 takes instances of the three metamodels - CAM, Service Model and the Annotation Model - as input and transforms them to application packages. Two important findings were made during the implementation of the model-to-code transformation. The first was that there is very much static code, which stays the same for every generated application. Generating this over and over again produces much overhead during the generation phase. The static code results from the static structure of the three

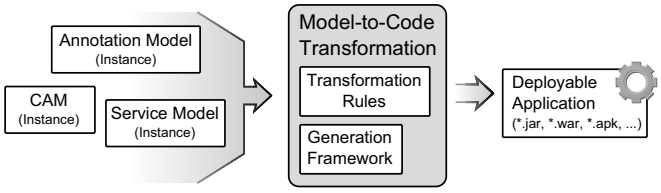

Fig. 4. Model-to-code transformation process

metamodels and thus represents their structure inside the code. To accelerate code generation and to simplify the generation rules, each metamodel was thus implemented as a static framework, which only needs to be instantiated upon generation time. The second finding is that platform specific information needs to be added at generation time as well. So the code generation rules and the implementation of the metamodel framework is platform dependend.

5.2 Android Platform

This section presents some issues with the runtime generation for the android platform as one example of supported platforms. Android applications are structured by `Activity` objects, which are screens a user can interact with. The ServFace Android generation framework and thus all generated applications are structured around a central dispatcher `Activity`. This `Activity` controls the control and dataflow by calling the correct `Activity` objects at the right time. Upon completion of its tasks every `Activity` returns control flow to the dispatcher, which then calls the next one. CAM Pages, since they can contain an arbitrary number of interactors, are split into several Android activities. These activities represent the page content tree from the CAM and the user can navigate along its edges to fill out input fields, invoke services and view output interactors. Annotations are resolved during the transformation.

The most complex part is the mapping of datatypes to interactors. Since webservices may accept and produce arbitrary values, the framework reads and displays these values on a best effort basis. This means for example if a list widget is selected to display a return value, the framework tries to convert the value to a list. If this is not successful it falls back to just displaying a string representation of the return value as the first entry of the list. The framework contains conversion rules for the most common object types for each widget and fallback rules for generic types (like `Object`).

6 An Example Scenario

The following sample scenario shows how the ServFace Builder can be integrated into web applications that are built upon annotated services. Furthermore, it demonstrates the potential and practical applicability of the concepts behind annotated services.

Our scenario is built around a web-based student portal through which a university gives students the possibility to organize their studies. Enrolled students receive a log-in for the portal by which they can manage their course and lecture subscriptions and view their timetable online (fig. 5a). The functionality of the portal is completely implemented by operations of a single Portal Web Service. As the number of students owning smartphones rapidly increases, the university decides to make subsets of the portal functionality available for these mobile platforms. Instead of implementing a mobile version at high cost that may not meet the requirements of all students, the university annotates the Portal Web Service and adds a link to the ServFace Builder through a logo on the portal's main page (lower left corner in fig. 5a), allowing students to build their own mobile applications based on the functionality of the Portal Web Service.

In our case, a student of psychology wants mobile access to her time table and details, e.g. to find out the room number were the next lecture is taking place, of the courses she is enrolled in. In the portal, she clicks on the logo and the ServFace builder opens (fig. 5b). She selects Google Android as the target platform for the new application. The ServFace builder shows the list of operations offered by the Portal Web Service. The student scans the list and identifies two operations: getTimetable, which returns the student timetable, including the id

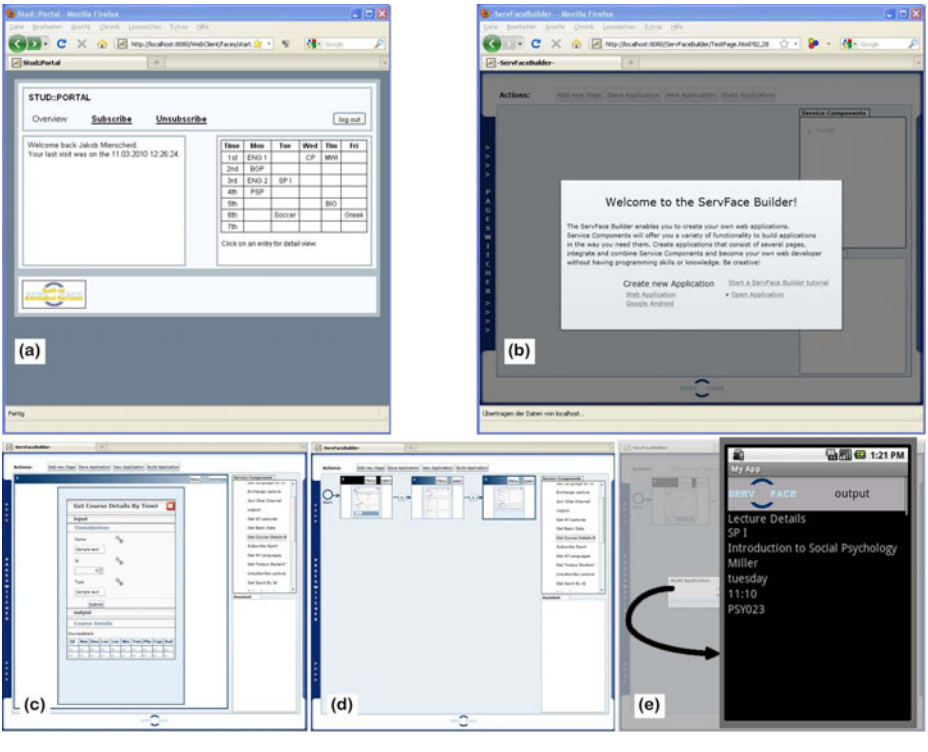

Fig. 5. University Portal Scenario

of each lecture given the student id, and getDetails, which returns lecture details given the lecture id, as the ones that meet her requirements. She now visually composes these operations using the ServFace builder by first dragging each operation on a separate page. Based on the annotations, a suitable UI is generated for each operation (fig. 5c). Also, the student is informed that she needs to add the login operation before she can use the others, creating a third page. In a next step, she defines the page flow (fig. 5d) and data flows based on input and output fields of the generated UIs. She links the input field for the getDetails operation to the output field of getTimetable that contains the course id. Another data flow is defined to fill the input field of the getTimetable operation that is already available from the login operation. She now clicks the 'Build Application' button of the ServFace builder which sends the CAM instance to the model-to-code transformation running on a server. The generation result of the transformation is a deployable Google Android application that integrates the Portal Webservice. Finally, a download link to the deployable Google Android application is presented (fig. 5e). After downloading and installing the application, the student can use a part of the portal functionality on her smartphone.

7 Related Work

Service Annotations. Our concept of service annotations tries to leverage existing work as much as possible and is partially based on former approaches, i.e., the GUIDD annotations [7] used by Dynvoker [8] and by the WSGUI project [9]. As a runtime engine applying the concept of annotated services for ad-hoc service usage, Dynvoker uses the GUIDD annotations to generate user interfaces for web services at runtime. Although these existing approaches already cover the generation of user interfaces for single web services, no solution is available for service composition and the usage of annotations for an application's design time.

Mashups. The so called Web 2.0 offers users the capability to build sophisticated web applications like mashups by combining distributed resources and contents into new *composite* web applications. Graphical *mashup platforms* (e.g. Open Mashup[4] and Yahoo Pipes[5]) aiming for a (partially) assisted web development for less skilled programmers or even non-programmers. However, mashups mainly focus on the data aggregation and still lack of concepts to create composite service-based applications by end-users ([10], [11]).

UI Integration. Research projects like CRUISe [12] or mashArt [11] build upon the fundamentals of UI integration and follow a comparable philosophy centered around the idea of event-based UI components. The three main differentiations of our work to both projects can be seen in the clear restriction to the presentation layer (1), the direct interaction with UI elements during the design time in

[4] `http://www.open-mashups.org/`
[5] `http://pipes.yahoo.com`

order to define all composition tasks on the UI level (2) and the focus on non-programmers and skilled web users as our main target group (3).

MDD of service-based applications. There already exist model-driven development tools that can be used for the creation of service-based applications such as UsiXML [13], EMODE [14] and Teresa/Maria [15]. These approaches apply a model transformation chain starting with abstract models and refining them to instances of concrete models, which are then transformed to executables or interpreted at run time. In contrast our approach of using annotations presents a loose coupling of model-driven development techniques and service annotations instead of a strong integration of abstract UI-models. In addition, our models are designed to be generic and reusable for applications on different platforms, which distinguishes our approach from solutions such as AndroMate [16] (Android only) and WebRatio [17] (Web Applications only).

8 Conclusion and Future Work

The paper presents a model-driven development approach for service-based interactive applications. Part of this approach is the ServFace Builder, a tool for the presentation-oriented composition of web services. The ServFace Builder automatically infers UIs for service operations. The user can combine these UIs via navigation and data flows in a visual manner to create a service-based application which is stored as an instance of a generic application meta-model called CAM. This instance serves as input for model-to-code transformations targeting different platforms. In addition to the already conducted end user studies (e.g. compare [18]), future work will concentrate, at first, on additional evaluations of the ServFace Builder's usability on a large group of real end users, and in different scenarios. Furthermore, it will be analysed how generated service frontends can be embedded into the context of already existing applications.

Acknowledgment

This work is supported by the EU Research Project (FP7) ServFace.

References

1. Janeiro, J., Preussner, A., Springer, T., Schill, A., Wauer, M.: Improving the Development of Service Based Applications Through Service Annotations. In: Proceedings of the WWW/Internet Conference (2009)
2. Feldmann, M., Nestler, T., Jugel, U., Muthmann, K., Hübsch, G., Schill, A.: Overview of an End User enabled Model-driven Development Approach for Interactive Applications based on Annotated Services. In: Proceedings of the 4th Workshop on Emerging Web Services Technology. ACM, New York (2009)
3. Nestler, T., Feldmann, M., Preussner, A., Schill, A.: Service Composition at the Presentation Layer Using Web Service Annotations. In: Proceedings of the Composable Web 2009 Workshop, ICWE (2009)

4. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. IEEE Internet Computing 11(3), 59–66 (2007)
5. Apple Developer Connection: Apple Human Interface Guidelines. Technical report, Apple Inc. (2009)
6. Krug, S.: Don't make me think! A Common Sense Approach to Web Usability. New Riders (2006)
7. Kassoff, M., Spillner, J.: GUI Deployment Descriptor (GUIDD) Standard Specification and Documentation (2006), http://inf.josefspillner.de/webservices/guidd.html
8. Spillner, J., Feldmann, M., Braun, I., Springer, T., Schill, A.: Ad-hoc Usage of Web Services with Dynvoker. In: Mähönen, P., Pohl, K., Priol, T. (eds.) ServiceWave 2008. LNCS, vol. 5377, pp. 208–219. Springer, Heidelberg (2008)
9. Kassoff, M., Kato, D., Mohsin, W.: Creating GUIs for Web Services. IEEE Internet Computing 7(5), 66–73 (2003)
10. Ro, A., Xia, L.S.Y., Paik, H.Y., Chon, C.H.: Bill Organiser Portal: A Case Study on End-User Composition. In: Hartmann, S., Zhou, X., Kirchberg, M. (eds.) WISE 2008. LNCS, vol. 5176, pp. 152–161. Springer, Heidelberg (2008)
11. Daniel, F., Casati, F., Benatallah, B., Shan, M.C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Proceedings of the 28th International Conference on Conceptual Modeling, pp. 428–443 (2009)
12. Pietschmann, S., Voigt, M., Rümpel, A., Meißner, K.: CRUISe: Composition of Rich User Interface Services. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 473–476. Springer, Heidelberg (2009)
13. Vanderdonckt, J.: Model-driven engineering of user-interfaces: Promisses, successes, failures, challenges. In: Proceedings of ROCHI 2008 (2008)
14. Behring, A., Heinrich, M., Winkler, M., Dargie, W.: EMODE Model-Driven Development of Multimodal, Context Sensitive Applications. i-com 6(3) (2007)
15. Paterno, F., Santoro, C., Spano, L.: Designing Usable Applications Based on Web Services. In: Proceedings of the Int. Workshop on Interplay between Usability Evolution and Software Development (2008)
16. Ebben, P., Heerink, L., Reitsma, J., Steen, M.: Andromate project (2008), http://www.lab.telin.nl/~msteen/andromate/
17. Brambilla, M., Comai, S., Fraternali, P., Matera, M.: Designing Web Applications with WebML and WebRatio, Web Engineering: Modelling and Implementing Web Applications. Technical report. Springer (2007)
18. Namoune, A., Nestler, T., Angeli, A.D.: End User Development of Service-based Applications. In: Proceedings of the Second Workshop on HCI and Services, at HCI (2009)