An Integrated Approach for Creating Service-Based Interactive Applications

Marius Feldmann[1], Jordan Janeiro[1], Tobias Nestler[2], Gerald Hübsch[1], Uwe Jugel[2], André Preussner[2], and Alexander Schill[1]

[1] Technische Universität Dresden, Department of Computer Science, Institute for Systems Architecture, Computer Networks Group
[2]SAP Research CEC Dresden, 01187 Dresden, Germany
{marius.feldmann, gerald.huebsch, jordan.janeiro, alexander.schill}@tu-dresden.de
{uwe.jugel, tobias.nestler, andre.preussner }@sap.com

Abstract. While the implementation of business logic and business processes based on service-oriented architectures is well-understood and covered by existing development approaches, integrated concepts that empower users to exploit the Internet of Services to create complex interactive applications are missing. In this paper, we present an integrated approach that fills this gap. Our approach builds upon service annotations that add meta-information related to user interface generation, service dependencies, and service composition to existing service descriptions. Services can be composed visually to complex interactive applications based on these annotations without the need to write any code. The application code is generated completely from the service composition description. Our approach is able to support heterogeneous target environments ranging from client/server architectures to mobile platforms.

Keywords: services, composition, annotations, interactive applications, MDA

1 Introduction

Service-oriented architecture is a design concept that implements system functions as a set of highly reusable and distributed services. These services are composed to implement business applications and processes. Existing work in this area mainly focuses on these aspects and does not sufficiently address the composition of services to create service-based interactive applications. In these applications, the user directly interacts with services via a rich user interface (UI). In this context we assume that reuse in service-orientation is not limited to the implementation of system functions. Reuse can be extended towards the aspects user interaction and service composition through *services annotations* that attach meta-information to services, thereby creating *annotated services*. The UI of a service-based interactive application can be composed out of reusable *UI fragments*, each of which is attached to a data type, parameter, operation or the service itself as an annotation. For example, a service operation that provides user authentication would be associated with one UI fragment for entering login information, and two other UI fragments for displaying the authentication result (‘successful’, ‘not successful’). With regard to service

composition, service annotations specify dependencies between services. For example, a service for editing user profiles will require an authenticated user. It depends on the presence of an authentication service in the composition. This dependency can be expressed in its annotation, including a reference to the required authentication service. The main benefit of service annotations is that they limit the effort for the development of service-based interactive application to a purely model-driven, visual composition of annotated services. In section 2, we discuss our approach in more detail. We summarize and conclude our findings in section 3.

2 Development Approach

In this section, we present our integrated development approach for service-based interactive applications. It builds upon the basic concept of service annotations which are explained in more detail below. The annotated services build the foundation for the composition of interactive applications. For this purpose, we define a *composite application model* (CAM). Annotated services can be composed in a visual way and transformed to CAM artifacts. The service composition includes arranging UI fragments into pages, defining the page flow, and the data flow between annotated services. A CAM describes the aspects of an interactive application and can be transformed into executable code, which implements the interactive application through *model-to-code transformations* in a process called runtime generation. Due to the technology independence of the underlying annotations and the CAM, multiple target platforms can be supported. We also define a methodology that ties all of these parts together in an integrated development process.

2.1 Methodology

Our development methodology defines three steps: *service annotation*, *service composition* and a fully automatic *runtime generation*. These steps cover the evolution from a set of services to the complete service-based interactive application and are supported by corresponding tools.

The service annotation step supports technical service descriptions, like WSDL[1]- or WADL[1]-files and produces service annotations. The developer, who annotates a service, performs the role of *service annotator* and uses the *service annotation tool*. Such role requires experience in UI development and technical background knowledge about the services which are to be annotated.

The service composition step builds upon annotated services and produces a CAM. The role of *service composer* defined for this step does not require any programming skills and can be performed by an end-user [1] with a basic understanding of the composition concepts. The service composer identifies and composes the annotated services that are required to implement the interactive application. To produce sophisticated applications, the service composer can add all necessary UI layout definitions, page-, and data-flows using the *composition tool*. During the composition

[1] http://www.w3.org/TR/wsdl; http://wadl.dev.java.net

process a CAM instance is continuously updated, to generate executable code for the platform chosen by the service composer.

2.2 Service Annotations

Service annotations are reusable information fragments which are associated *i)* with service definitions, *ii*) with service operations, *iii*) with input parameters, *iv*) with return values, *v*) with failure messages, or *vi*) with structured data types to facilitate the creation of service-based interactive applications.

We categorized the annotations in three groups: *visual*, *behavioral*, and *relational*. Visual annotations concern static UI aspects, behavioral annotations concern the aspect of the behavior of UI elements, and relational annotations concern dependencies between services. Examples of visual annotations are: labels, tooltips, MIME-type information, design templates, and grouping or ordering of UI elements. Behavioral annotations are, for example: rules for client-side input validation, the specification of a data source for an automatic form completion and the specification of a time interval in which a service performs the call of a certain operation (e.g. information retrieval for news tickers). Relational annotations describe service dependencies that express relations between services from a functional and a data perspective, e.g. that a service requires authentication information and a reference to the service that can provide this information.

More than 20 types of annotations have been identified and formally defined in a meta-model. An instance of this model holds all information provided during the annotation task and a reference to the associated service description. The technology-independence of the information within the model assures a maximum level of compatibility with existing and future target platforms.

2.3 Service Composition

The purpose of service composition is the development of interactive applications based on annotated services. The usage of the service annotations enables a simplified design and creation of an application in a visual manner following the concept of *integration at the presentation layer* [2]. The goal of this concept is to hide the complexity of the actual programming task by representing a service entirely by its corresponding UI. The service composer only works with visual representations to build the application.

Service composition integrates annotated services through the definition of pages, and the specification of page- and data-flows. Each page represents a screen in the final application. Pages consist of UI elements that are arranged according to a layout. They enable the interaction between the user and services. The service annotations are used to automatically infer UI elements, page layout, and additional behavior (e.g., suggestion capability). The service composer may modify the generated pages manually by, for example, relocating UI elements within a page or between pages. To build multipage applications, pages can be linked by specifying a page flow. Data flows specify the flow of information between services. For example, the return value of a service operation may be an input parameter of another service operation. These data flows can be partially derived from service dependencies defined in the

annotations. The CAM that results from the service composition contains all necessary information for generating an executable application for the platform that has been selected by the service composer.

2.4 Runtime Generation

After finishing the design-process, the resulting instance can be transformed into an executable application via a model-to-code transformation process. Due to their high relevance the focus has been directed to fat and rich clients (e.g. AJAX-based applications). For every platform supported by the composition tool such a code generation mechanism is provided. For the prototypical implementation two platforms are supported. Web applications are supported by mapping the CAM to AJAX toolkits embedded into the Spring framework. Code generation is also provided for Google Android to validate the approach for a mobile fat client platform. After the final application has been generated, it can be deployed automatically using different deployment strategies for the supported target platforms.

3 Conclusion and Outlook

This paper presented an approach for the visual creation of interactive applications based on a technologically heterogeneous service infrastructure. The central idea of this approach is the introduction of service annotations. These reusable information fragments are attached to services and can be used within the service composition to semi-automatically instantiate a composite application model for describing interactive applications. Instances of the composite application model serve as input for a model-to-code transformation that generates executable applications.

Future work will concentrate on improving the tool environment for service annotation and service composition. The service composition tool will be evaluated continuously in end-user studies to achieve the intended goal of enabling end-users to create applications without programming skills. These applications will be more sophisticated than those that can be created using current mashup platforms [3]. Furthermore, the mentioned meta-models are improved in an iterative manner based on real-world examples. We also consider making the service annotation model available as an open standard.

References

1. Nestler, T. *Towards a mashup-driven end-user programming of SOA-based applications*. In Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, 2008
2. Daniel, F. et al. *Understanding UI Integration: A survey of problems, technologies, and opportunities.* IEEE Internet Computing, 2007
3. Hoyer, V., Fischer, M. *Market Overview of Enterprise Mashup Tools*.
4. Proceedings of the 6th Int. Conf. on Service Oriented Computing, 2008