# IMPROVING THE DEVELOPMENT OF SERVICE-BASED APPLICATIONS THROUGH SERVICE ANNOTATIONS

Jordan Janeiro†, André Preußner‡, Thomas Springer†, Alexander Schill† and Matthias Wauer†

†*Department of Computer Science,*
*Chair of Computer Networks*
*Technische Universität Dresden*
*Dresden, Germany*
*{jordan.janeiro, thomas.springer, alexander.schill, matthias.wauer}@tu-dresden.de*

‡*SAP Research Center Dresden*
*Dresden, Germany*
*andre.preussner@sap.com*

**ABSTRACT**

Following the concept of service-oriented architectures, applications can easily be built by selecting services and composing them in a loosely coupled manner. Because of the high flexibility and reusability of services the approach is well established for the implementation of business processes and software in business-to-business scenarios. However, SOA approaches also have a high potential to improve the development of interactive applications in business-to-consumer scenarios involving human interaction. But despite of this potential, the development of user interfaces for service-based applications is not properly reflected in current service technology standards and development methodologies. In this paper we present an approach for the annotation of services with UI related information which can be exploited for the generation of user interfaces for interactive service-based applications. We discuss the requirements for UI related information annotated to single, reusable services, present our specification of annotations, and explain how these annotations can be exploited during the process of composing annotated services at design time. To evaluate our approach we present an extended case study.

**KEYWORDS**

UI annotations; semi automatic UI generation

## 1. INTRODUCTION

Service-oriented architectures (SOA) have gained a lot of interest in research and industry because of their high agility and flexibility for the design of applications. With the focus on business-to-business scenarios, new applications can easily be built by composing services to complex business processes. The SOA approaches have also a high potential to improve the development of interactive applications like business processes involving human interaction. Despite of this potential, the development of interactive applications, i.e. the development of corresponding user interfaces and its flexible combination together with functional services is not properly reflected in existing service standards and software engineering methodologies. Thus, the development of the appropriate user interface for service-based applications is still a complex task because of its manual development process.

Some approaches propose to generate the user interfaces (UI) automatically by inferring input elements and the overall layout from information about expected parameters and their data types specified in the service interface. However, the user interfaces generated in this way inherently suffer from usability issues. A major problem is that the automatically generated user interfaces are just based on the technical service specification which is not sufficient to generate UIs concerning to usability aspects. Such approaches remain limited to the generation of web forms containing input fields which correspond to the operation parameters and their data type.

Another issue is the automatic generation of important functionalities commonly present in UIs, which enhance the user experience, such as: the validation of values typed in input fields against a certain validation

rule; the automatic completion of form fields; or a suggestion mechanism which presents a list of possible values for an input field based on a text fragment provided by the user.

In this paper we present an approach for the annotation of services with UI related information which can be exploited for the generation of user interfaces for interactive service-based applications. Our goal is to improve the results of semi-automatic UI generation for service compositions and the user experience of such generated UIs. Moreover, the development of interactive service-based applications based on the composition of services should be simplified.

## 2. BASIC CONSIDERATIONS

In this section we explain important terms in the context of this paper and make some basic considerations about the nature and the requirements of service annotations.

### 2.1 Service Descriptions and Annotatable Elements

The functional aspects of services can be described in a number of different service description languages, like WSDL (Chinnici et. al. 2007), covering the interface of the service which consists of a set of data type definitions and a set of operations defined by an operation name, a set of parameters and a return value, these both referring to the data type definitions.


Figure 1. Abstract Web Service Structure (a) and Concrete Example (b)

Figure 1 (a) shows the abstract structure of a service interface description. In this service structure we abstract from technical details that are not interesting for the annotation process, like bindings. Figure 1 (b) shows a concrete example for the web service *AccountManagement*. It has the data type *Customer* with attributes *name*, *email*, and *isEnterprise*, the operation *createCustomer* with the input parameter *customer* of type *Customer*, and the output parameter *return*. The marked elements denote the annotatable service elements, i.e. elements that can be enhanced with annotations.

### 2.2 Example Scenario

To demonstrate the idea behind some of our annotations we introduce the following example scenario from a customer relationship management (CRM) application. A merchant from the sales department of a computer company needs an application for creating quotations for goods. The merchant wants to be able to quickly react on opportunities. Therefore she needs to use a service which provides the latest business news offered by other companies. Having identified an opportunity to sell computer hardware to a potential customer, she wants to create a new quotation. For that, she searches the customer data base to find the potential customer. If the customer does not exist, she creates a new account for the customer, providing some common registration data, such as name, telephone and address. Depending on the type of customer (enterprise or personal), specific data should be filled in. Then she adds quotation items. She searches for the

products she wants to offer and fills them in the quotation, adding the quantity and an optional discount. Then she submits the quotation to the customer.

# 3. SERVICE ANNOTATION MODEL

A service annotation, or just annotation, is a piece of information attached to an annotatable element of a web service and describes a certain aspect of it. The idea behind annotations is that the service developer can transfer the implicit knowledge about a service interface and functionality to the application developer who wants to create a user interface for the service. There are several kinds of information relevant for the generation of UIs which we take into consideration in our annotations: the basic information concerning the appearance of UIs like labels and descriptions of operation elements; the layout like grouping and ordering of elements; the importance of an element as mandatory or optional, depending respectively on whether they need to be displayed or not. The semantics of an operation element can also influence the way it is presented in the UI, e.g. a date can be displayed as a calendar. Another aspect are information needed to add a certain functionality like validation or suggestion.

An annotation model encapsulates the annotations for a complete service, or for so called global data types, i.e. data types that are used by several services. The annotations are stored in separate models to make them independent from the functional service description. This approach is more flexible as it allows to have multiple different annotation models for the same service description.

The annotation models are stored in an annotation repository. When a new service interface description is loaded the repository can be searched for existing annotation models for the service. This must be done for the service itself and for all data types defined or imported in the service interface description, since they might be global data types which have already been annotated.

## 3.1 Service Annotation Meta-Model

The service annotation meta-model defines the different types of annotations, their relationships and how annotations are attached to web service elements. Figure 2 shows an excerpt of the meta-model.

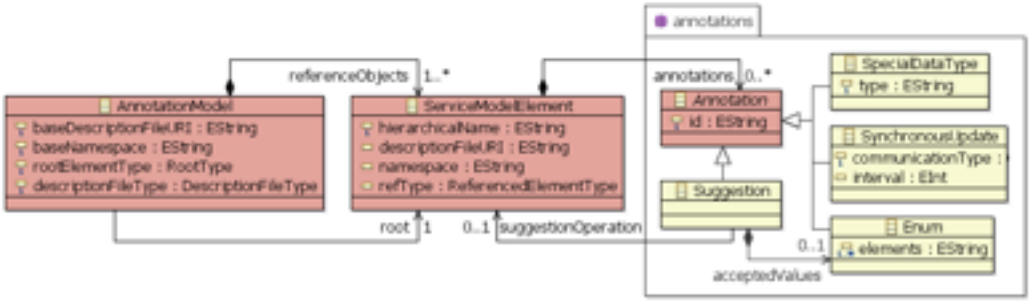


Figure 2. Excerpt from the Service Annotation Meta-Model

An *AnnotationModel* represents either a web service or a global data type. It contains an arbitrary number of reference objects of type *ServiceModelElement*. Each service model element represents a concrete annotatable element in a service interface description. A service model element can have an arbitrary number of *Annotation* instances attached to it. All annotations implement the annotation interface, and can have references to other annotations and to service elements. For demonstration purposes a small selection of annotations (*Suggestion*, *SpecialDataType*, *Enum*, *SynchronousUpdate*) is shown in Figure 2. The suggestion annotation shows that annotations can use other annotations (*acceptedValues*), and reference service model elements (*suggestionOperation*).

## 3.2 Annotations

The annotation meta-model contains 22 annotations in the three categories *visual*, *behavioral*, 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*. Behavioral annotations that provide information to generate a certain behavior are *Group & Order, Validation, Suggestion, Default Value, Example Data, Synchronous Field Update, Mandatory Field,* and *Form Completion*. Relational annotations that describe the relations between annotated elements are *Semantic Data Type Relation, Bundle, Conversion, Authentication,* and *Appearance Change Rule*. Since the space of this paper is not sufficient to present the details of all annotations we present the details of a selection of annotations in the following section.

Using our approach of annotated services has several advantages compared to traditional application development using services. The annotations provide additional layout information, like 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. MIMEType, 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 avoid 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. In addition, annotations can support the understanding of services and their functionality.

## 3.3 Annotation Examples

We chose to clarify four annotations through the running scenario explained in the introduction: Suggestion, Synchronous Field Update, Appearance Change Rule and Special Data Type.

### 3.3.1 Suggestion

This annotation is generally associated with text fields. It recommends the user possible values, based on the text fragment he already started to type.

For example, in our running scenario, the application provides the functionality to create a product quotation for a certain customer. This requires the name of the customer as one of the parameters. In this case, as soon as the user starts to type the name of the customer, a list with possible customer names, beginning with the typed text, is presented to the user, as shown in Figure 3 (a).

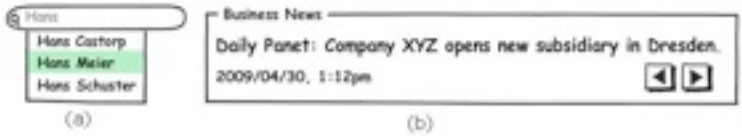


Figure 3. a) Suggestion of Names for a Name Input Field; b) Frequently updated Business News Ticker

To find the suggested values, the typed text fragment might be compared with two possible types of data sources, a static predefined list of values or a web service operation which returns a dynamic list.

### 3.3.2 Synchronous Field Update

This annotation allows a certain UI component to receive frequently information to update its content. A common example is the use of a news feed service, which receives the most recent news from a news provider, as presented in Figure 3 (b).

In our running scenario, we assume that the system displays the recent needs of a company to buy goods in a news ticker UI component. Therefore, the person in charge of buying goods is aware of the needs of the company and can prepare a quotation.

In this annotation, it is possible to specify three content retrieval types: polling, long polling and push. The polling type invokes a certain operation and immediately receives a response with the content. The long polling behaves as the description of the polling but the response may be delayed until new content is available. The push type follows a different approach; it receives the content automatically from the data source instead of invoking a certain operation.

### 3.3.3 Appearance Change Rule

This annotation defines rules to show or hide UI components depending on the value of another UI component.

For example, in the running scenario, to create a quotation for a certain customer, it is necessary to specify whether the customer is a private client or a corporate client. Depending on the kind of customer, different payment options may be displayed. Private customers pay with credit card or bank transfer options, whereas enterprise customers pay through special business-to-business web sites. Therefore, as the user specifies the kind of customer, different UI components are automatically displayed, as shown in Figure 4 (a) and (b).


Figure 4. Different Payment Options for Private Clients (a) and Business Clients (b), triggered by an AppearanceChangeRule

In the context of this annotation we defined a language for the specification of simple conditions using values from several UI components, constants, and boolean operators. When a condition evaluates to true a set of actions (show, hide, enable, disable and clear) can be performed on several UI component.

### 3.3.4 Special Data Type

This annotation indicates that a parameter has one of the predefined special data types *Date*, *Address*, *File*, *Image*, *Color*, or *Font*. These special data types can be represented by special UI widgets. The list of available data types serves for demonstration purposes and may be extended in the future.

For example, in the running scenario, after the creation and approval of the quotation, the purchased goods should be sent to a certain address in a certain date. The data type which describes a date may be automatically represented by a calendar component, by which the user just selects a certain date without concerning about specifying it in the right format as shown in Figure 5 (a). The data type which represents a location may be represented by a map component as shown in Figure 5 (b), where the user selects a location and the address containing the name of the street, number and ZIP code is automatically filled in.


Figure 5. Representation of fields with special data types: Calendar for Dates (a) and Map for Addresses.

For this annotation we identified some UI widgets which are automatically mapped to certain special data types. In our example, we generate a calendar widget if a string data type is annotated with date information and we generate a locator selector widget if a string data type is annotated with location information. Other UI widgets are: file selector, which allows the user to select a specific file from a certain URI location; color selector, which contains already a standard palette of color which a user can select; and the font selector; which also contains the options of standard font formatting.

## 4. EXPLOITING ANNOTATIONS FOR SERVICE COMPOSITION

The presented idea of service annotations is an integral part of a three step methodology for the model-driven development of interactive service-based applications (Nestler et. al. 2009) which is currently developed in the EU-funded ServFace project (ServFace).

The first step is the *annotation* of web services done by the *service annotator*, an IT expert with a good insight in the technical aspects of the services, and basic skills in UI design. The annotation process is supported by an annotation tool. The annotator imports the technical description of a service and obtains a representation of the service structure. The annotator can then annotate the individual service. The resulting annotation model is stored in an annotation repository. The second step is the *visual composition* of services done by the *application developer*. In this step interactive applications are built from annotated services (Nestler et. al. 2009) by exploiting the visual and relational annotations, like Label, Contextual Help and Grouping & Ordering. The result of this visual composition is transformed into an executable application for a certain platform (such as Google Web Toolkit, or Google Android) in the last step, the *runtime generation* (Nestler et. al. 2009). In this step the behavioral annotations, like Validation, Suggestion and Synchronous Field Update are transformed into code.

# 5. IMPLEMENTATION

Our evaluation is based on a case study for which we implemented the formerly introduced CRM application. We implemented the application using the service oriented approach based on web service technologies. After the implementation phases, we applied some of the annotations presented in this paper. Due to space reasons we present just one annotation of our scenario, namely Synchronous Field Update.

The Synchronous Field Update annotation supports the generation of UI components which are constantly updated. In our scenario, the application contains the news ticker component, which displays information about possible customer who needs to buy goods.

We implemented an operation called *getNews*, which retrieves specialized business news about purchasing intentions from customers in the market. This operation accesses a specialized news database to retrieve the latest news about the customer and receives a string containing the source of the information, the date and time of the publication and the information itself in an RSS format, as specified by the web service description in Table 1.

Table 1. Fragment of the Specification for the GetNews Operation

```
<operation name="getNews">
 <input message="tns:getNews"/>
 <output message="tns: getNewsResponse"/>
</operation>
<message name="getNews"/>
<message name="getNewsResponse">
 <part name="return" type="xsd:string"/>
</message>
```

Additionally to the implementation of the operation, it is necessary to specify the annotation properties to build the news ticker component, as shown in Table 2.

Table 2. Instance of the Synchronous Field Update Annotation

```
<servfaceannotationmodel:AnnotationModel baseServiceDescriptionURI=http://
services.news.com/business?wsdl baseNamespace="http://services.news.com">
<referenceObjects xsi:type="servicemodel:WebService" id="NewsService"/>
 <referenceObjects xsi:type="servicemodel:Operation" id="NewsService.getNews">
  <annotations xsi:type="annotations:SynchronousUpdate" interval="60000"/>
 </referenceObjects>
  <referenceObjectsxsi:type="servicemodel:OperationElement"
id="NewsService.getNews.return">
  <annotations xsi:type="annotations:MIMEType" mimeType="application/rss+xml"/>
 </referenceObjects>
</servfaceannotationmodel:AnnotationModel>
```

First, the annotation specifies the location of the service description for the web service which retrieves the news (Line 1-2). Then, the annotation specifies which operation exactly is invoked to retrieve the content, in this case the *getNews* operation (Line 3). To this operation we assign the content retrieval type as polling and we assign the time interval of 60 seconds to poll it (Line 4). We also annotate the return type of the operation, to specify the format of the retrieved data. In our case we defined that the service element return (Line 6) has the MIME type rss+xml which means that the received string actually contains data in the format of an RSS feed (Line 7).

Using this annotation a code generator can generate a UI widget that displays the news content of the RSS feed and create the code that constantly updates the content by calling the *getNews* operation all 60 seconds.

## 6. RELATED WORK

Our concept for annotating services with the goal to support the generation of user interfaces for interactive service-based applications are influenced by several research areas.

Annotations focusing on semantic meta-information for services are supported by approaches like Semantic Annotations for WSDL (SAWSDL) (Kopecky, et. al. 2007), Web Service Modeling Ontology (WSMO)( Roman et. al. 2006), and Ontology Web Language for Services (OWL-S) (Web-Ontology Working Group 2005). OWL-S and WSMO support the specification of the semantics of service operations, parameters and return values based on knowledge represented in ontologies. SAWSDL is a concept for annotating service descriptions with semantic meta-information. The named approaches influenced the specification of our UI related annotations to some degree, since the definition of semantics is part of the hints necessary for UI generation.

Approaches for generating suitable user interfaces for web services are described in WSGUI (Hori 2000) and Dynvoker (Spillner, J. et. al 2008). Both approaches focus on the creation of UIs for single services. They are based on an inference mechanism which uses the technical web service specification to generate a UI, mainly for the input of parameters and the presentation of processing results. Annotations containing meta-information about layout, UI controls and labels are used in addition to improve the quality of the generated UI. Further, purely inference based projects are the open source library Xydra (Chipara and Slominski 2003), the Dynamic SOAP Portlet (Dynamic SOAP Portlet) and SOAPClient (SOAPClient). In (Steele 2005) a concept for dynamic creation of multimodal UIs using XForms and VoiceXML elements infered from WSDL is described. The transformation to concrete UI representations is based on XSLT. All these approaches are dependent on particular technologies and do not propose a generic, service technology independent solution as required for our project.

Some existing development tools such as the XML Forms Generator (XML Forms Generator) provides as an Eclipse plug-in a tool which already supports the generation of UIs for composed services. It analyses WSDL service specifications and generates XHTML based UI descriptions with associated CSS style sheets. The NetBeans IDE 7 (NetBeans) offers a comparable functionality for REST-based. It offers a forms inference tool for testing services during the development time.

For the specification of abstract UI descriptions several languages, such as DDL (Springer and Göbel 2002) (Feldmann, et. al. 2009), UIML (Abrams et. al. 1999) or TERESA (Mori 2004) have been developed. In our approach of annotating UI related meta-information to services we adopt elements of these languages.

## 7. CONCLUSION AND FUTURE WORK

This paper discusses the current limitations of automatic user interface generation for services and service-based applications, and based on the findings we propose a set of annotations which are attached to services to improve the results of the UI generation. Some annotations are also designed to help developers during the development process of interactive service-based applications. We developed a total of 22 annotations, divided in three groups: visual elements, behavioral elements and relational elements. We used the example of a CRM application to explain the usage of six annotations.

Currently a prototype is under development that leverages our annotations to build UIs for annotated services. First tests with this prototype in a user study received positive feedback. Most of the users rated the

resulting UIs as well-arranged and comprehensible. Also the functionality added by the behavioral annotations was well received. An extensive comparison of automatically generated UIs for services with and without annotations has still to be done. Furthermore, tests with complex service interfaces as can be found in the enterprise SOA environment have to be made to prove the feasibility of our approach also in that area.

The next step foreseen is to further enhance the exploitation of our annotations in the UI generation and service composition to build applications, and possibly extend the set of annotations. Services are composed on a functional level and on the UI level. Service composition on the functional level incorporates e.g. the definition of a data flow between service elements of several services, or the invocation of a service operation. The composition on the UI level denotes the arrangement of the UI representations of several services to build one consistent application UI, including the navigation between several pages of an application.

# REFERENCES


Abrams, M., et. al.. UIML: An Appliance-Independent XML User Interface Language, in Proc. EighthWWWConf. 1999.

Chinnici, R.; et. al.; WSDL Working Group: Web Service Description Language. http://www.w3.org/TR/wsdl20/. 2007. Last access 31/July/2009.

Chipara, O.; Slominski, A.; Xydra generic client. Available in: http://www.extreme.indiana.edu/xgws/xydra. 2003. Last access 31/July/2009.

Dynamic SOAP Portlet. Available in: http://soap-portlet.sourceforge.net. Last access 31/July/2009.

Feldmann, M., et. al.. An Intergrated Approach for Creating Service-Based Applications. In Proceedings of the 12th IFIP TC13 Conference in Human-Computer Interaction (INTERACT'09), to appear. 2009.

Hori, M.; et. al.: Annotation-Based Web Content Transcoding, 9th International World Wide Web Conference (WWW9), Amsterdam, Niederlande, 15.-19. Mai 2000.

Kopecky, J., et. al., "SAWSDL: Semantic Annotations for WSDL and XML Schema", IEEE Internet Computing, vol. 11, 2007, pp. 60-67

Mori, G., et. al.; Design and Development of Multi-Device User Interfaces through Multiple Logical Descriptions, in IEEE Transactions on Software Engineering, August, Vol.30, N.8, IEEE Press pp.507-520. 2004

Nestler, T., et. al.; Service Composition at the Presentation Layer using Web Service Annotations. In Proceedings of the ComposableWeb'09 Workshop at ICWE, to appear. 2009

NetBeans. Available in: http://www.netbeans.org. Last access 31/July/2009.

Roman, D.; et. al.; Reference for WSMO's upper level elements: http://www.wsmo.org/TR/d2/v1.3. 2006. Last access 31/July/2009.

ServFace Research Project: http://www.servface.eu/

SOAPClient. Available in: http://soapclient.com. Last access 31/July/2009.

Spillner, J. et. al.; Ad-hoc usage of web services with dynvoker. In P. M¨ah¨onnen, K. Pohl, and T. Priol, editors, ServiceWave 2008, LNCS 5377, pages 208–219, Madrid, Spanien, 11 2008. NESSI, Springer.

Springer, T., Göbel, S.: A Modular Adaptation Framework for Single Source Publishing; In: Pedro Isaías (Ed.), Proc. of the IADIS International Conference WWW/Internet 2002, ISBN: 972-9027-53-6, IADIS Press, Lissabon, Portugal, pp. 11-19, 2002; 2002

Steele, R.et. al.; Mobile web service discovery and invocation through auto-generation of abstract multimodal interface. itcc 2, 35–41. 2005

Web-Ontology Working Group. OWL-S: Semantic Markup for Web Services. Technical Report, World Wide Web Consortium. 2004.

XML Forms Generator. Available in: http://www.alphaworks.ibm.com/tech/xfg. Last access 31/July/2009.