User-Centric Composition of Service Front-Ends at the Presentation Layer

Tobias Nestler, Lars Dannecker, and Andreas Pursche

SAP Research Center Dresden
Chemnitzer Str. 48, 01187 Dresden, Germany
{tobias.nestler,lars.dannecker,andreas.pursche}@sap.com

Abstract. The emerge of web services in Service-Oriented Architectures (SOA) within companies or at the global internet offers new ways for the creation of web applications. Even though the composition of services via business processes are covered by existing tools and solutions, concepts for a lightweight service consumption are still in a preliminary phase. The complexity of state-of-the-art SOA technology prevents users with limited IT skills getting easy access to web services and their offered functionalities. This paper presents a user-centric design approach to model and create simple service-based applications in a graphical way without being necessary to write any code.

Keywords: Service Composition at the Presentation Layer, UI Integration, Service Front-ends.

1 Introduction

The introduction of Web 2.0 offers users the capability to take part in the development of the WWW. Non-technical users are able to create web pages in form of blogs or customize web pages such as iGoogle [1] to serve their daily needs. The next steps towards a user centric design of web applications are mashups that combine the philosophy of SOA and approaches of end user development [2]. This approach is dedicated to data aggregation and highly relies on computing knowledge and skills of the end user. The graphical composition style facilitates user empowerment by aggregating web feeds, web pages and web services from different sources using special builders. However, these existing approaches lack several concepts to support a real end user driven application development [3].

The concepts presented in this paper follow the approach of service composition at the presentation layer enhanced by user interface (UI) related service annotations [4]. We adopt the idea of integration at the presentation layer [5] to compose services by combining their presentation front-ends, rather than their application logic or data [6]. A design-time authoring tool, the ServFace Builder, utilizes the mentioned concepts and aims to empower even non-programmers to create their own service-based application. Following our preliminary investigations, this paper discusses the following contributions:

A. Dan, F. Gittler, and F. Toumani (Eds.): ICSOC/ServiceWave 2009, LNCS 6275, pp. 520–529, 2010.

- We propose a mechanism to visualize the service front-ends already during the design time of the application development. An inference mechanism uses the information gained from the original service description and the attached annotations to create a UI for each service operation. In addition, also common usability recommendations were formalized and integrated to ensure the generation of usable UIs (see Sec. 3).
- We propose a graphical lightweight service composition approach in order to model data flows and control flows at the UI level. The modeling of data flows will be done by the graphical connection of single UI elements of the front-ends. In addition, the paper presents two ways of defining a control flow to create multi-page applications (see Sec. 4).
- We integrate the mentioned concepts into our authoring environment and evaluate their usability and acceptance in form of a user study (see Sec. 5).

2 Service Composition at the Presentation Layer

The general approach of service composition at the presentation layer aims to support non-programmers in the design and creation of simple service-based applications. The target user group refers to end users in general and to knowledge workers and skilled web users in particular. The composition process is fully integrated in a three step development methodology presented in [7].

The composition is based on annotated services that act as the foundation for the designed applications. The annotations are reusable information fragments attached to the service description (like WSDL or WADL), which are typically not available for an application developer or service composer. They are created by the service developer and stored in an annotation model based on a formally defined meta-model. Annotations provide extensive additional information covering the visual appearance of a service, the behavior of UI-elements and relations between services to further improve the visual appearance of resulting composite applications. The following examples explain the use of annotations:

- **Annotations defining visual appearance:**
  - Visual Property: This annotation adds information about the designated visual appearance of a service element; e.g. if a parameter is a string that represents a password, this parameter should be displayed obscured.
  - Unit and Conversion: This annotation assigns a unit to a specific service parameter. In addition, with a set of conversion rules it is possible to offer one value in different units and convert between them.
- **Annotations defining the behavior of UI-elements:**
  - Suggestion: This annotation assigns an external suggestion service to a specific service element to provide a list of suggested values while typing.
  - Validation: To proof the correctness of entered data, this annotation defines rules entered data is checked up on.
- **Annotations defining the relations between services**
  - Bundle: A set of disjunct services or service operations that beneficially work together, can be defined with this annotation.

In general, the 21 currently available annotations should thus be seen as a kind of knowledge transfer from the service developer to the service composer to facilitate the understanding and simplify the composition of web services.

During the design time the information gained from the service description and the annotations are used to create a service front-end, which is a UI for a specific service operation (detailed description in Sec. 3). The user, in his role as a service composer and application designer, interacts with these front-ends only to create the desired application in a kind of WYSIWYG (What you see is what you get) principle. No technical knowledge about service composition is required to build an application, so the user can model and layout his application in a graphical way without being necessary to write any code. An application consists of several pages that can be connected with each other to define a navigation flow through the final application. Each page acts as a container for the front-ends and represents a dialog visible on the screen. The complete design process including the integration and composition of the front-ends (detailed description in Sec. 4) is supported by a web-based authoring tool, the ServFace Builder. Figure 1 shows a screenshot of the current prototype. The ServFace Builder supports the user in the design of multi-page applications and the composition of the service front-ends. An internal object model represents the current modeling state and supports the generation of executable applications for different target platforms. This application model is constantly updated according to user changes and its serialization serves as input to the model-to-code generation process.

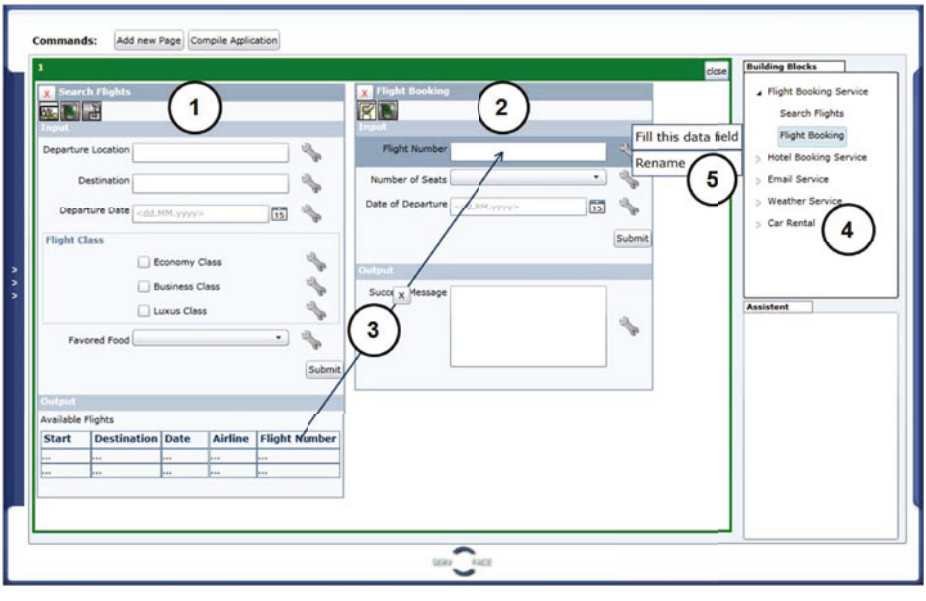

Fig. 1. Screenshot of the ServFace Builder

3 Generating Service-Front-Ends

As mentioned above, services and in particular the operations of those services are represented by generated service front-ends. Number 1 and 2 of figure 1 are examples of service front-ends for service operations from a service called "flight booking service". A service front-end representing a service operation comprises of a nested container structure including a root operation container and an interaction container for the input and output parameters of a web service. Those interaction containers comprehend the visualization of the operation parameters that are the key elements to later invoke the service when using the final composite application. Figure 2 illustrates the five step generation process:

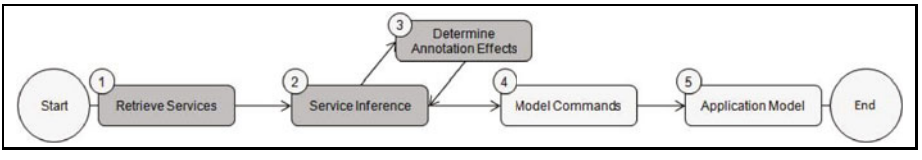

Fig. 2. Generation process for service front-ends

1. Retrieve service elements and structure
Since a remote service repository manages the services, all available information about a service need to be retrieved from the repository at first. To avoid extensively communication affords the information about a service are stored in a local data model. This model represents all information necessary for the front-end generation process. All available services including their operations are presented to the user within the *Building Blocks* browser (Fig. 1 No. 4).

2. Analysis of the services
To infer the structure and the elements of a service operation, an *Inference Engine* parses the model that represents the service and analyses the particular service operation. It generates UI elements for each service element that needs to be displayed within this service-front-end. To infer which UI element should be used for a specific service element the inference engine considers the following:

- parameter type (input or output)
- base data type
- data type enhancements and restrictions
- occurrence definitions

Complex data types need special consideration because they can be arbitrarily nested or can contain recursions. The inference engine does not follow recursions arbitrarily to avoid infinite loops, but rather aborts further inspection after a certain number of loops. For input parameters complex data types are represented by a nested structure, dividing the complex type into its child elements. For output parameters complex data types are presented by tables as standard

output format or lists. Depending on the depth of a nested complex type, the table could contain a expand mode to show complex child elements. Minimum and maximum occurrences are taken into consideration by redundantly displaying the data type element or allowing multi-selection in the range of the occurrence. As a result the inference engine creates model commands (4) that are executed by the API of the underlying application model. The execution affects the current state of the application model (5), e.g. by including new elements or changing existing ones. The following example explains this process:

A simple flight search operation needs two inputs to search available flights. A destination (data type: string) and a departure date (data type: date). The return value is a complex type containing the data type elements "FlightNo.", "Airline", "Price" that all are defined as string data types. Firstly, the inference engine creates the container structure including an operation root group, an input group and an output group. Secondly, both input parameters are inferred. Because the destination parameter is defined as string a textbox is chosen, whereas the departure date is a date type and therefore it is visualized by a calendar widget. Thirdly, for the output parameter as a complex type a structured output e.g. a table is chosen to display the results. Finally, an execution button is created to invoke the web service in the resulting composite application.

3. Fetching additional effects

The particularity of this generation process is the consideration of additional information provided by the ServFace service annotations. Furthermore, a consolidation of several HCI guidelines (e.g. provided by Apple or Microsoft) or perceptions of usability experts like [8]) result in a set of formalized UI design recommendations, which improve the service front-end visualization. Whenever the inference engine detects annotations attached upon a service element it requests a description for each annotation, which specifies the effects of this particular annotation. Possible effects include e.g. the creation of additional UI elements, the addition/change of UI element properties, the change of the appearance of UI elements, the restriction to a set of valid values, etc. The annotation effect determination component of the inference engine creates a description object for the annotation. The inference engine analyses the description and includes the effects of the annotation in its analysis process.

A continuation of the flight search example above explains this process:

The parameter of the destination input field is defined as string. Without the influence of annotations it is displayed as a standard textbox handling arbitrary phrases. The addition of an *Enumeration* annotation defines a collection of valid values e.g. a selection of all available destination airports. To avoid false inputs the inference engine would not choose a textbox as representing UI element, but rather a combobox to restrain the input possibilities. In addition a *Feedback* annotation is added to all parameters defining a name readable for humans. This causes the inference engine to not determine the parameter name from the service description that does not necessarily provide an understandable name for its elements, but to use the name provided by the *Feedback* annotation.

The service inference engine as well as the annotation effect determination take the formerly mentioned UI design recommendations into consideration. These recommendations influence the choice which UI element is used to represent a service parameter, or change the configuration of those UI elements. The following example lists the consolidated UI recommendations for the *Enumeration* annotation:

- Use a Radio-Button-Group whenever there is a choice between 2 or 3 values.
- Use a Drop-Down-List whenever there are more than three but under 20 valid values.
- Use a ComboBox (Drop-Down-List with search capability) whenever there are more than 20 but under 200 valid values.
- Use a Textbox when there are more than 200 valid values.
- Show at least 8 items at the same time before a scrollbar appears whenever a Drop-Down-List or ComboBox is used.
- Sort the items alphabetically whenever there are more than 12 items.

The generation of UIs for web services were also analyzed in former research projects like WSGUI [9] or Dynvoker [10]. Both approaches also use additional information to enhance the visualization result, in particular GUIDD [11]. Unlike these approaches the presented concept not only focuses on the generation of UIs to directly invoke services at run time, but to use those UIs already at design time to allow the composition of several services (description in Sec. 4). Furthermore, the used ServFace service annotations exceed the expressiveness of the GUIDD-annotations and a consolidation of recommendations from several end user design guidelines is used to further improve the ease of use of the generated front-ends.

4 UI-Centric Composition via Service Front-Ends

Nowadays, there are already many mashup-editors available like Yahoo Pipes [12] or IBM Mashup Center [13] (detailed overview provided by [14]), where the majority focuses on the processing of data. A quite new area for mashups is the definition of simple (business) processes as discussed in [15]. This kind of mashup enables the user not only to aggregate data from different sources but also to combine the invocation of several service operations. Following the previously introduced example, the user is now not only able to search for flights but rather can create an application to book them. To compose such a service-based application, the user needs to define the order of execution of the used service operations and the data flow among the operations.

The common visualizations of data flows in mashups are ports (often displayed by little circles) for each variable and arrows from the source to the target port. The activity of connecting those ports is called "Wiring" [14]. In contrast to this approach, the main idea of our UI-centric composition is to directly use the input and output UI elements of the generated service front-ends. To model a data flow between two different front-ends, the user has to click on the target UI element that should be filled with the data (see context menu in Fig. 1 No.5).

To define the source element he directly clicks on the UI element that provides the designated data. Finally, the tool presents the connection in a visual way (see Fig. 1 No.3). The selection of the connection point via UI elements leads to a large target area for the mouse courser and makes it easy to hit (according to Fitts' Law [16]). The direct use of UI elements for the composition benefits from the fact, that form-based applications are well-known and most users are familiar with them. Therefore, the barrier to use the editor is lower than in tools, which require additional technical concepts like ports. In addition, we assume that it is easier for an end user to handle the definition of data flows within the presentation layer.

For the definition of the order of execution we have two views offering different levels of expressiveness. Both use pages, which can contain front-ends for service operations. The transition from one page to another will cause the invocation of the corresponding web services. The first approach is intended for users who are new to the tool and want to create applications simply and quickly. Therefore, thumbnails of the pages are listed (Fig. 3 a) similar to the well-known slide listing in Microsoft Powerpoint. The order of execution is defined by the order of the pages in the list view. For example moving a page B before another page A, means that the service operations associated with page B are executed before the ones of page A. With this approach you can only define a sequential order without branches in the execution flow, but this is suitable for many simple processes and easy to use. The user has not to consider the details of the process flow. For an experienced user who wants to define more complex processes we provide a detailed flow view showing the transitions between the pages (Fig. 3 b). In this view he is able to create his own transitions and to define alternative flows for an application. We do not only use nodes in this view as it is known from process diagrams, but also the thumbnails of the pages. The user

(a) Sequential Page View

(b) Page Flow View

Fig. 3. Different Page Views

can click on a thumbnail and open it, to see the service front-ends contained by this page, integrate new ones or connect UI elements as described above.

5 Evaluation

To evaluate the presented approaches a user study was conducted. While for the first study [17] only some mockups and a UI prototype were available, a fully functional prototype could be used in course of this evaluation. The goals of the evaluation were to evaluate the acceptance of the concepts, the ease of use of the tool, the necessary period of vocational adjustment and other aspects regarding the usability of the ServFace Builder.

Since the aspired end user should not necessarily have advanced IT-experience, we ensured that the background knowledge of all participants was not related to computer science. The main scope of the evaluation was divided into two parts. The first part was an exploratory evaluation of the ServFace Builder with a duration of approx. 15 minutes. The participants had the chance to try out the tool without further tasks or guidance. They have not received extensive information beforehand either. However the observers stood by to assist or answer questions whenever necessary. As shown in figure 4 most users stated that the bench of the ServFace Builder is very clearly structured (92%) and the functionality is self-explanatory (67%). This means that the usage of the tool should be easily understandable even without a previous tutorial or extensive explanations. However many user (67%) had problems getting started right after they reached the first tool screen. This leaves room for further optimizations regarding the start-up process and the user guidance coordinating the first steps.

In the second part of the evaluation the users got the task to adopt the role of a company worker who tries to ease the business travel booking process. They had to create a composition that satisfies the requirements of the given scenario description. A selection of the results of the second part is presented in figure 5. As expected, the participants had less trouble to get started with the composition process now (92%). Both, the participants and the observers had the impression that in most cases the design process came more naturally to the participants. This impression was confirmed by the comments of the participants: 92% stated that the design process of a multi-page composite application is straightforward and all participants declared that the front-ends are self-explanatory (75%) or

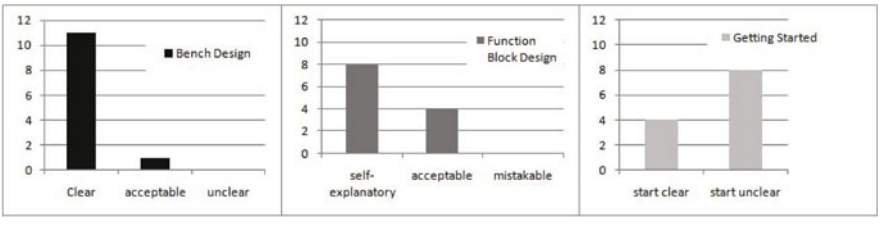

Fig. 4. Evaluation results part one

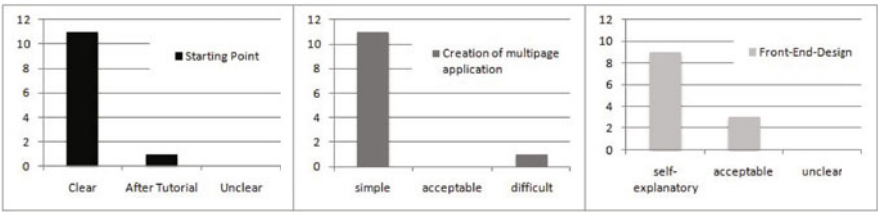

Fig. 5. Evaluation results part two

at least understandable (25%). This points out that the users of the tool are able to handle the composition process quite well with little familiarization. The 15 minutes exploratory try-out was sufficient to enable the participants to design the required application without further difficulty. Therefore the period of vocational adjustment is rather short even for users without IT-background. In contrast many users (67%) criticized the lack of guidance provided by the ServFace Builder. In addition, the Drag&Drop concept to add the service operations to the canvas was rated neither intuitive nor comfortable.

The flow view (Fig. 3 b) was used as the central view showing the pages during the user study. But only 25% of the participants asked for the possibility to influence the order or to define alternative flows. The observers noticed that most participants just added new pages and integrated service front-ends in the linear order they had created the pages before. This motivates the modification of the ServFace Builder, the simple sequential view should be the standard view on start-up and the flow view should be available on demand.

6 Conclusion and Future Work

The paper presented two concepts supporting the approach of service composition at the presentation layer. The generation and composition of service front-ends offers new ways in designing service-based applications for non-programmers. An evaluation with a working prototype showed that most users handled the tool without extensive difficulties after a short familiarization period. Most concepts pointed out to be understood and accepted by the users. However, there are further fields of improvement. An extended guidance and a better way to get the composition process started are two important examples. In addition, the visualization process of the service front-ends could be enhanced by an extended support for different platforms. An expert-based evaluation as well as a third iteration of the user study are planned to estimate the recommended changes gained from the presented evaluation.

Acknowledgment

This work is supported by the EU Research Project (FP7) ServFace. In addition, we would like to thank the participants of the user study and the supervisors of our theses Marius Feldmann (L. Dannecker) and Gerald Hübsch (A. Pursche).

References

1. iGoogle (2009), http://www.google.de/ig?hl=de&source=iglk
2. Hoyer, V., Stanoevska-Slabeva, K.: The Changing Role of IT Departments in Enterprise Mashup Environments. In: 2nd International Workshop on "Web APIs and Services Mashups" (2008)
3. Nestler, T.: Towards a Mashup-driven End-User Programming of SOA-based Applications. In: 10th International Conference on Information Integration and Web-based Applications & Services (2008)
4. Nestler, T., Feldmann, M., Preussner, A., Schill, A.: Service Composition at the Presentation Layer using Web Service Annotations. In: ComposableWeb 2009 Workshop at ICWE 2009 (2009)
5. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A Framework for Rapid Integration of Presentation Components. In: WWW 2007, Banff, Canada (May 2007)
6. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding UI Integration: A survey of problems, technologies, and opportunities. In: IEEE Internet Computing (May/June 2007)
7. Feldmann, M., Janeiro, J., Nestler, T., Hübsch, G., Jugel, U., Preussner, A., Schill, A.: An Integrated Approach for Creating Service-Based Interactive Applications. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009. LNCS, vol. 5727, pp. 896–899. Springer, Heidelberg (2009)
8. Krug, S.: Don't Make Me Think! A Common Sense Approach to Web Usability, vol. 2. New Riders, Indianapolis (2006)
9. Kassoff, M., Kato, D., Mohsin, W.: Creating GUIs for Web services. In: IEEE Internet Computing, CA, USA, vol. 7 (2003)
10. 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)
11. Kassof, M., Spillner, J.: GUIDD: Standard and Specification (2006)
12. Yahoo! Pipes (2009), http://pipies.yahoo.com
13. IBM Mashup Center (2009), http://ibm.com/software/info/mashup-center
14. Hoyer, V., Fischer, M.: Market Overview of Enterprise Mashup Tools. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 708–721. Springer, Heidelberg (2008)
15. de Vrieze, P., Xu, L., Bouguettayay, A., Yangz, J., Chenx, J.: Process-oriented enterprise mashups. In: Grid and Pervasive Computing Conference (2009)
16. Fitts, P.M.: The information capacity of the human motor system in controlling the amplitude of movement. Journal of experimental psychology (1954)
17. Namoune, A., Nestler, T., Angeli, A.D.: End User Development of Service-based Applications. In: 2nd Workshop on HCI and Services at HCI 2009 (2009)