OntoCompo: A Tool To Enhance Application Composition

Christian Brel, Anne-Marie Dery-Pinna, Philippe Renevier-Gonin, Michel Riveill
I3S Laboratory (Université Nice-Sophia Antipolis / CNRS)
930 route des Colles, BP 145
06903 Sophia Antipolis Cedex, FRANCE
{brel,pinna,renevier,riveill}@polytech.unice.fr

Abstract. Mash-ups emerged through the web 2.0 to juxtapose several applications and use them together. The next step after juxtaposition is the composition of existing applications to build a new one. A solution of this being born need is the reuse of parts from formers applications. To perform this composition and reuse in an easy and comfortable way, we propose a tool based on several extensions of selection to help the developer during his composition.

Keywords: Application Composition, Semantic Annotation, CBSD, UI, task model

1 Introduction

The advent of web 2.0 and the apparition of a lot of “applications stores” introduce implicitly new needs for users and developers faced to this set of applications disposed on the web. Mash-up solutions for example allow them to juxtapose several applications and use them together. They can have ideas for new functionalities creating a new application combining existing ones. Adapting applications to users' requirements may be done through composition of applications. Tools for composing former applications (and probably corresponding source codes) should introduce developers' comfort and a reduction of the time-to-market for new applications by recycling former applications.

In this paper, we present the tool OntoCompo dedicated to easily realize new applications by composition of their User Interface. This tool deals with component-oriented applications respecting a separation in two parts: the User Interface (UI), visible and well-known part of the application and the functional core (FC), underground part of the application. Due to this clear separation, the composition process lets the possibility to the developer to build the new application selecting, extracting and positioning UI part of former applications, one after another [**2**]. So we focus on the connections between UI, FC and tasks. We consider that a composition driven by a checked selection is a guarantee to preserve the global consistency of the final application. So we choose to help the developer for broadening selection.

In the next section, we describe related works and we underline our originality. Then we present the hypothesis of our work and our tool for application composition.

2 Related Works

As we aim at composing applications by manipulating their UI, we have to decompose UI, i.e. describe UI in order to deal with sub-parts of former UI. The description of an UI both involves (1) description of its structure (like UIML [1], ALIAS [7], UsiXML [5] or MARIA [8]) and (2) the spatial positioning of these components (like in different layouts used in the UI toolkits).

To manipulate applications in order to compose them, there are currently three main approaches: (i) the composition could be triggered by the functional part as in [7], (ii) the composition could be triggered by the users' goals (i.e. tasks) as in [8] and (iii) the composition could be triggered by the UI as in [4].

Each trigger addresses a specific problem of composition: presentation and layout considerations at the UI level, behavior of the application at the functional level, users' needs at the task level. These works do not reuse complete architecture of the former applications. Either they compose and reuse UI as first concern without any consideration of the links between UI and the functional part either their first concerns are functionality or task and provide the new application by (re-)generating UI.

Our originalities are (i) to consider links between UI, tasks and functionalities, (ii) to lead the developer by suggesting him and asking him about elements to keep for aiming at composition consistency and (iii) to reuse existing UI in order to preserve former developments, former designs and former practices. Our tool, OntoCompo, helps the developer of application for reusing existing applications to constitute his new one. We purpose the developer to select UI elements he wants to keep and suggest him extensions for his selection in order to obtain a new functional application after composition.

3 Hypothesis on Former Applications

To be able to reuse elements of the former application, we need a software organization authorizing selection, extraction and rejigging of such elements. We opt for applications developed with FRACTAL components [1]. For reusing of former applications parts, we use: (i) component-based software development to manipulate functionality assemblies and (ii) component-based UI with Java Swing JComponent encapsulated in FRACTAL component in order to manipulate concrete UI parts. Applications are not expected to be provided with sources. Indeed FRACTAL components are seen like black box and inputs and outputs software *interfaces* are available. To reuse existing applications, our hypothesis is to let the developer doing composition through the interfaces of applications. So, our approach is to enhance links and to extend connections between UI elements and Functional Core elements. That strengthening is based on the Task Model (TM). We use semantic annotations

(using OWL Light[1] language) for the description of applications. So the OWL Light description includes the description of the task model (an OWL representation of CTT [6]), the description of the UI elements (an OWL representation of MARIA [8]) and their layout and the description of functionalities. The OWL Light description also includes links between tasks and functionalities, links between tasks and UI elements, links between functionalities and the *concrete* FRACTAL component, links between UI elements and the *concrete* FRACTAL component.

4 Composing thanks to Extensions of Selection

The simple selection of a part of an application is the direct manipulation. By a click on an UI element, the developer can select it in order to extract it later. Selected UI elements are graphically highlighted. That simple selection is extended for performing complex selections or aiming at verifying consistency.

First, there is ***the layout extension***. With the height toggle buttons for selected extension directions, the developer has the possibility to broaden the selection. SPARQL[2] queries are parameterized with the current selection and with each chosen directions. Such a query returns the relevant fractal component identifiers.

Secondly, there is ***the (container) parent extension***. It's also about queries layout of application to obtain the parent container of last selected UI component in current selection. This extension allows the developer to be more efficient on his selection of all elements in a container potentially "hidden" by its contents.

Thirdly, there is the ***task extension***. Each UI element is linked with a task described with semantic annotations. From the last selected component, we use SPARQL queries to obtain the task linked to it. From each returned tasks, we query semantic annotations to obtain all UI elements linked with this task.

Finally, there is ***the functionality extension***. UI elements are directly linked to functionality but also through tasks. Since a task may be connected to several functionalities, it is possible to extend the selection to each part of the application by following these links. We start with selected UI elements. Thanks to SPARQL queries, we go back "up" to related tasks and then "up" to related functionalities. From these functionalities, we go back "down" to UI elements.

Each of these extensions can be activated by the developer. He is free of combination between all proposed extensions. To help him and to lead him towards to a coherent composition, we develop a help selection. This help is a guide for the developer during all selection process. For each UI element, several questions suggest to the developer different possibilities for extending his selection. The developer can partially or fully use that help (guided by tasks and/or by functionalities and/or by layouts) to perform his selection.

[1] OWL Web Ontology Language. http://www.w3.org/TR/owl-features/

[2] SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/

5 Conclusion

To conclude, with OntoCompo, we provide a solution to compose application from a manipulation of UI. We help the developer during the composition with all proposed selection extensions. Those selection extensions based on suggestions to enhance the reused part of former applications lead to an usable application. The developer being able to choose his entry point (UI layout, functionalities or tasks) to perform his extensions, we are now planning developer evaluation to validate the different extension. Once that evaluation performed, we will work on a new step in the composition process about merging application elements (UI elements or functionalities).

References

1. Abrams M., Phanouriou C., Batongbacal A., Williams S., Shuster J. 1999 UIML: An appliance-independent XML user interface language. In proceedings of the 8th World Wide Web Conference (WWW), pages 617-630, Elseiver.
2. Brel C., Renevier-Gonin P., Occello A., Pinna-Déry A.-M., Faron-Zucker C., Riveill M.. "Application Composition Driven By UI Composition" in Proceedings of the Human Computer Software Engineering 2010 (HCSE 2010), IFIP International Federation for Information Processing, pages 198--205, LNCS, oct 2010
3. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V. and Stefani, J.-B. (2006), The FRACTAL component model and its support in Java. Software: Practice and Experience, 36: 1257–1284. doi: 10.1002/spe.767
4. Fujima J., Lunzer A., Hornbæk K., Tanaka Y., 2004. Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access, In Proceedings of UIST 2004, pages 175-184, Santa Fe, NM.
5. Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Florins M., and Trevisan D., 2004. Usixml: A user interface description language for context-sensitive user interfaces. AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" UIXML'04, pages 55–62.
6. Mori G., Paternò F., Santoro C., 2002. Ctte: Support for developing and analyzing task models for interactive system design. IEEE Transactions on Software Engineering, pages 797–813.
7. Occello A., Joffroy C., Pinna-Déry A.-M., Renevier-Gonin P. and Riveill M., 2010. Experiments in Model Driven Composition of User Interfaces. In 10th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'10), volume LNCS 6115, pages 98-111, Amsterdam, Netherlands. Springer-Verlag.
8. Paternò F., Santoro C., and Spano L. D., 2009. Maria: A universal, declarative, multiple abstraction level language for service-oriented applications in ubiquitous environments. In Computer-Human Interaction (TOCHI), volume 16.