Application Composition Driven by UI Composition

Christian Brel, Philippe Renevier-Gonin, Audrey Occello, Anne-Marie Déry-Pinna, Catherine Faron Zucker, Michel Riveill

Application Composition Driven By UI Composition

Christian Brel, Philippe Renevier-Gonin, Audrey Occello, Anne-Marie Déry-Pinna, Catherine Faron-Zucker and Michel Riveill,

I3S Laboratory (UMR 6070 - Université Nice - Sophia Antipolis et CNRS)
930 routes des colles – BP 145
{brel, renevier, occello, pinna, faron, riveill}@polytech.unice.fr

Abstract. Ahead of the multiplication of specialized applications, needs for application composition increase. Each application can be described by a pair of a visible part –the User Interface (UI) –and a hidden part –the tasks and the Functional Core (FC). Few works address the problem of application composition by handling both visible and hidden parts at the same time. Our proposition described in this paper is to start from the visible parts of applications, their UIs, to build a new application while using information coming from UIs as well as from tasks. We base upon the semantic description of UIs to help the developer merge parts of former applications. We argue that this approach driven by the composition of UIs helps the user during the composition process and ensures the preservation of a usable UI for the resulting application.

Keywords: User Interface Composition, Application Composition,

1 Introduction

There are more and more software tools: on the web, on Smartphones, on laptops, etc. Having so many widgets is interesting; however, to reach a friendly use, there is a need to compose them. For example, a Smartphone can provide its user with a diet list and an application that gives restaurants close to her and their menus. She would probably enjoy a second application filtering or emphasing dishes from her diet list.

To construct new applications by reusing other application sub-parts is a key challenge of Software Engineering. This is a mean to speed up development cycles. An interactive systems is composed at least of a functional part, usually called Fonctional Core (FC),and a User Interface (UI). Moreover, in the HCI research field, there is a strong recommendation of using a Task Model (TM) during requirements analysis. The TM describes the needs and the procedures to achieve these needs. The TM is not often explicitly implemented, but it can express the relationship between FC and UI entities. We choose to use UI as primary artifacts of the composition process because UI are the parts of applications manipulated by both developers and ergonomic designers. We aim at enabling them to reuse existing UI for creating new applications while preserving user requirements of individual original systems and keeping some of the links between the FC part and the UI part in the resulting system.

In this paper, we propose to combine information at the three levels: FC, UI and TM. For this, we base the composition process on the selection, extraction and placement of the existing application's UI as elementary composition actions to impact underlying task trees and FC part.The remainder of this paper is organized in 5 sections, respectively, the description of other UI composition works, the presentation of our model used in our composition, the overview of our composition process, the description of our implementation of the global process and finally the conclusion.

2 Related Work

This section presents related work on UI composition grouped by their entry point in the composition process according to the application cutting: the Functional Core (FC), the Task Model (TM) and the User Interface (UI). Each entry point addresses a specific problem of composition: presentation and layout consideration at the UI level, behavior of the application at the FC level, user needs at the TM level. We classify works related to UI compositions according to their approach: an "X" in the Table 1 means that corresponding work explicitly takes into account this part.

Table 1. Classification of composition approaches.

	FC	UI	Tasks
Developing adaptable user interfaces [9]		**X**	
Amusing [8], ComposiXML [3]		**X**	
C3W [11]		**X**	
Task Models Merging [4]			**X**
Servface [7]	**X**		**X**
Compose [2]	**X**		**X**
Scenarios [12]		**X**	**X**
SOAUI [9], ALIAS [6], Transparent Interface[13]	**X**	**X**	

We group related works in four categories:

- Works only considering UI composition, either for defining specific toolkit for adaptive UI [9], either based on abstract definition of UI [8,3] or either adopting end-user programming [11],
- Works only considering TM composition (composition of two task trees [4]),
- Works deriving Tasks in FC composition and later in UI composition, because of generation UI from service annotation [7] or thanks to specific adaptable couple FC-UI [2] or deriving Tasks in UI [12]
- And Works considering both FC and UI composition. The main goal in [13] is to maintain a stable UI for using a composition of volatile service. The SOAUI approach [9] derives web service composition into UI composition, by searching the best-fitting UI in a repository for each service and then UI composition. The aim of [6] is to deduce the UI composition from the FC composition.

We notice a lack in underlying composition processes. Either the design of original applications' UI with man-crafted properties such as ergonomic or usability is lost, or both FC and UI parts are no longer connected together in the resulting application, or there is no UI reuse. In the context of fast development processes, reusing UI without keeping ergonomic and usability criteria is useless. Loosing links between the UI and the FC parts engenders human interventions to connect the two parts which is error prone and fastidious for large applications.

So we propose in our approach to mix information from all the levels to improve the application composition. The collaboration between the three levels are expressed in a unifying model, we call Enhanced Task Tree (ETT), presented in the next section.

3. Enhanced Task Tree

In our approach, the process is guided by the composition of former UI and by their reuse to build the new application. Our work is based on a model that lets consider information from Functional Core (FC) and UI and from Task Model (TM).

3.1 Connecting Conception and Implementation of the Interactive System

We assume the decomposition in two parts of an application: the FC and the UI. Links between both parts are difficult to analyze in the code, so we use an external description with references to some running objects. We use the task model (TM) as a pivot. The TM is established at design time from requirements and user models. We enhance it with information from the running objects. For each initial application and for each task, we add semantic annotations. In the following, we call "UI block" one piece or a group of pieces of UI. **Fig. 1** shows an overview of the relationships between the different entities: tasks, UI Blocks and FC part. As a result, to implement our model, we need to retrieve from all the composed applications both their tasks description and the links between parts of their UI and parts of their FC. This knowledge is represented in a so-called Enhanced Task Tree.

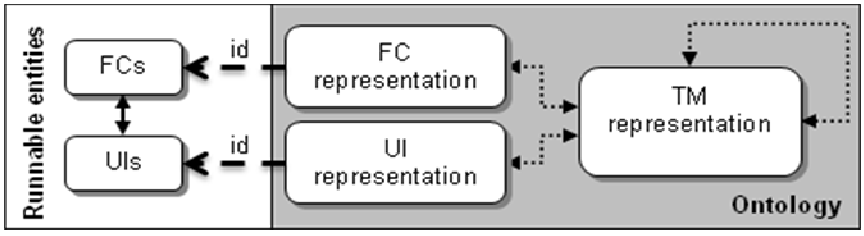

Fig. 1. Links between tasks, UI tree and FC

3.2 Enhanced Task Tree Definition

We define three sets for each application:

- Let UI the set of UI blocks.
- Let T the set of tasks.
- Let FC the set of functionalities.

The sets FC, UI and T of an application are defined by analyzing this application with the aim of extracting knowledge to create an enhanced task tree (ETT). Such a task tree includes the description of UI (blocks and layout) and links with the Functional Core (FC). This analysis is performed by developers.

Based on the ontology presented in the right part of the **Fig. 1**, we represent an ETT by a knowledge graph linking the multiple conception levels: it captures the links between the tasks in T and the UI blocks in UI, and between the tasks in T and the functionalities in FC. A task may be linked to at least one FC or UI entities. A UI (and respectively a FC) is at least linked to one task.

With ETTs, we are able to extract the right part of the reused UI in order to place them in the new UI without losing the links with the FC. ETTs enable composition at different levels that we can express through functions. In the remainder of section 3, we present the composition functions we specified for each of the three levels.

3.3 Selection/Extraction of Tasks at UI level

To represent the links between information, we define three functions:

Let δ a function associating to a UI block the corresponding tasks and its inverse δ^{-1} associating to each task its corresponding UI blocks:

$$\delta : UI \longrightarrow T^{+}; u \mapsto \{t_1,...,t_n\} \qquad \delta^{-1} : T \longrightarrow UI^{+}; t \mapsto \{u_1,...,u_m\}$$

In order to go further than the simple UI hierarchical relationship of container-component, we can identify each UI blocks $\{u_k\}$ connected with a given UI block "u", i.e. $\{u_k, \forall k, \exists j, t_j \in \delta(u) \quad and \quad u_k \in \delta^{-1}(t_j)\}$. That connection is the expression that all UI blocks $\{u_k\}$ are required to perform the tasks associated with the given UI block "u". So it makes sense to extend the selection from the single UI block "u" up to the set of UI blocks $\{u_k\}$, according to the acknowledgment of the developer.

3.4 Selection/Extraction of Tasks at Task level

Let ρ a function associating to each task all the tasks related to it:

$\rho : T \longrightarrow T^{+}; t \longrightarrow \{t_1,...,t_n\} \mid \forall i \in \{1,...,n\}, r(t,t_i)$ where r is a relation representing the temporal operations between tasks and the hierarchy between tasks from the CTT model [5].

Sometime, extracting only one task, like a dialog box to select a file, may have no sense, because the result of the task is not used. So if the developer has selected a task "t" through the UI blocks selection by selecting the set of Ui blocks defined by

$\{u_k, \forall k, u_k \in \delta^{-1}(t)\}$, the developer has to extend that "t" up to ρ(t). By this way, the developer can select some (or all) tasks directly related to "t". So, the function $\rho \circ \delta$ retrieves the tasks attached to a given UI block. The function χ representing the set of UI elements selected by extension of an initial selection:

$\chi : UI^+ \longrightarrow UI^+$

$\chi(\{ui_1,...,ui_n\}) = \{ui_1,...,ui_p\} \mid p >= n \wedge \forall ui_{k \in \{n+1,..,p\}}, \exists j \in \{1,..,n\}, \exists t \in \rho(\delta(ui_j)) \mid ui_k \in \delta^{-1}(t)$

3.5 Selection/Extraction of Tasks at FC level

Let γ a function associating to a task all its corresponding FC elements:

$\gamma : T \longrightarrow FC^+; t \longrightarrow \{fc_1,..., fc_n\} \mid \forall i \in \{1,...,n\}, c(t, fc_i)$ where c is a relation representing the links between a functionality and the task it implements.

Like ρ, γ enables extension of selection, but at a functional level, i.e. at the data processing level. The retrieval of the functionalities attached to a given UI block relies on the functions $\gamma \circ \delta$ and $\gamma \circ \rho \circ \delta$.

4. Composition Process

The goal of the process is to produce a new application resulting from the composition of UI of former applications. The new UI is composed of several parts reused from former UI and possibly of some new consolidating parts like graphical glue used to fill remaining voids.

Through a special UI, the user (i.e. a developer) loads each application containing functionalities (and corresponding UI blocks) to be inserted in the new application. The loading step corresponds to the construction of the multiple level descriptions introduced above. From each application's UI, he selects UI blocks to be reused. Thus, he can compose his new application. He obtains in only one application the different functionalities he wants to keep from the reused applications.

The construction of the new UI is done in three steps iterated during the building of the complete new UI:

1. First, the developer makes a *selection* of pieces of UI to be reused in the new UI. Here we check whether the selected UI block is valid or not. To be valid, a UI block must enable the end user to completely perform one or more functionalities. He may either select an entire screen to add in the new UI, or select a UI block to be reused in the new UI, or select several UI blocks. In that case, the relative positions of the selected blocks in the initial UI are kept and they are placed in a new undividable UI block.
2. If the selected UI block is not valid, we propose the *extraction* of complementary pieces of UI to "validate" the selection. During this extraction step, questions are asked to the developer to help the validation of the UI block. This step constitutes the extension of the selection.

3. Once the selected UI block is valid, the third step consists in the *placement* of it in the new UI through various possible layouts. The selected UI is an entire screen, he has the possibility to place selected screen in the new screen flow. If it is a UI block, he can place it in the screen according predefined layouts that are proposed to the developer to define the placement in a screen or in a group of UI blocks.

5. Implementation: UI for Composing UI

We developed a proof of concept to perform a first validation of the different steps of the process we propose. It is made of (i) a UI to graphically compose several applications and (ii) several well-built applications. By "well-built application" we mean that it is developed with a clear separation between the UI and the Functional part. Moreover, both parts of the application are "*public*": for the UI part, we can explore all UI Components starting from the main window (and its content pane); for the FC part, we get all the called methods, i.e. the Functional part is accessible through a "*façade*". By "well-built application" we also mean that it is provided with an external description, its Enhanced Task Tree (ETT). The whole development is made in Java.

5.1 Enhanced Task Tree

ETTs are represented in RDF[1], the W3C standard for the Semantic Web; the ontology is represented in RDFS[2], the W3C standard for light-weight ontologies. To implement our composition mechanism, we use the Corese [1] semantic web engine to process and query the RDF representations of the different parts of the application. We implement the functions δ, ρ and γ by SPARQL[3] queries over the RDF(S) representation of the ETT.

Our model of tasks is based on CTT [5] and our UI model is based on MARIA [7] (we added some UI elements like graphical glue for the description of UI component tree). In our RDF models, there are references to Java Objects both for UI and FC. Thus, we define a unique ID for each UI component, based on the main class of the application and the place of the UI component in the component tree. For the Functional part, we define a unique ID based on its "*façade*" class and method name.

5.2 Selection and Extraction

We developed a UI for manipulating the different applications the user (i.e. a developer) wants to compose. This tool lets the developer compose his new

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/rdfschema/
[3] http://www.w3.org/TR/rdf-sparql-query/

application, place the different elements and save or load a composition already done. The developer performs the selection step by interacting with the former UI and by controlling those interactions with our tool for manipulating. Indeed, our tool enables to activate or deactivate the interaction in the former UIs (by adding / removing initial graphical event listeners) and the selection process (by adding / removing our own graphical event listeners)

The extraction step is interleaved with the selection step: each time a (group of) UI component(s) is selected, its extraction is determined as a set of questions asked to the developer. Because we have an access of the task tree corresponding to actions performed through the interface, we are able to warn the developer of the need of extracting other components linked to the selected component. The developer can deactivate the questioning.

5.3 Placement

For this step, we propose to place components between each others, through relative positions like “above of”, “on the right of”, “on the left of”, etc…

We express conditions with RDF properties and we transform these conditions in a Java layout. At the same time, the Corese engine deduces relative layouts thanks to a base of 14 inference rules we wrote. For example, from absolute positions of two different UI components, our rules enable Corese to deduce whether the first component is on the left, on the right, above or below the second component. This deduction is necessary to provide a relevant feedback to the developer during the placement step.

6. Conclusion

In this paper, we proposed an original process based on the manipulation of UI that improves the composition result in terms of UI design reuse while preserving the links between FC and UI parts.

Our process is made of three steps: selection, extraction and placement of former UI blocks. Each of these steps uses the enhanced task trees associated to the applications to compose and to build a new task tree keeping some links between the parts of applications. The originalities of the proposed process are: (i) in its starting point (the UI) but with a cover of also Task Model and Functional Core; (ii) in our commitment to reuse former UI (including their design properties). Moreover, its strengths are: (iii) in the possibility to build the resulting enhanced task tree in function of the user actions on the former UI and (iv) in the extraction of the right part of the UI and its placement in the new UI without losing the links with the FC.

Our approach must to be improved before performing test with developers. Indeed, we are working on merging UI blocks, at different levels (FC, TM or UI), according to the compatibility of manipulated entities and by importing adapters given by the developer.

7. Acknowledgments

Our work is funded by the DGE M-Pub 08 2 93 0702 project.

References

1. Corby O., Dieng-Kuntz R., and Faron-Zucker C.. Querying the semantic web with the corese search engine. In 16th European Conference on Artificial Intelligence (ECAI2004), IOS Press, Valencia, Spain, 2004.
2. Gabillon Y., Calvary G., and Fiorino H.. Composing interactive systems by planning. In UbiMob'08, pages 37–40, Saint Malo, France, 28 - 30 mai 2008.
3. Lepreux S., Hariri A., Rouillard J., Tabary D., Tarby J.-C., and Kolski C.. Towards multimodal user interfaces composition based on usixml and mbd principles. Lecture Notes in Computer Science, 4552(134):134–143, July 2007.
4. Lewandowski A., Lepreux S., and Bourguin G. Tasks models merging for high-level component composition. Human-Computer Interaction, Part I, HCII 2007, Lecture Notes in Computer Science (LNCS), 4550:1129–1138, July 2007.
5. Mori G., Paternò F., and Santoro C. Ctte: Support for developing and analyzing task models for interactive system design. IEEE Transactions on Software Engineering, pages 797–813, Aug. 2002.
6. Occello, A., Joffroy, C., Pinna-Déry, A.-M., Renevier, P. and Riveill, M. 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, 2010. Springer-Verlag.
7. Paternò F., Santoro C., and Spano L. D.. Maria: A universal, declarative, multiple abstraction level language for service-oriented applications in ubiquitous environments. In Computer-Human Interaction (TOCHI), volume 16, Nov 2009.
8. Pinna-Déry A.-M. and Fierstone J. Component model and programming: a first step to manage Human Computer Interaction Adaptation. In Mobile HCI'03, volume LNCS 2795, pages 456–460, Udine, Italy, Sept. 2003. L. Chittaro (Ed.), Springer Verlag.
9. Tsai W.-T., Huang Q., Elston J., and Chen Y. Service-oriented user interface modeling and composition. In ICEBE '08, pages 21–28, Washington, DC, USA, 2008. IEEE Computer Society.
10. Grundy J.C. and Hosking J.G. Developing Adaptable User Interfaces for Component-based Systems. In Interacting with Computers, Volume 14, 2, pages 175-194., March 2002, Elsevier Science Publishers.
11. Fujima, J., Lunzer, A., Hornbæk, K. and Tanaka, Y. Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access, In Proceedings of UIST 2004, pages 175-184, Santa Fe, NM, 2004.
12. Elkoutbi, M.., Khriss, I., and Keller, R.K. Generating User Interface Prototypes from Scenarios, In RE'99, Fourth IEEE International Symposium on Requirements Engineering, pages 150-158, Limerick, Ireland, June 1999.
13. Ginzburg, J., Rossi, G., Urbieta, M. and Distante D. Transparent interface composition in Web Applications, In proceedings of Web Engineering, Volume 4607, pages 152-166, Heidelberg, 2007, LNCS , Springer