UI Modeling as Ontology for Composition

Christian Brel, Philippe Renevier-Gonin, Anne-Marie Déry-Pinna, Michel Riveill

UI Modeling as Ontology for Composition

Christian Brel, Philippe Renevier-Gonin, Anne-Marie Pinna-Déry, Michel Riveill

Laboratoire I3S - UMR7271 - UNS CNRS, 2000, route des Lucioles - Les Algorithmes
BP 121 - 06903 Sophia Antipolis Cedex – France
{christian.brel,philippe.renevier, anne-marie.pinna, michel.riveill}@unice.fr

Abstract

The number of specialized applications, like (web) services or smartphone apps, is quickly increasing. Composing such applications is a need for developers in order to quickly produce new applications according to end-users' requirements and customs. In order to support developers, we propose a composition approach using semantics description for component-based applications. We propose to use some roles as cornerstones for the composition by substitution.

1 INTRODUCTION

With the increasing number of applications for smart phone, end-user could use the same functionality in several situations. For example, Google Maps is often integrated for geo-localization.

In an idealistic way, developers must be able to reuse functionalities without (or with minor) developments. The life cycle of an application must not be closed when former developers finish them. But the life cycle must be still open in order to reuse or to rearrange functionalities. Consequently we propose a new UI composing approach, based on semantics description of elements' roles. So we present a model enabling substitution of elements of the UI coming from two former applications, according to their known roles: Input (elements that produce values), Output (elements that consume values) and Trigger (elements triggering (re)actions of the applications).

The paper presents our results starting by positioning our work with a description of related work. Then, the model an application has to respect in order to be composed is described. In Section 4, the composition by substitution is defined. Section 5 details the substitution of a UI element by another, whereas the section 6 illustrates the substitution of several UI elements by the kept UI element. The last sections are the description of an implementation of our model, a discussion and a conclusion.

2 RELATED WORK

The described problem leads naturally to a state of the art around UI composition. We identify the two different approaches:

- The UI composition may be based on abstract description, like in UsiXML [9, 11], in the ServFace project [5, 12], Alias [14] and in Transparent Interface [7]. Those models are defined by XML languages. Final UI are obtained thanks to transformations of those models.

- The UI composition may be based on "UI Components". These components are reusable high-level widgets, available in repositories. "UI components" are reused by applying design pattern (code level) and detecting pattern of use (UI level). Compose [6], COTS-UI [4], CRUISe [13], WinCuts [16], Composable UIs [10], UI façades [15] and on-the-fly mashup composition [17] illustrate such kind of UI composition.

From the analysis of these works, none of these approaches allows both (i) reusing of former applications while replacing UI parts and (ii) building a runnable application based on elements of those former applications. Our proposition is a composition model based on the so-called roles of former elements of the UI, preserving functional links and former elements. The roles of former UI elements are expressed in abstract representations of UI, thanks to ontology. The composition will be performed by transforming the manipulation on the ontology to manipulation of components links. The next section describes our model of composed applications.

3 APPLICATION MODEL BASED ON ROLES

Our goal is to compose applications and in particular their UI, not only by juxtaposition but also by substitutions between former elements of the UI. To

obtain a functional application, we also want to preserve former functional links.

In order to be composition compliant, the existing applications in our composition must follow the separation of concern principle. Indeed, we need to have a clear separation between the functionalities (business part) and the UI. An existing pattern to set up this separation is the Model-View-Controller pattern [8].

Following such a separation of concern, our application model is composed by a View that we considered as a set of Elements of the UI, associated to Elements of Controllers themselves linked to Elements of Functionalities (Model). Each element of the UI is annotated with "UI roles"; we define three conceptual roles to represent the behavior of an element of the UI: Trigger, Input and Output.

We use ontology to model applications to be composed. We define a class Role and its subclasses "UI Role". In additional of UI roles, we define two other roles: Controller (corresponding to the C of M-V-C) and Functionality (corresponding to the M of M-V-C). The **Fig. 1** corresponds to an instance of our ontology about roles. In this figure, each role is represented separately and attached to its element. The same element may appear several times (one by played role). All Elements are represented in another ontology based on an abstract description of the architecture (e.g. a component-based decomposition of the application). Such abstract Element descriptions have a property referencing the concrete Element (the component in the application). In a role, there is also a "method" property indicating which methods are called in the concrete element when it is used for that role.

For each Element, we can associate two sets of roles: the "intrinsic roles" and the "used roles". The "intrinsic roles" of an Element are roles it can intrinsically play. The "used roles" of an Element are roles it effectively plays in an application. Those sets are represented as properties of Elements. The arrows represent the "calling" properties.

In **Fig. 1,** only the "used roles" are represented. The pairs (Element, "UI role") appear in boxes with a grey background. Only one "Controller" is represented, but a pair (Element, "UI role") may be connected to several Controllers. Finally, an Element is not identified on that general representation because an Element may appear several times in different pairs (with different roles). For example a "Text Input" Element can have three "intrinsic roles" – Input (with a "getValue" method), a first Trigger (corresponding to "key typed" events) and a second Trigger (corresponding to "focus" events).

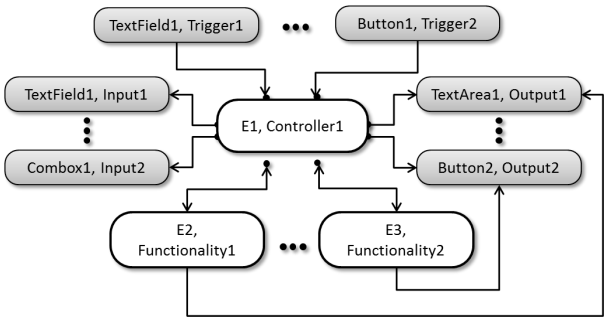

Fig. 1. Application decomposition with roles, a controller-centered view

Based on application decomposition, we describe in the next section how we can compose different UI.

4 COMPOSITION BY SUBSTITUTION

According to the application model presented in the previous section, we can consider the composition of applications:

- $appli_{\mathrm{i}} = EP_{\mathrm{i}} \cup CP_{\mathrm{i}} \cup FP_{\mathrm{i}}$ *where*
- $EP_{\mathrm{i}} = \{E_{\mathrm{i}1} \ldots E_{in}\}$ is a set of Elements belonging to $appli_i$ used for their role of Input, Trigger or Output
- $CP_{\mathrm{i}} = \{C_{\mathrm{i}1} \ldots C_{im}\}$ is a set of Elements belonging to $appli_i$ used as Controller
- $FP_{\mathrm{i}} = \{F_{\mathrm{i}1} \ldots F_{\mathrm{i}p}\}$ is a set of Elements belonging to $appli_i$ used as Functionality

- $R_{intrinsic}\left(E_{i_j}\right) = \{R_1..R_n\}$ is the set of the n intrinsic "UI roles" associated to E_{i_j}, $\forall p, q \in \{1..n\}, p \neq q \Rightarrow R_p \neq R_q$
- $R_{used}\left(E_{i_j}\right) = \{R_1..R_m\}$ is the set of used "UI roles" of E_{i_j} in $appli_i$, $\forall p \in \{1..m\}, R_p \in R_{intrinsic}(E_{i_j})$.

The composition of former UI is made through the composition of a selection of pairs built with an Element and one of its "UI roles". The objective is to substitute such pairs, with the following constrain: for the substituted pair, the "UI role" must be a used one.

Considering ELEMS the set of Elements from n former applications $\{appli_i\}$: $ELEMS = \bigcup_{i=1}^{n} EP_i$. A substitution between several pairs $(E_{i_j}, R_{k_j})_j$ implies to choose a pair in order to conserve it and substitute all other pairs by the conserved one. Considering each UI Element E have a set of "intrinsic roles" $R_{intrinsic}(E)$ and a set of "used roles" $R_{used}(E)$, we define:

- $sel = \left\{(E_{i_j}, R_{k_j})_j , j \in \{1 \dots q\}\right\}$, sel is the set of the q pairs to compose,
- $(E_{i_C}, R_{k_C})_C$, C ∈ {1...q}, is the pair to conserve,
- $\forall j \in \{1 \dots q\}$ and j $\neq$ c $\Longrightarrow R_{k_j} \in R_{used}(E_{i_j})$
- PAIRS = { $ELEMS \times$ { Input, Output, Trigger} }
- subst : PAIRS^n × PAIRS → PAIRS^n is the substitution function: $\text{subst}(sel\ ,\ (E_{i_C}, R_{k_C})_C) = \left\{(E_{i_C}, R_{k_j})_j , j \in \{1 \dots q\}\right\}$

So to substitute a pair $(E_{i_j}, R_{k_j})_j$ by the conserved one $(E_{i_C}, R_{k_C})_C$, the E_{i_C} needs to be compatible with R_{k_j}. The "subst" function changes the ontology by modifying the "calling" property of impacted Elements ($\{E_{i_j}\}$ or Elements "calling" an E_{i_j}). The identification of compatibility between two pairs will be studied in next section.

5 SUBSTITUTING TWO PAIRS

In the next section we will consider the substitution between several pairs. But we first consider substitution between two pairs: (Element, a used "UI role") is the replaced one; (Element, "UI role") is the kept one. That "UI role" may be intrinsic or used. We present the compatibility between the two pairs according to the kept "UI role".

5.1 Keeping an Output

When keeping an output, there is no constraining on the substituted role. By placing an adapter before the Element playing the "output" role, the substitution can be performed. The adapter has to:

- Adapt the format of data to display if the substituted role is also "output" (see **Fig. 2**) or to define a policy of displaying data if the substituted role is also "output" (see **Fig. 2**). Such policy may be displaying all data, the last received data, etc.
- Store displayed data and can restitute them when asked if the substituted role is an "input".
- Generate an event when the output is updated if the substituted role is a "Trigger".

In **Fig. 2**, the adapter A can store displayed data in E_3 and can restitute them to C_2 when asked. With that solution, E_3 doesn't need to play a role of Input, but the Adapter is both an output for the pairs (E4, C1) and (E5, F1) and an input for the pair (E8, C2).

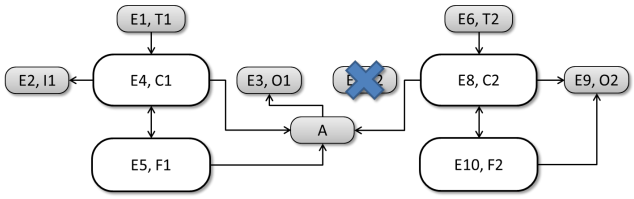

Fig. 2 (E3, Output) replacing (E7, Input), adapter before (E3, Output)

In such case, if we don't use the adapter like in **Fig. 2**, the adapter is simpler and the connections between elements are less impacted. In the second solution (see **Fig. 3**), E_3 also has to be an Input to give data to (E8, C_2).

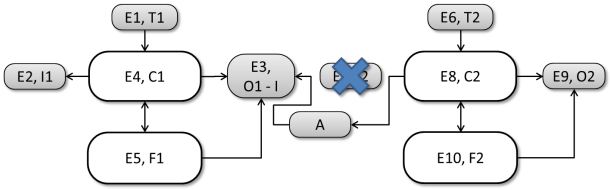

Fig. 3. (E3, Output – Input) replacing (E7, Input) , adapter between (E3, Output – Input) and (E8, Controller)

5.2 Keeping an Trigger

As "Trigger" is the only one "UI role" that makes the associated Element a "caller", the role of substituted pair must be also a "Trigger". Like illustrated in **Fig. 4**, we place an Adapter after the kept "Trigger" for two reasons: (i) adapting the format of the "event" and (ii) defining the policy of the substitution. The adapter can proceed a sequence between the two triggered actions or put them in parallel etc…

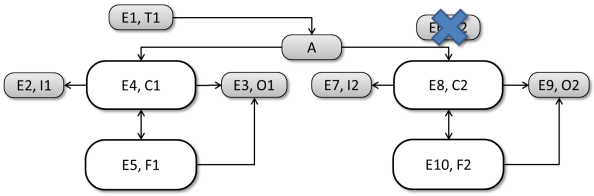

Fig.4. Substituting a "Trigger" by another

5.3 Keeping an Input

An "Input" can not replace an "Output" because of the direction of the data. Both are called by a "Controller", but an Output receives data, while "Input" provides data. Inversely, an "Input" may replace a "Trigger" (see **Fig. 5**). The adapter placed before the kept pair (E2, I1) can provide data to the Controller (E4, C1) by delegating to (E2, I1). In the same time, when called, the adapter can generate an event and so call (E8, C2). The "Trigger" is "on access" (i.e. when the value is got).

Of course, an "Input" can replace another "Input" (see **Fig. 6**). In that case, the adapter is used to adapting the provided data to what is expected by the Controller (E8, C2).

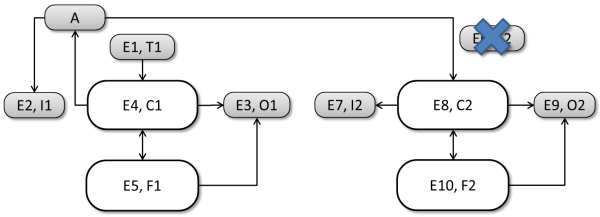

Fig.5. Substituting a "Trigger" by an "Input"

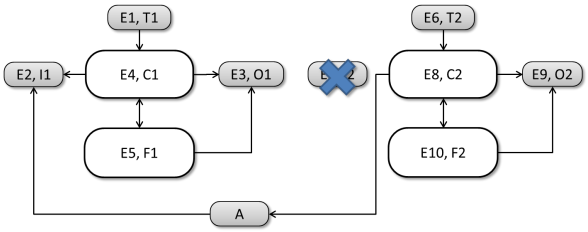

Fig.6. Substituting a "Input" by another

5.4 Summary of substituting two pairs

In order to perform a substitution between two pairs (Element, "UI role"), we need adding an Adapter between the substituted pair and the conserved one. Adapters may have several uses: (i) adapting formats of the data or (ii) defining a policy of substitution or (iii) adding a role when the new role makes the Element the "caller".Thanks to the identification of the Adapter and its roles, we can now define the "subst" function for two pairs. Indeed, in subsections 5.1, 5.2 and 5.3, we define both the definition domain for two pairs (compatibility) and the result. In the next section, we generalize that function to several pairs.

6 SUBSTITUTIONS BETWEEN SEVERAL UI ELEMENTS

6.1 Heuristic

We consider substitutions of several pairs (Element, "UI role") like successive substitutions between pairs. Making a substitution between *p* (Element, "UI role") is like making (*p*-1) substitutions between two pairs.

We propose the following heuristic: a composed UI is obtained by keeping the conserved pair and by operating (*p-1*) substitutions between each removed pair and the kept one.

This is not the only way for building the composed application. Probably we can optimize the number of step by parallelizing substitution. But with our heuristic, we minimize the hypothesis on the roles played by the composed elements. Indeed, only the kept element has to have the "same roles" than the removed pairs, there is no additional constrain between merged elements.

6.2 Prototype

We use the Corese [3] semantic web engine to process and query the RDF. For the concrete Elements, we require Component-based applications to compose. We choose the Julia implementation of the Fractal model [2]. The ontologies describing the applications to be composed must be provided with their application.

We develop a composition process made of two main steps: the Selection Step [1] and the Substitution Step. The Selection step provides complete pieces of application to the Substitution step. During the Substitution Step, as explained in previous sections, we rely on the roles of the different components. When selecting components, we have to choose the replaced used "UI roles". By selecting the kept component, we have to choose one of its "UI roles" (intrinsic or played).

Exploiting the ontology and the description of Fractal component (Provided or Required software interfaces), we generate a Fractal component skeleton of Adapter. The developers have to complete the generated adapter with the expected behavior to obtain the final application (see the subsection 5.4).

7 CONCLUSION

In this paper we present a new UI composing approach, based on description of roles of elements in the views of applications. Our model enables substitution of elements coming from former applications, according to their known "UI roles" (Input, Output, Trigger). By inserting adapters between elements, we could substitute one UI element by a compatible one. We also propose a solution for substitutions involving several elements, as we decompose such complex substitutions in successive substitutions between the kept element and one of the other elements.

We are now working on a software for composition integrating our composition by substitution. The composition software integrates the substitutions and other operations like deleting an element. Our aim is to develop a complete framework of UI compositions preserving functional links.

8 References

1. Brel C., Pinna-Déry A.-M., Renevier P., Riveill M. OntoCompo: A Tool To Enhance Application Composition in Proceedings of the 13th IFIP TC13 Conference in Human-Computer Interaction INTERACT 2011 (Interact 2011), sep 2011
2. 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

3. 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.
4. Criado, J., Padilla, N., Iribarne, L., Asensio, J.: User Interface Composition with COTS-UI and Trading Approaches: Application for Web-Based Environmental Information Systems. CCIS 111, pp. 259-266, Springer-Verlag, Berlin (2010)
5. Feldmann M., Janeiro J., Nestler T., Hübsch G., Jugel U., Preussner A., Schill A. An Integrated Approach for Creating Service-Based Interactive Applications. In: Proceedings of the Conference in Human Computer Interaction (Interact), 2009
6. Gabillon, Y., Petit, M., Calvary, G. and Fiorino, H. Automated planning for userinterface composition. In Proc. of the 2nd Int. Wksp. on Semantic Models for Adaptive InteractiveSystems: SEMAIS'11, (Palo Alto, CA, USA, Feb. 2011), Springer HCI series, 5 pages.
7. Ginzburg, J., Rossi, G., Urbieta, M., Distante, D.: Transparent Interface Composition in Web Applications. In Proceedings of the 7th International Conference on Web Engineering (ICWE2007: July 16-20, 2007; Como, Italy), pp. 152-166 (2007).
8. Goldberg, A., Smalltalk-80: The Interactive Programming Environment, Addison-Wesley Publ., 1984.
9. Lepreux S., Vanderdonckt J., Kolski C. User Interface Composition with UsiXML. Faure D., Vanderdonckt J. (Ed.), Proc. of 1st Int. Workshop on User Interface Extensible Markup Language UsiXML'2010 (Berlin, 20 June 2010), Thales Research and Technology France, Paris, pp. 141-151, juin (2010)
10. Leventhal, E., Grubis, A.: Composable User Interfaces. The MITRE Corporation, Bedford USA (2004)
11. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V.: USIXML: a Language Supporting Multi-Path Development of User Interfaces. In Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425.Springer-Verlag, Berlin (2005) 200–220
12. Nestler T., Feldmann M., Preußner A., and Schill A., Service Composition at the Presentation Layer using Web Service Annotations, in Proc. of the 1st Intl. Workshop on Lightweight Integration on the Web (ComposableWeb'09), June 2009.
13. Pietschmann S., Voigt M., Rümpel A., Meissner K. CRUISe: Composition of Rich User Interface Services. ICWE'09, pp. 473-476.
14. Pinna-Déry A.-M., Joffroy C., Renevier P., Riveill M., and Vergoni C., ALIAS: A Set of Abstract Languages for User Interface Assembly. Proceedings Software Engineering and Applications (SEA 2008). November 16 – 18, 2008. Orlando, Florida, USA
15. Stuerzlinger, W., O. Chapuis, D. Phillips, and N. Roussel, User Interface Façades: Towards Fully Adaptable User Interfaces. In UIST 2006: ACM Symposium on User Interface Software and Technology. 2006.
16. Tan, D.S., Meyers, B., Czerwinski, M.: WinCuts: Manipulating Arbitrary Window Regions for more Effective Use of Screen Space. In Proc. of ACM Conf. on Human Aspects in Computing Systems CHI'2004 (Vienna, April 2004). ACM Press, New York (2004) 1525-1528
17. Zhao Q., Huang G., Huang J., Liu X., Mei H., Li Y., Chen Y. A Web-Based Mashup Environment for On-the-Fly Service Composition. SOSE 2008, pp. 32-37