Application and UI Composition Using a Component-Based Description and Annotations

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

▶ **To cite this version:**

Christian Brel, Philippe Renevier-Gonin, Anne-Marie Déry-Pinna, Michel Riveill. Application and UI Composition Using a Component-Based Description and Annotations. 38th Euromicro Conference on Software Engineering and Advanced Applications, Sep 2012, Izmir, Turkey. pp.204-207, 10.1109/SEAA.2012.44 . hal-01302040

HAL Id: hal-01302040
https://hal.science/hal-01302040

Submitted on 13 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Application and UI composition using a Component-Based Description and Annotations

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*— A possible way to obtain easily new applications is to compose existing applications. In order to support developers in this way, we propose a composition approach manipulating functionalities but also the User Interfaces. We propose a model of applications inspired from Component-Based approaches, describing ports for all Elements of the applications to be composed. We define a substitution between Elements based on those ports.**

Keywords- composition; ontology; component-based architecture;

I. INTRODUCTION

With the increasing number of specialized applications, the need for developers to produce new applications grows up. End-user can 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. To combine features from several applications can be done with composition. To support developers in their task of composition, the Component-Based Software Development (CBSD) is a solution to reuse units of application (components) and reduce production cost. However, this composition takes into account business part of an application and commonly doesn't concern the User Interface (UI) of the application. We propose to go further into reduce of developers' efforts to combine existing application. We propose an application composition through its UI. Considering a UI as an assembly of Elements (components in CBSD approach), we explore the composition via the ports of a component in CBSD. Using the fact that a port can be provided by an Element or required by it, we add some information about the role the port plays for its attached unit. The added information lets to combine the different Elements to obtain a running application.

The paper presents a description of related work, the model that an application has to respect in order to be composed is described, the composition by substitution, the substitution between several ports.

II. RELATED WORK

The described problem leads naturally to a state of the art around software composition and UI composition. For UI composition, we identify two different approaches. In the first approach, the UI composition is based on abstract description, like in UsiXML [4], in the ServFace project [7], Alias [9] and in Transparent Interface [3]. Those models are defined by XML languages. Final UI are obtained thanks to transformations of those models. In the second approach, the UI composition is based on "UI Components". These ones 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 [2], COTS-UI [1], CRUISe [8], WinCuts [11], Composable UIs [5], UI façades [10] and on-the-fly mashup composition [12] illustrate such kind of UI composition.

From the analysis of these works, none of these approaches allows both (i) the reusing of former applications with supporting replacement of UI parts and (ii) the built of a runnable application (UI and business parts) based on elements of those former applications. Our goal is to compose applications and in particular their UI, not only by juxtapositions but also by substitutions between former elements of the UI. To obtain a functional application, we also want to preserve former functional links between Elements (unit of application), in particular between Elements of the UI and Elements of business part.

Our proposition is a composition model based on roles of ports of former Elements of the applications. The roles are expressed as annotations on an abstract representation of application. That representation is "CBSD-like", i.e. we represent Elements as components with ports. Elements of the UI are also represented as component. The composition will be performed by transforming the manipulation on the abstract representation to manipulation on elements. The next section describes our model of application.

III. APPLICATION MODEL BASED ON PORTS AND ROLES

In order to be compliant with our composition method, the existing applications must follow the separation of concern principle: a clear separation between the functionalities (business part) and the UI. An Element, a unit of the application, may belong to the two parts, but the way it is used must be explicit. So each Element is described with its ports that we can tag with an application concern. If a port of an element may be used for a UI concern, the port will be tagged as "UI". Each Element may have required ports (ports required to obtain wished behavior of other Element) and may have provided ports (ports that Element can provide to other Elements to be linked with them). Moreover each port of an Element must be annotated with a "role" representing the involved behavior of the Element. This role can be

Trigger, Input or Output. **Trigger** describes the fact that through its attached port, the Element can call other Element. It can be the button to trigger a particular action or it can be an observable "Element" notifying its observers. **Input** is used to describe a port to get some data. The Element with an Input port can provide data to other Elements (like an "input text" in UI or any "Getter" facet of an Element). **Output** is used to describe a port to set some data. The Element can receive data to store or to display (like a "list" or a "label" in UI or any "Setter" facet of an Element). An application to compose must be provided with the annotations of ports of its Elements. Those annotations are about roles (trigger, input, output / provided or required) or kind ("UI" or not, i.e. Business). For the remain of the article, we use "required-trigger" (rt), "provided-trigger" (pt), "required-input" (ri), "provided-input" (pi), "required-output" (ro) or "provided-output" (po) to refer to a port.

IV. COMPOSITION BY SUBSTITUTION

We can now define the composition of applications: $app_i = \{E_n\}$ where $\{E_n\}$ is the set of Elements from application app_i. We define "Ports", the set of ports of an element, and "UsedPorts", the set of used ports:

$\forall Ej \in appi$, $Ports(Ej) = \{ Pn \}$ is the set of the n ports associated to Ej

$UsedPorts(E_j, app_i) = \{ P_k \}$ is the set of used ports of E_j in app_i

We define the role of the ports and their compatibility:

$\forall E_j \in app_i, \forall P_m \in Ports(E_j)$, $Role(P_m) \in \{pi, po, pt, ri, ro, rt\}$

$\forall Ej \in applii1, \forall Ek \in applii2, \forall Pm \in Ports(Ej), \forall Pn \in Ports(Ek)$, we note $rm = Role(Pm)$ and $rn = Role(Pn)$.

$isProvided(P) = true \Leftrightarrow Role(P) \in \{pi, po, pt\}$

$isRequired(P) = true \Leftrightarrow Role(P) \in \{ri, ro, rt\}$

$Compatible(P_m, P_n) = true \Leftrightarrow (r_m = ro$ and $r_n = po)$ or $(r_n = ro$ and $r_m = po)$ or $(r_m = ri$ and $r_n = pi)$ or $(r_n = ri$ and $r_m = pi)$ or $(r_m = rt$ and $r_n = pt)$ or $(r_n = rt$ and $r_m = pt)$

We define a link between two elements through two connected ports as a function: $Link((E_j, P_m), (E_k, P_n), app_i)$ is true is E_j and E_k are linked in a app_i. Such link is possible only if $E_j \in app_i$, $E_k \in app_i$, $P_m \in UsedPorts(E_j, app_i)$, $P_n \in UsedPorts(E_k, app_i)$ and $Compatible(P_m, P_n)$. For each Element E_j, we define the set $Links(E_j, P_m, app_i)$:

$Links(E_j, P_m, app_i) = \{ (E_k, P_n), E_k \in app_i, P_n \in UsedPorts(E_k, app_i) / Link((E_j, P_m), (E_k, P_n), app_i) \}$

For all ports, we define a function "*isUIPort*" indicating if the port has a "UI" concern and a function "*isUIPortInApp*" for contextual "UI" concern:

$\forall E_j, \forall P_m \in Ports(E_j)$, $isUIPort(P_m) = true$ if $isProvided(P_m)$ and P_m is annotated "UI"

$\forall E_j \in app_i, \forall P_m \in Ports(E_j), isUIPortInApp(P_m, app_i)$ is true if $(P_m \in UsedPorts(E_j, app_i)$ and $isUIPort(P_m))$ or $(isRequired(P_m)$ and $\exists (E_k, P_n) / isUIPort(P_n)$ and $Link((E_j, P_m), (E_k, P_n), app_i))$

Our composition is made through the construction of a new application, app_r:

$app_r = \cup_1^{nb} app_i$ where *nb* is the number of applications being composed. $\forall E_j \in app_i$, when initializing the new app_r :

$E_j \in app_r$

$UsedPorts(E_j, app_r) = UsedPorts(E_j, app_i)$

$\forall E_k \in app_i$, $Link((E_j, P_m), (E_k, P_n), app_r) = Link((E_j, P_m), (E_k, P_n), app_i)$

$\forall P_m \in Ports(E_j)$, $isUIPortInApp(P_m, app_r) = isUIPortInApp(P_m, app_i)$

The new application app_r will change with the successive substitutions. A substitution is made between a selection of pairs $\{(E_j, P_m)\}$ and a kept pair (E_k, P_k). We define the "*subst"* function as following:

We note $PreLinks_k$, $Links(Ek, Pk, appr)$ before the substitution.

We note $card(PreLinks_k)$ the number of Elements in $PreLinks_k$ i.e. the number of Elements linked with E_k though P_k.

$\forall j$, We note $PreLinks_j = Links(E_j, P_m, app_r)$ before the substitution.

$\forall j$, we note $card(PreLinks_j)$ the number of Elements in $PreLinks_j$ i.e. the number of Elements linked with E_j though P_m.

We note $sel = (\{(E_j, P_m)_j, j \in \{1 \cdots z\}\}$ the set of the substituted pairs.

$subst_z$:

$PAIRS^z \times PAIRS \rightarrow (PAIRSxPAIRSxPAIRS^{nk(j)})^q$

$subst(sel, (E_k, P_k)) = \{ (Ec_{j-i}, Pm_{j-i}) \times (Ec_{j-i}, Pn_{j-i}) \times \cup (Ec_{j-i}, Pk_x), x \in \{1 \cdots nk(j)\}, i \in \{1 \cdots card(PreLinks_j)\}, j \in \{1 \cdots z\} \}$ where :

Before the substitution, $\forall (E_j, P_m)_j$, $Pm \in UsedPorts(E_j, app_r)$:

$\forall (E_j, P_m)_j$, $isProvided(P_m) = isProvided(P_k)$

$q = \Sigma_j card(PreLinks_j)$, i.e. q is the number of replaced links

$\forall j$ $nk(j) = 0$ if the connector Ec_{j-i} doesn't impact previous link with (E_k, P_k), i.e. the new link and previous links are independent.

$\forall j$ $nk(j) = card(PreLinks_k)$ if the connector Ec_{j-i} impacts previous link with (E_k, P_k), i.e. the new link and previous links are dependent (merged).

After the substitution,

$\forall j$, $Pm \notin UsedPorts(E_j, app_r)$

$\forall j$, $Links(E_j, P_m, app_i) = \emptyset$

$\forall j$, $\forall i \in \{1 \cdots card(PreLinks_j)\}$, Ec_{j-i} is a new Element of app_r / $\{Pm_{j-i}, Pn_{j-i}\} \subset Ports(Ec_{j-i})$ and $\{Pm_{j-i}, Pn_{j-i}\} \subset sedPorts(Ec_{j-}$

$_i$,app$_r$) and Compatible(P_m,Pm_{j-i}) and Compatible(Pn_{j-i}, P_k)

$\forall Ec_{j-i}$, (Ec_{j-i}, Pn_{j-i}) $\in$ Links(E_k, P_k, app$_i$)

$\forall$(E,P)$\in$PreLinks$_j$, $\exists$(Ec_{j-i},Pm_{j-i}) / (Ec_{j-i}, Pm_{j-i}) $\in$ Links(E, P, app$_i$)

$\forall$(Ec_{j-i},Pm_{j-i}), $\exists$(E,P) $\in$PreLinks$_j$ / (E, P) $\in$ Links(Ec_{j-i}, Pm_{j-i} app$_r$)

$\forall$j, nk(j)>0 => $\forall$x $\in${1..Card(PreLinks$_k$)}, Ec_{j-i} is a new Element of app$_r$ / {Pk_x} $\subset$ Ports(Ec_{j-i}) and {Pk_x} $\subset$ UsedPorts(Ec_{j-i},app$_r$) and $\exists$(E,P) $\in$ PreLinks$_k$ / (Ec_{j-i}, Pk_x) $\in$ Links(E, P, app$_r$)

In other words, the substitution creates q connectors [6] in order to replace previous links involving substituted pairs by the kept one.

if (nk(j) > 0 $\forall$j) then PreLinks$_k$ $\cap$ Links(E_k, P_k, app$_r$) = $\varnothing$, i.e. all connectors also replace the previous links involving the kept pair (E_k, P_k) like in Figure 1.

So for each Element in *sel*, *UsedPorts*, *Link* and *isUIPortInApp* may be impacted by substitution. Finally, Elements no more involved in links are removed.

The substitution of any pair (E_j, P_m) by a pair (E_k, P_k) is based on the annotations. The role of a port is used to define possible substitutions and the way connectors are used. This is explained in the section V. The use of the kind of ports ("UI" or not) is used as following: if before the substitution isUIPortInApp(P_m,app$_r$) $\neq$ isUIPortInApp(P_k,app$_r$), then the substitution changes the concern implied in the link, i.e. an input field may be replaced by a data coming from a "Business" Element. That is possible, but in such case we could emit a notification of such change.

V. SUBSTITUTING TWO PAIRS

We now consider substitution between two pairs: a replaced pair and a conserved or kept pair. The function "subst" can replace *n* pairs, but it is just n substitution performed in parallel. We present the compatibility between the two pairs according to the role of port of conserved Element in conserved pair.

A. Keeping a Provided Output

When keeping an output, there is no constraining on the role of substituted port. By placing a connector before the Element having port playing the Output role, the substitution can be performed. This is a case in the "subst" function where nk(j) > 0 $\forall$j.

First, the connector may be used to adapt the format of data to display if the substituted role is also Output (a Conversion Connector in [6]) or to define a policy of displaying data if the substituted role is also Output (a mix between a Conversion Connector and a Data Access Connector in [6]). Such policy may be displaying all data, the last received data, etc. Secondly, the connector may also be used to store displayed data and can restitute them when asked if the substituted role is an Input (see Figure 1) (a Data Access Connector in [6]). Thirdly, the connector may also be used to generate an event when the output is updated if the substituted role is a Trigger (an Event Connector in [6]).

In Figure 1, the connector C1 can store displayed data from (E3,ro1) and can restitute them to (E8,ri1) when asked. With that solution, E5 doesn't need to have a port playing a role of Input, but the Connector has both provided port with Output role for (E3,ro1), required port with Output role for (E5,po1) and required port with Input role for (E8,ri1).

B. Keeping a Provided Trigger

As "Trigger" is the only one port's role that makes the associated Element a "caller", the role of the port in substituted pair must be also a "Trigger". we place a connector after the kept "Trigger" for two reasons: (i) adapting the format of the "event" and (ii) defining the policy of the substitution (a mix between an Event Connector and a Procedure Call Connector in [6]). The connector can proceed a sequence between the two triggered actions or put them in parallel etc. This is a case in the "subst" function where nk(j) > 0 $\forall$j.

C. Keeping a Provided Input

An "Input" can't replace an "Output" cause of the direction of the data. Inversely, an "Input" may replace a "Trigger". The connector placed before the kept port can provide on demand (a Data Access Connector in [6]). In the same time, when called, the connector can generate an event and so it can "call" the requiring port (an Event Connector in [6]). The "Trigger" is "on access" (i.e. when the value is got). Of course, an "Input" can replace another "Input". In that case, the connector is used to adapting the provided data to what is expected (a Conversion Connector in [6]). Keeping a "Input" is a case where nk(j) could either be 0 (pi replacing pi) or be greater than 0 (pi replacing a pt).

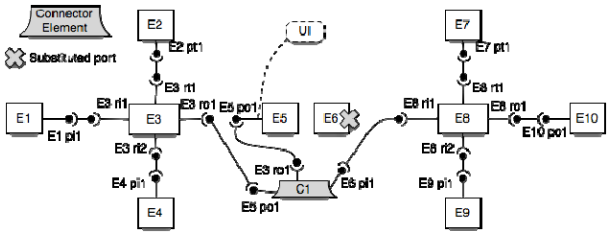

Figure 1 : (E5, po1) replacing (E6, pi1), connector C1 before (E5, po1)

D. Keeping a Required port

In the "*subst*" function, if P_k is a required port, all substituted pairs must be made of ports with the same role as P_k. Even through connector, requirements could not be change: a setter requirement (required output) could not become a getter requirement (required input). So even if a connector could have two ports, one "po" and "ri" to be connected to a "pi", inside that connector, the setter used by E_k could not be functionally translated in a getter. The connector could not appropriately exploit the value coming from E_k. Such substitutions are cases where nk(j) > 0 $\forall$j. All connectors are not only conversion one [6] but they may:

(i)merge all input or trigger if Role(P_k) ∈ {ri, rt} or (ii) call of setter on output if Role(P_k) = ro.

E. Summary of substituting two pairs

In order to perform a substitution between two pairs (Element, Port with a role), we need to add a connector between the substituted pair and the conserved one. Connectors 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 Connector and its roles, we can know define the "subst" function for two pairs. Indeed, in subsections 5.A, 5.B, 5.C and 5.D, we define both the definition domain for two pairs and the results.

F. Towards Automatic Composition

From this substitution operator, we can define an operator in a higher level. The objective is to compose two Elements from the new application app_r. Based on substitutions between ports of Elements, we can define the substitution of two Elements. Let E_l the removed Element and E_k the kept Element. For each P ∈ UsedPorts(E_l, app_r), we define:

CompatiblePorts(P, Ek), the set of all possible port P' of Ek for a substitution subst({(E1, P)}, (Ek, P'))

If isRequired(P) or P = po, CompatiblePorts(P, E_k) = { P' ∈ Ports(E_k) / Role(P') = Role(P) }

If Role(P) = pi, CompatiblePorts(P, E_k) = {P' ∈ Ports(E_k) / Role(P')∈{po, pi} }

If Role(P) = pt, CompatiblePorts(P, E_k) = { P' ∈ Ports(E_k) / isProvided(P') }

We note card(CompatiblePorts(P, E_k)) the number of ports in CompatiblePorts(P, E_k). We apply the "Pair Selection" algorithm PairSelection(P, KeptElements):

Let KeptElements the set of Elements used in the substitution. Initially KeptElements = {E_k}.

Let nb_potential_pairs = Σcard(CompatiblePorts(P, E)), E∈ KeptElements

If (nb_potential_pairs == 1), (E, P) could be substituted by only one pair is possible. Let E' ∈ KeptElements / ∃P'∈ CompatiblePorts(P, E'). The following substitution is computed: subst({(E, P)}, (E', P')}

If (nb_potential_pairs > 1), one of the ports in CompatiblePorts(P) must be selected. That selection may be by the developer operating the composition or by an external algorithm.

If (nb_potential_pairs == 0), (E, P) could not be substituted by a pair involving an Elements form KeptElements.

If KeptElements = app_r, the algorithm finishes without substituting (E, P). Else, we extend the substitution by searching possible ports in Elements linked with Elements from KeptElements:

ExtendedSelection = { Ej ∈ appr / ∃E'∈ KeptElements / ∃ Pm∈ UsedPorts(Ej, appr) and ∃ Pn∈ UsedPorts(E', appr) / Link((Ej, Pm), (E', Pn), appr). Then we apply PairSelection(P, ExtendedSelection)

At the end of the process, if UsedPorts(E_l, app_r) == ∅, E_l is removed from app_r.

VI. CONCLUSION

In this paper we present a new application composition approach from the UI composition. This approach is based on description of roles of ports belonging to Elements constituting the applications. Our model enables substitution of Elements coming from former applications, according to their known ports roles. We also propose a solution for substitutions involving several elements. By tagging the port with their "UI" concern, we take into account the UI part of application in a same level as business part.

REFERENCES

[1] 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)

[2] 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.

[3] 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).

[4] 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)

[5] Leventhal, E., Grubis, A.: Composable User Interfaces. The MITRE Corporation, Bedford USA (2004)

[6] Mehta N. R., Medvidovic N., and Phadke S., Towards a taxonomy of software connectors. Proc. International Conference on Software Engineering, 2000.

[7] 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.

[8] Pietschmann S., Voigt M., Rümpel A., Meissner K. CRUISe: Composition of Rich User Interface Services. ICWE'09, pp. 473-476.

[9] 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

[10] 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.

[11] 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

[12] 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