Application and UI Composition Using a Component-Based Description and Annotations
Christian Brel, Philippe Renevier-Gonin, Anne-Marie Pinna-Déry, Michel Riveill

To cite this version:


HAL Id: hal-01302040
https://hal.archives-ouvertes.fr/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 geolocalization. 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 re usable 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 = (Ej) where Ej is the set of Elements from application app. We define “Ports”, the set of ports of an element, and “UsedPorts”, the set of used ports:

∀Ej ∈ appi, Ports(Ej) = { Pn } is the set of the n ports associated to Ej

UsedPorts(Ej, appi) = { Pn } is the set of used ports of Ej in appi

We define the role of the ports and their compatibility:

∀Ej ∈ appi, ∀ Pn ∈ Ports(Ej), Role(Pn) ∈ {pi, po, pt, ri, ro, rt}

isProvided(P) = true ⇔ Role(P) ∈ {pi, po, pt}

isRequired(P) = true ⇔ Role(P) ∈ {ri, ro, rt}

Compatible(Pi, Pj) = true ⇔ (ri = ro and r = po) or (ri = r and r = pi) or (ro = rt and r = pt) or (ro = r and r = pt)

We define a link between two elements through two connected ports as a function: Link((Ei, Pm), (Ek, Pn), appi) is true is Ei and Ek are linked in a appi. Such link is possible only if Ei ∈ appi, Ek ∈ appi, Pm ∈ UsedPorts(Ei, appi), Pn ∈ UsedPorts(Ek, appi) and Compatible(Pm, Pn).

For each Element Ej, we define the set Links(Ej, appi):

Links(Ej, appi) = { (Ej, Pm), Ei ∈ appi, Pm ∈ UsedPorts(Ei, appi) / Link((Ei, Pm), (Ek, Pn), appi)}

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

∀Ej ∈ appi, ∀ Pm ∈ Ports(Ej), isUIPort(Pm) = true if isProvided(Pm) and Pm is annotated “UI”

∀Ej ∈ appi, ∀ Pn ∈ Ports(Ej), isUIPortInApp(Pn, appi) = true if ( Pn ∈ UsedPorts(Ej, appi) and isUIPort(Pn)) or (isRequired(Pn) and ∃(Ei, Pm) / isUIPort(Pm) and Link((Ei, Pm), (Ek, Pn), appi))

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

app = ∪nb appj, where nb is the number of applications being composed. ∀Ej ∈ appj, when initializing the new app, :

Ej ∈ app, UsedPorts(Ej, appj) = UsedPorts(Ej, appi)

∀ Ej ∈ appj, Link((Ej, Pm), (Ej, Pn), appj) = Link((Ej, Pm), (Ej, Pn), appi)

∀ Pn ∈ Ports(Ej), isUIPortInApp(Pn, appj) = isUIPortInApp(Pn, appi)

The new application app, will change with the successive substitutions. A substitution is made between a selection of pairs {(Ej, Pm)} and a kept pair (Ek, Pk). We define the “subst” function as following:

We note PreLinksj, Links(Ek, Pk, appj) before the substitution.

We note card(PreLinksj) the number of Elements in PreLinksj i.e. the number of Elements linked with Ek though Pk.

Vj, We note PreLinksj = Links(Ej, Pn, appj) before the substitution.

Vj, we note card(PreLinksj) the number of Elements in PreLinksj i.e. the number of Elements linked with Ej though Pn.

We note sel = ({{Ej, Pn}, j ∈ {1…z}}) the set of the substituted pairs.

We define a subst function as following:

∀ subst : PAIRS × PAIRS → (PAIRS×PAIRS×PAIRS)q

∀ subst(subj, (Ej, Pj)) = ( { (Ecj-i, Pmj-i) x (Ecj-i, Pmj-i)} x subst(sel, (Ek, Pk, appj)) ) where:

Before the substitution, ∀(Ej, Pj), Pm ∈ UsedPorts(Ej, appi) :

∀ (Ej, Pj), isProvided(Pj) = isProvided(Pj)

∀(Ej, Pj), q = Σ card(PreLinksj), i.e. q is the number of replaced links

Vj nk(j) = 0 if the connector Ecj doesn’t impact previous link with (Ej, Pj), i.e. the new link and previous links are independent.

Vj nk(j) = card(PreLinksj) if the connector Ecj impacts previous link with (Ej, Pj), i.e. the new link and previous links are dependent (merged).

After the substitution, ∀ Ej, Pj ∈ UsedPorts(Ej, appi)

∀ Vj, Links(Ej, Pj, appi) = ∅

∀ Vj, ∀ Pj ∈ Ports(Ej), isUIPortInApp(Pj, appi) = true if (Ej, Pj) is a new Element of appi / {Pmj-i, Pnj-i} ⊆ Ports(Ecj-i) and {Pmj-i, Pnj-i} ⊆ sedPorts(Ecj-i)
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,p01) 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 ∀v.

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).

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 ∀v.

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

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 define the “substitution” 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 appr. Based on substitutions between ports of Elements, we can define the substitution of two Elements. Let \( E_1 \) the replaced Element and \( E_2 \) the kept Element. For each \( P \in \text{UsedPorts}(E_1, \text{appr}) \), we define:

\[
\text{CompatiblePorts}(P, E_k) \text{, the set of all possible port } P' \text{ of } E_k \text{ for a substitution subst} \{(E_1, P), (E_k, P')\}
\]

If \( \text{isRequired}(P) \) or \( P = po \),

\[
\text{CompatiblePorts}(P, E_k) = \{ P' \in \text{Ports}(E_k) / \text{Role}(P') = \text{Role}(P) \}
\]

If \( \text{Role}(P) = pi \),

\[
\text{CompatiblePorts}(P, E_k) = \{ P' \in \text{Ports}(E_k) / \text{Role}(P') = \text{Role}(P) \} \}
\]

If \( \text{Role}(P) = pt \),

\[
\text{CompatiblePorts}(P, E_k) = \{ P' \in \text{Ports}(E_k) / \text{isProvided}(P') \}
\]

We note \( \text{card}(\text{CompatiblePorts}(P, E_k)) \) the number of ports in \( \text{CompatiblePorts}(P, E_k) \). We apply the “Pair Selection” algorithm PairSelection(P, KeptElements):

Let KeptElements the set of Elements used in the substitution. Initially

\[
\text{KeptElements} = \{E_2\}.
\]

Let \( \text{nb\_potential\_pairs} = \sum \text{card}(\text{CompatiblePorts}(P, E)) \), \( E \in \text{KeptElements} \)

If \( \text{nb\_potential\_pairs} = 1 \), \((E, P)\) could be substituted by only one pair is possible. Let \( E' \in \text{KeptElements} / \exists P' \in \text{CompatiblePorts}(P, E') \). The following substitution is computed: \( \text{subst} \{(E, P), (E', P')\} \)

If \( \text{nb\_potential\_pairs} > 1 \), one of the ports in \( \text{CompatiblePorts}(P) \) must be selected. That selection may be by the developer operating the composition or by an external algorithm.

If \( \text{nb\_potential\_pairs} = 0 \), \((E, P)\) could not be substituted by a pair involving an Elements form KeptElements.

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

\[
\text{ExtendedSelection} = \{ E_j \in \text{appr} / \exists E' \in \text{KeptElements} / \exists P \in \text{UsedPorts}(E_j, \text{appr})
\]

and \( \exists P \in \text{UsedPorts}(E', \text{appr}) / \text{Link}(E_j, P_m), (E', P_n), \text{appr}) \). Then we apply PairSelection(P, ExtendedSelection)

At the end of the process, if \( \text{UsedPorts}(E_1, \text{appr}) = \emptyset \), \( E_1 \) is removed from appr.

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


