OntoCompo: An Ontology-based Interactive System to Compose Applications

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

ONTOCOMPO: AN ONTOLOGY-BASED INTERACTIVE SYSTEM TO COMPOSE APPLICATIONS

Christian Brel, Anne-Marie Dery-Pinna, Catherine Faron-Zucker, Philippe Renevier-Gonin, Michel Riveill

I3S Lab, Université Nice-Sophia Antipolis / CNRS, 930 route des Colles, BP 145, 06903 Sophia Antipolis Cedex, FRANCE
{brel,pinna,faron,renevier,riveill}@polyecth.unice.fr

Keywords: UI Composition, Application Composition, UI Semantic Description

Abstract: In this paper, we present an ontology-based approach and a semantic web system to compose applications while preserving their ergonomic properties. Our composition process relies on the manipulation of User Interfaces (UI) and is intended to assist by a knowledge based system which exploits semantic annotations of applications on their users' aims, UIs and functionalities through semantic queries and inference rules.

1 INTRODUCTION

User-Centred Software Engineering aims at producing useful and usable applications. To catch users' needs and to integrate them inside the software development, there are different steps to follow: analyzing requirements, designing User Interfaces (UI), specifying software architecture, performing software tests, testing with final users, etc. This is a long and costly process.

At the same time, there are more and more specialized applications, such as web services or Smartphone applications. Sometimes, users swap from one application to another. In such cases, they memorize and type again data or use copy-past in order to exchange information between applications. To avoid mistakes during "application swapping", composing applications seems to be a solution.

What is at stake here is to compose existing UI and functionalities by preserving results of user-centred methodologies. Composing functionalities is quite a well-known process, but composing at the same time the UI is still an on-going work.

We propose a composition process based on the selection, extraction and positioning of existing application's UI as elementary composition actions to impact underlying users' aims and links to the functionalities (business part). The choice of UI as primary artefacts manipulated by the composition process is justified by the fact that UI are the visual part of an application. They can be directly manipulated with an immediate visual feedback. We aim at enabling the developers to reuse existing UI for creating new applications while preserving final user requirements.

We propose a composition process driven by the developer that enables to avoid redundancies by selecting preferred UI parts, to preserve the initial layouts of these UI parts and to associate them with some new layout knowledge to constraint their composition. The process is based on two iterative steps: the selection of UI parts and the organization of their layouts.

In this paper we focus on the management of selection of the different parts of UI from existing applications and on the management of the layouts chosen by the developer. Our solution relies on ontologies we built to provide a usable description of an application, i.e. the abstraction of usual layouts used in programming languages graphical libraries, the description of tasks being able to perform by users and the description of the different links between UI, functionalities and tasks. We use these ontologies, inference rules and constraints in our UI composition process to assist the developer.

The paper is structured into 5 sections. In section 2, we summarize related works. In section 3, we

describe our three ontologies and links between them. Before the conclusion, in section 4, we present how we use these ontologies to represent the interests of three ontologies, to define inference rules to converge to a pivot representation of relative layouts, and finally to define constraints to control positioning chosen by the developer.

2 RELATED WORK

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) the description of its structure, i.e. the listing of the different components used in the interface and the inclusion relationship, like UIML (Abrams, 1999), ALIAS (Occello, 2010), UsiXML (Limbourg, 2004) or MARIA (Paternò, 2009)

(2) the spatial positioning of these components. By analysing the different layouts used in the UI toolkits, we identified three ways to position the components in an interface: the AbsoluteLayout with X and Y coordinates, the TableLayout to place a component in a grid and the RelativeLayout to express the positioning of two UI components relatively to each other.

There are currently three main approaches to application composition depending on the composition entry point: (i) the functional (i.e. business) part, (ii) the users' goals (i.e. tasks to be performed by users) and (iii) the UI. Each entry point addresses a specific problem of composition: presentation and layout considerations at the UI level, behavior of the application at the functional level (F in Table 1), user needs at the task level (T in Table 1). We group and classify the works related to UI composition in Table 1. We notice a lack in underlying composition processes. Either the original design of application UI with man-crafted properties such as ergonomic or usability is lost, or both functional 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 functional parts engenders human interventions to connect the two parts which is error prone and fastidious for large applications. So in order to obtain a functional application at the end of the composition, we need to keep links between the different levels to guide the developer during the selection and positioning steps.

In next sections, we introduce how we represent an application and how we use this representation to help the developer in her selection of UI pieces and to help in their positioning in the new UI.

Table 1. Classification of composition approaches.

Category		F	UI	T
only considering UI composition	Developing adaptable user interfaces (Grundy, 2002)		**X**	
	Amusing (Pinna-Déry, 2003), ComposiXML (Lepreux, 2007)		**X**	
	C3W (Fujima, 2004)		**X**	
only considering tasks composition	Task Models Merging (Lewandowski, 2007)			**X**
deriving Tasks in functional composition and later in UI composition	Servface (Paternò, 2009)	**X**		**X**
	Compose (Gabillon, 2008)	**X**		**X**
	Scenarios (Elkoutbi, 1999)		**X**	**X**
both functionalities and UI composition	SOAUI (Tsai, 2008), ALIAS (Occello,2010), Transparent Interface (Ginzburg, 2007)	**X**	**X**	

3 APPLICATION REPRESENTATION

To represent an application, we propose a model relying on three ontologies: UIOnto, LayOnto and TaskOnto. UIOnto gathers the concepts necessary to represent knowledge about UI structures, i.e. their components and hierarchical organization. LayOnto gathers those necessary to represent knowledge about the layout of UI components. The third ontology TaskOnto gathers the concepts necessary to represent the task tree describing the available actions in the application and the unfolding between the different tasks.

3.1 UI Representation

Our modeling of UIOnto relies upon the MARIA model. UIOnto is represented in the OWL Lite standard (W3C Working Group, 2004) and comprises 26 classes and 4 properties. Figure 1

presents its main classes and properties. Under OnlyOutput, there are classes like Text, List, Link, etc. Under Interaction, there are classes like Edit (and then TextEdit, NumericalEdit, etc.), Selection, etc.

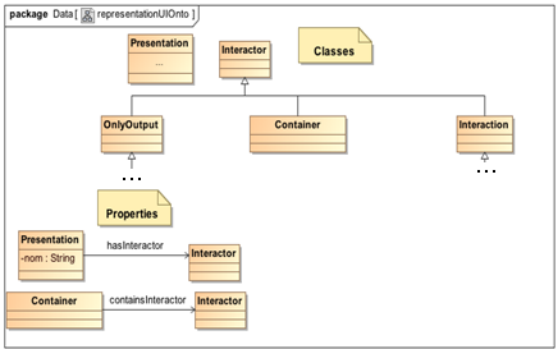

Figure 1. UIOnto

Our aim is to help designers building their new applications keeping constraints of existing applications. What is interesting in layout is the meaning of spatial proximity: two close UI elements may be perceived and analysed together as explained in the ICS model (Barnard, 1991). So keeping such proximity may preserve ergonomics. We let the developer decide whether to keep such proximity. As a result, we chose to express the meaning of UI elements spatial proximity by RelativeLayout, a universal way to express all traditional layouts by highlighting their proximity properties.

3.1.1 An Abstract Layout Description for Composition

This is what has guided our modelling of LayOnto. LayOnto is represented in the OWL Lite standard and comprises 2 classes and 42 properties. Figure 2 presents its main classes and properties. The main property of LayOnto is *isPositionnedRelativelyTo* that represents a relation between two Interactors and describes the position of one of them relatively to the other. Its three subproperties of correspond to the three layouts discussed above:

- *isGridPositionnedIn* corresponds to TableLayout, specialized into subproperties representing the different possible positioning (and combinaison) by considering a 3x3 grid inside the first interactor,
- *isAbsolutePositionnedIn* corresponds to AbsoluteLayout with a Point Properties with X and Y Values,
- *isGridPositionnedRelativelyTo* corresponds to RelativeLayout, specialized into subproperties representing the different possible relative positioning (and combinaison) by considering a 3x3 grid centered around the first interactor.

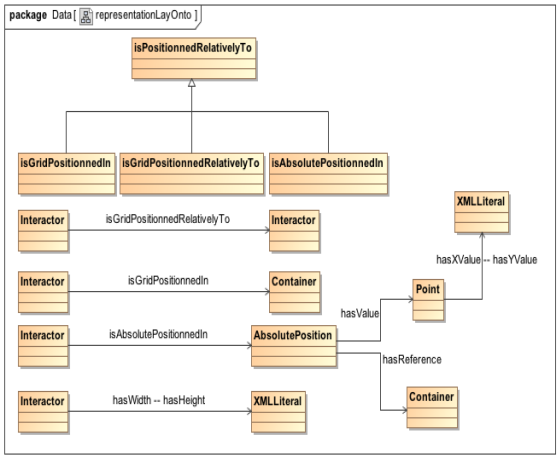

Figure 2. LayOnto

3.1.2 From final UI to Semantic Representation of UI

UIOnto and LayOnto enable to represent the layout of interfaces at an abstract level shared by all the usual layouts in graphical libraries. For instance, all the layout managers described in the Java API (Sun, 2008) can be represented. Let us consider a part of an UI. Its simplified Java code is as follows:

```
1. JPanel insurSearch = new JPanel();
2. insurSearch.setLayout(new
BoxLayout(insurSearch, BoxLayout.X_AXIS));
3. insurSearch.add(new JLabel("Insurance Card
Id:"));
4. JTextField insurSearchInput =
       new JTextField("123456", 20);
5. insuraSearchInput.setMaximumSize(
     insurSearchInput.getPreferredSize());
6. insurSearch.add(insurSearchInput);
7. JButton insurSearchSubmitButton =new
JButton("show insurance information");
8. insurSearch.add( insurSearchSubmitButton );
9. insurSearch.add( Box.createHorizontalGlue() );
```

Figure 3. Java code of a form

In this Java code, variable `insurSearch` represents the container of the whole form. A BoxLayout is set to this container. With this layout, elements are put in a row (line 2). Components added in the container (lines 3-6-8-9) are then aligned in the form. From this Java code, a semantic annotation of the UI can be constructed with UIOnto in RDF model (W3C Working Group, 2004) as follows:

```
<#insurSearch> a :Container.
<#insurSearchLabel> a :Text.
<#insurSearchInput> a :TextEdit.
<#insurSearchSubmitButton> a :Activator.
<#insurSearchGlue> a :Glue.
```

```
<#insurSearch> <#containsInteractor>
<#insurSearchLabel>, <#insurSearchInput>,
<#insurSearchSubmitButton>, <#insurSearchGlue>.
```

It expresses that Container insurSearch contains four interactors: a Text corresponding to the JLabel in the Java code, a TextEdit corresponding to the JTextField, an Activator corresponding to the JButton and a Glue corresponding to the createHorizontalGlue in the Java code. This RDF description can be further enriched with knowledge about layouts with LayOnto concepts:

```
<#insurSearchInput> <#isOnTheRightOf>
<#insurSearchLabel>.

<#insurSearchSubmitButton> <#isOnTheRightOf>
<#insurSearchInput>.

<#insurSearchGlue> <#isOnTheRightOf>
<#insurSearchSubmitButton>.
```

The association of a BoxLayout with X_AXIS (respectively Y_AXIS) attribute to container `insurSearch` is represented by instances of a subproperty of *GridPositionnedRelativelyTo* corresponding to BoxLayout in the Java code - *isOnTheRightOf* (respectively *isBelowOf*) - with `insurSearch` as their subject and interactors contained in it as their values. In a similar vein, we can associate each position of the Java BorderLayout with a position of our TableLayout: "West" with *isInLeft*, "North" with *isInAllTop*, "East" with *isInRight*, "South" with *isInAllBottom* and "Center" with *isInCentre*. With the high degree of abstraction of UIOnto and LayOnto, similar translations hold for almost all Java Layout Managers but also for layouts of other interface description languages like XAML.

3.2 Linking UI, Tasks and Functionalities

Our modelling of TaskOnto relies on the ConcurTaskTree (CTT) (Mori, 2002) model. TaskOnto is represented in the OWL Lite standard and comprises 5 classes and 3 properties. In a nutshell, the main class of TaskOnto is *Task* that represents an action possible to perform in the application. There are 4 types of Task, appearing as 4 subclasses of class Task.

- *InteractionTask* describing an action performed through the UI,
- *SystemTask* representing an action performed by the functional part,
- *UserTask* representing an action performed by the final user (without inputs for the application) and
- *AbstractTask* that represents a task composed of subtasks (of all type – InteractionTask, SystemTask or UserTask).

The properties *hasSubtask* and *hasParentTask* enable to describe the tree by dividing the different tasks into an unfolding of tasks. To construct this tree, property *hasTemporalOperator* applies to a task and enable to describe how a subtask is executed, sequentially or competitively, etc…

To obtain a functional application resulting of an application composition, we relate UIOnto and LayOnto to TaskOnto by associating to any task: (i) the functionalities used to perform the corresponding system actions and (ii) the UI parts used to interact with the application during the task.

We define two RDF properties *linkedWithUIEntity* and *linkedWithFunctionality* which apply to a task and relate it to a UI entity in UIOnto and to functionality. By relating the three ontologies UIOnto, LayOnto and TaskOnto, we can entirely describe any application. In next section, we explain how we use all annotations to build the application's UI resulting of the composition.

4 INTERESTS OF UIONTO, LAYONTO, TASKONTO FOR UI COMPOSITION

Once the RDF representations of applications are extracted by analysing selected parts of existing UI, these representations need to be unified for their manipulation by our algorithm for computer-supported composition.

4.1 Deduction of relative layouts

To complete our models UIOnto and LayOnto we have built a base of 14 inference rules enabling to deduce relative layout of UI components from any layout description. 4 rules state that from two positions in the RelativeLayout, we may obtain a third one, e.g. if an interactor S1 is above a S2 and S1 is on the left of S2, then we can deduce that S1 is above left of S2. We formalize it in the SPARQL (W3C Working Group, 2008) language:

```
CONSTRUCT { ?s isAboveLeftOf ?s2 }
WHERE {
   ?s1 isCenteredAboveOf ?s2.
   ?s1 isOnTheLeftOf ?s2
}
```

In a similar vein, 4 rules enable to deduce relative positions from absolute positions and 6 rules enable to deduce relative positions from grid positions. With these rules we can deduce relative positions of any component in the new composed UI. These results are necessary because we use the

RelativeLayout to represent the constraints of the developer expressed during the composition process (positioning of selected parts of former UI).
The developer is helped by these rules in maintaining the consistency of the new UI. For example, it will enable the developer to extend her selection of UI parts to fix the position of a larger and more appropriate or coherent group of already-placed UI parts. It will also enable the developer to perform specific selections like "all interactors in the left of…". Rules are useful for the detection of conflicts, like when the developer positions two different UI parts at the same place, as two Activators sequentially placed on the left of the same Interactor. Where must be placed the second Activator? On the left of the first one? Between the first Activator and the Interactor? etc.

4.2 Consistency of UI Composition

During the composition process, when the developer places selected parts of existing UI, she is helped in these actions to guaranty the consistency of the new interface. This help is done thanks to 3 categories of semantic queries built upon our ontologies. These categories of queries are dedicated to complete the selection of the developer.

4.2.1 Help from layout

The first category of queries uses layout information to help the developer to complete her selection. For example, the query below retrieves all components in the container of the selected component.

```
SELECT ?o WHERE {
 ?container containsInteractor ?selectedComponent.
 ?container containsInteractor ?o
}
```

With this first category, we are able to ask the developer if she wants to select the container and its components, only the selected component, selected component and some other components in the same container etc. With such interaction, we can help her to select difficult parts to point out (like a Container "hidden" by its contained Interactors).

4.2.2 Help from tasks

This second category of queries uses task information to help the developer to complete her selection. There are two steps of queries. The first retrieves the tasks attached to the selected component with the query below:

```
SELECT ?t WHERE { ?t linkedToUIElement
?selectedComponent }
```

After getting the attached tasks, the second step retrieves the parent task of the retrieved tasks and from that parent task all the UI elements attached to its subtasks. Here, the idea of this help is to consider a semantic proximity of the UI elements when they perform a global common task. The query bellow retrieves all UI elements achieving a common task:

```
SELECT ?uielement WHERE {
  ?retrieveTask hasParentTask ?parentTask .
  ?parentTask hasSubtask ?subtask .
  ?subtask linkedToUIElement ?uielement
}
```

Initially, this query retrieves the parent task of the retrieved task (obtained by the first step). Then, it reaches the different subtasks of the parent task and finally UI element attached to these subtasks. All UI elements may be submitted to the developer for validation in order to be added to the selection.

4.2.3 Help from functionalities

This third category of queries uses functionalities information to help the developer to complete her selection. These queries retrieve pieces of UI manipulating the same functionalities. So, this enables to avoid forgetting some UI part potentially doing the same type of task or at least performing a task using the same functionality. We need to retrieve the tasks associated to a selected UI component, then to the parent task of these retrieved tasks. Among these subtasks, we can search a task linked with functionalities:

```
SELECT ?funct WHERE {
  ?retrievedtask linkedToUIElement
?selectedComponent .
  ?retrievedtask hasParentTask ?parentTask .
  ?parentTask hasSubtask ?subtask .
  ?subtask linkedToFunctionality ?funct
}
```

If such functionalities exist, then we can execute the second step of our help i.e. obtain the different tasks linked with the retrieved functionalities and reach the UI elements attached to their parent task:

```
SELECT ?uielement WHERE {
  ?retrievedtask linkedToFunctionality ?funct .
  ?retrievedtask hasParentTask ?parentTask .
  ?parentTask hasSubtask ?subtask .
  ?subtask linkedToUIElement ?uielement
}
```

With these three categories of queries, we are able to help the developer during the selection of the different UI parts she wants to reuse in the new UI.

5 CONCLUSION

With the semantic representation of applications, we propose in this article to automate the composition of applications. The developer is the initiator of the composition of a new interface by selecting the

components she wants to keep. The developer controls consistency of the composition along the selection (she is helped thanks to layout description, task description and functionalities description) but also after this selection, during the composition by expressing constraints about the layout of the new interface. In our approach, the composition is semi-automatic because a feedback is done to the developer by requiring precisions about selections and about the constraints on the new interface.
In this work, there is an enrichment of the semantic descriptions by associating knowledge about functional features to current knowledge about the layout of interface components. The functional descriptions allow us to do the fusion of some interactors. It allows providing feedback to the developer about how to lead the UI composition.

ACKNOWLEDGEMENTS

Our work is funded by the DGE M-Pub 08 2 93 702 project.

REFERENCES

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,.

Barnard P.J., 1991. Teasdale J.D. Interacting cognitive subsystems: A systemic approach to cognitive-affective interaction and change. *Cognition & Emotion*, 5, 1, 1-39

Elkoutbi M., Khriss I., Keller R.K., 1999. Generating User Interface Prototypes from Scenarios, *In RE'99, Fourth IEEE International Symposium on Requirements Engineering*, pages 150-158, Limerick, Ireland.

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.

Gabillon Y., Calvary G., Fiorino H., 2008. Composing interactive systems by planning. *In UbiMob'08*, pages 37–40, Saint Malo, France.

Ginzburg J., Rossi G., Urbieta M., Distante D., 2007. Transparent interface composition in Web Applications, *In proceedings of Web Engineering*, Volume 4607, pages 152-166, Heidelberg, LNCS , Springer.

Grundy J.C., Hosking J.G., 2002. Developing Adaptable User Interfaces for Component-based Systems. *In Interacting with Computers*, Volume 14, 2, pages 175-194., Elsevier Science Publishers.

Lepreux S., Hariri A., Rouillard J., Tabary D., Tarby J.-C., and Kolski C., 2007. Towards multimodal user interfaces composition based on usixml and mbd principles. *Lecture Notes in Computer Science, 4552(134)*:134–143.

Lewandowski A., Lepreux S., Bourguin G., 2007. Tasks models merging for high-level component composition. *Human-Computer Interaction, Part I, HCII 2007*, Lecture Notes in Computer Science (LNCS), 4550:1129–1138.

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.

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.

Occello A., Joffroy C., Pinna-Déry A.-M., Renevier 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.

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.

Pinna-Déry A.-M., Fierstone J., 2003. 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. L. Chittaro (Ed.), Springer Verlag.

Tsai W.-T., Huang Q., Elston J., Chen Y., 2008. Service-oriented user interface modeling and composition. *In ICEBE '08*, pages 21–28, Washington, DC, USA, IEEE Computer Society.

W3C Working Group. OWL Web Ontology Language http://www.w3.org/TR/owl-features/, 2004.

W3C Working Group. Resource Description Framework(RDF). http://www.w3.org/RDF/, 2004.

W3C Working Group. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2008.