User Interface Composition with UsiXML

Sophie Lepreux[1,2,3], **Jean Vanderdonckt**[4], **Christophe Kolski**[1,2,3]

[1]Univ Lille Nord de France, F-59000 Lille,
[2]UVHC, LAMIH, F-59313 Valenciennes,
[3]CNRS, FRE 3304, F-59313 Valenciennes,
France - +33 327 511 465
{sophie.lepreux, christophe.kolski}
@univ-valenciennes.fr

[4]Université catholique de Louvain
Louvain School of Management,
Place des Doyens, 1
B-1348 Louvain-la-Neuve (Belgium)
+ 32(0)10/47.85.25
jean.vanderdonckt@uclouvain.be

ABSTRACT
This paper presents novel ongoing works on user interfaces composition. These works have emerged with the problematic of component software composition transposed to the Human-Computer Interaction domain. Some software architectures indeed allow components assembling at the final design step. Our work, based on UsiXML, aims at proposing a composition/decomposition of user interfaces. These works begin with the concrete level of UsiXML dedicated to the graphical modality and continue with higher abstraction levels. This article provides a positioning of the proposal related to composition compared to the seven dimensions related to the "μ7" concept of UsiXML project.

Author Keywords
Composition, decomposition, interactive component, user interface adaptation, user interface extensible markup language, "μ7" concept.

General Terms
Design, Human Factors, Languages.

ACM Classification Keywords
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *User interfaces*. H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces.*

INTRODUCTION

The issue of software component reuse has given a new issue in Human-Computer Interaction (HCI) on the *User Interface* (UI) *Composition* [2,6,10,11,18]. If we take as hypothesis that a software component can be associated with interactive(s) component, then the composition of user interface components emerges. Many different terms have been used to refer to this problem, such as fusion [2], transparent composition [10], composable UIs [22], component-based adaptation [11], or pattern-based merging [7]. It is also interesting to note that UI compositions has been addressed at different levels of abstraction, ranging from code level [22], components [6,11], concrete UI [20], abstract UI [23], and task [1,3]. Different UI types have also been considered such as: web UI [1], cross-platform applications [2], web services [5,8,9], context-aware systems [13], tangible UIs [16,17], decision-support systems [18], and multimodal UIs [19]. Many different techniques have been used [7,8,32,33].

This article aims at presenting works ongoing on UI composition, in particular with respect to UsiXML [27]. The UsiXML language is XML-compliant language so the UI elements can be structured according to tree structure. The first section of introduction presents the tree algebra which is used to handle the model. The second section aims at giving the major principles of UsiXML which are useful to understand the paper. Then, the first part introduces the composition operators and algorithms which had proposed to compose user interfaces. These operators were initially proposed to be used on graphical user interfaces [20], modeled with UsiXML. Nevertheless, we can see in second part they can be used relatively to higher abstraction levels, i.e. abstract user interface and task level. Algorithms proposed specifically to the concrete graphical user interface are generalizable to the set of models proposed by UsiXML [27], but could be also applied to equivalent User Interface Description Languages (UIDLs) [12]. The last section shows how composition operators support UI adaptation according the 7 dimensions introduced in the UsiXML.

Tree-algebra based UI Composition

Since the UI is represented in UsiXML terms and since it is a XML-compliant language, operations could be defined thanks to tree algebra. The composition operations could be logically defined on the XML tree and directly performed. Jagadish *et al.* define a data model [15]. A data tree is a rooted, ordered tree, such that each node carries data (its label) in the form of a set of attribute-value pairs. Each node has a special, single valued attribute called tag whose value indicates the type of element. A node may have a content attribute representing its atomic value. Each node has a virtual attribute called pedigree drawn from an ordered domain. The pedigree carries the history of "where it came from". Pedigree plays a central role in grouping, sorting and duplicate elimination. They define a pattern tree as a pair $P = (T, F)$, where $T = (V,E)$ is a node-labelled and edge-labelled tree such that:

- Each node in V has a distinct integer as its label ($i).
- Each edge is either labelled pc (for parent-child) or ad (for ancestor-descendant).
- F is a formula, i.e. a Boolean combination of predicates applicable to nodes.

In Faure, D., Vanderdonckt, J., (Eds.), *Proc. of 1st Int. Workshop on User Interface Extensible Markup Language UsiXML'2010* (Berlin, 20 June 2010), Thales Research and Technology France, Paris, 2010. ISBN 978-2-9536757-0-2

This pattern is used to define a database and to define the predicate used in the operations. As the initial notation is specific to the database, we had proposed a variant which is adapted to documents specific to interface. Indeed, in the HCI case, the most important is the structure and not the content. For example, it is more important to know that the window has a box as subitem than that the window has a height equal to 300. So the attributes are stored with the tag. A node is a tag with these attributes and their content. The pattern tree keeps coherent with the variant definition. Another point specific to the database is that the data are in several data trees so the operators use a collection of data trees in input and output. In the HCI case, the input is one (for the unary operators) or two (for the binary operators) XML documents so one or two data trees.

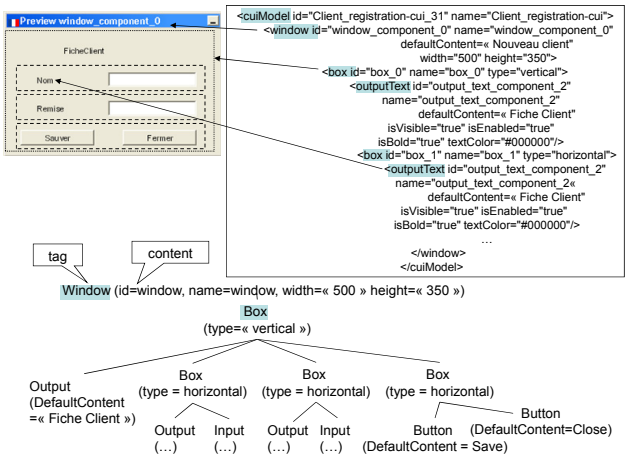


Figure 1. Corresponding between GUI-UsiXML/CUI – tree.

UsiXML framework

UsiXML is structured according to the four abstraction levels of the 'CAMELEON reference framework' [4] for multi-target UIs (Fig. 2). A *Final User Interface* (FUI) refers to an actual UI rendered either by interpretation (e.g., HTML) or by code compilation (e.g., Java). A *Concrete User Interface* (CUI) abstracts a FUI into a description independent of any programming or markup language in terms of Concrete Interaction Objects, layout, navigation [28], and behavior [29,30]. An *Abstract User Interface* (AUI) abstracts a CUI into a definition that is independent of any interaction modality (such as graphical, vocal or tactile). An AUI is populated by *abstract components* and *abstract containers*. Abstract components are composed of facets describing the type of interactive tasks they are able to support (i.e., input, output, control, navigation). The *Tasks & Concepts* level describes the interactive system specifications in terms of the user tasks to be carried out and the domain objects.

COMPOSITION/DECOMPOSITION OPERATORS

Several operators have been proposed [20]. They are distributed in two parts, the unitary operators which act on a single interface and the binary operators which take as arguments two interfaces. A global view of these operators is presented in Fig. 3. The operators are presented in this part as well as several algorithms. They are illustrated with examples in the following part and according "μ7" concept of UsiXML project in the last part.

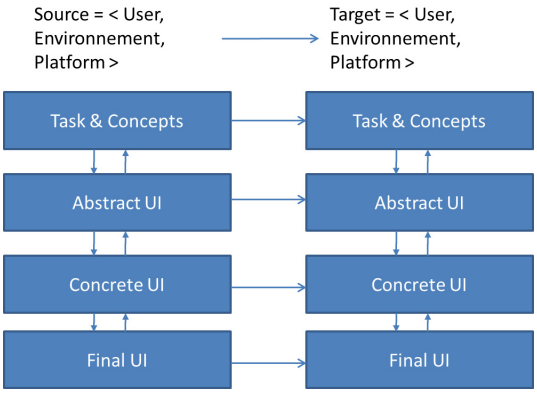


Figure 2. The four abstraction levels used in the Cameleon Reference Framework (CRF) [4].

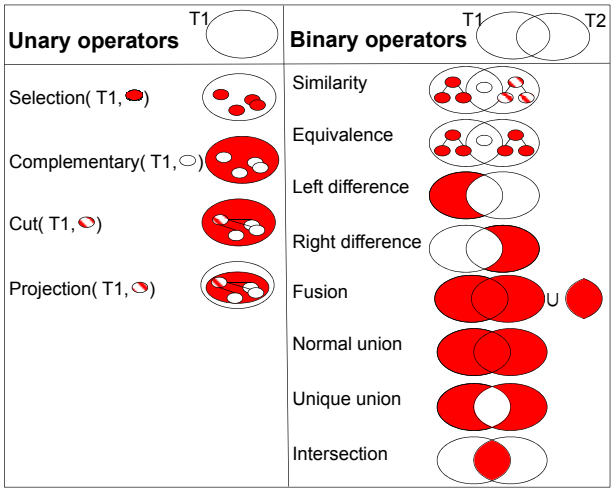


Figure 3. The set of composition operators.

Unary operators

In this part, all the operators are unary. The inputs are a tree T1 and an item or a set of items defined by a pattern tree. The output is a tree.

Selection operator (σ(T1, P))

The selection operation applies on a tree and on a set of items defined by a pattern tree. The output tree contains the set of items corresponding to the pattern tree in input which exist in the tree T1. The items are individual components, as leaf nodes. The output tree has the same structure (layout) as the input tree T1. The algorithm is provided in [20].

Complementary operator

The Complementary operation applies on a tree and on a set of items defined by a pattern tree. The output tree is the input tree minus the set of items corresponding to the pattern tree. The algorithm for CUI model is presented in Figure 4.

Cut operator

The Cut operation applies on a tree and on an item. The output tree is the input tree minus the item (I). If the item (I) is not a leaf, all the items of the sub tree (I) are deleted.

Projection operator ((T1, P))

The Projection operation applies on a tree and on an item. The output is the sub-tree defined from the item. The objective of this operator is not to provide an interface but rather to select a logical set of items. Thus, this operator can be used to capitalize a pattern from an existing interface.

```
Complementary (courant : tree, expression : Item)
// Depth First Search
If (courant != NULL) {
        If (courant.attribut != NULL)
                {If (courant.attribut = express on)
                        {// delete item
                        If (courent.child != null)
                                { //a child became parent
                                courant.parent.child <- courant.child
                                neighbor <- courant.child.neighbor
                                While (neighbor.neighbor !=NULL)
                                        neighbor<- neighbor. reighbor)
                                neighbor. neighbor <- courant. neighbor
                                Courant. neighbor.parent<- neighbor
                                Complerrentery (courant.child)
                                }
                        Else if (courant. neighbor!= NULL)
                                { courant.parert.chilc <- courant.neighbor
                                Courant. neighbor.parent<- courant. parent
                                Complerrentary (courant.neighbor)
                                }
                        }
                Else {//attribut != expression
                //i.e. expressior is the complementary so it is not celeted
                Complementary (courant.child)
                // Depth First Searcr
                Complementary (courant.neighbor)
                // Breadth Searcr
                }
        Else {% ro attribut so it is no treatec
        Complementary (courant.ch ld)
                % Depth First Search
        Complementary (courant.neighbor)
                % Breadth Search
        }
}
```

Figure 4. Complementary algorithm proposed to tree algebra for UI.

Binary Operators

In this part, all the operators are binary. They apply on a pair of trees T1 and T2. The Similarity and Equivalence operators provide a Boolean whereas the other operators provide a tree T3.

Similarity operator (T1 ~ T2)

The Similarity operation allows checking whether the two input trees are similar. A distance criterion is computed using the Levenshtein distance from the contents of the two trees (not the structure) [20]. This distance informs about the similarity between two strings through an evaluation of the minimal number of char operations (deletion, insertion or replacement) to transform a string into another one. In this evaluation - as in the detection of common parts - we suppose the semantic alignment of the user interfaces.

Equivalence operator (T1 ≈ T2)

The Equivalence operation allows knowing whether two trees are equivalent. Two interfaces are said to be equivalent if they are 100% similar.

Left (resp. right) difference operator (T1 / (T1∩T2)

The output tree of the Left (resp. right) operation is composed of the content of T1 (resp. T2) minus the redundant items between T1 and T2.

Fusion operator (T1+T2= (T1 U T2) U (T1∩T2))

The output tree of the Fusion operation is composed of the contents of both T1 and T2; the redundant items are placed twice. At the CUI level, particularly with the graphical modality, the order of the input argument is essential as the operation is not commutative. Furthermore, it is possible to precise whether the fusion (resp. Normal Union and Unique union) has to be horizontal or vertical. Such a parameter allows structuring the T3 output [20].

Normal union operator (T1 U T2)

The output tree of the Normal Union operation is composed of all contents of T1 and T2 but, contrarily to the Fusion operator, the redundant items are placed only once. An illustration of this operator with a horizontal parameter is shown in Fig. 5 and its tree representation in Fig. 6.

Unique union operator ((T1 U T2)/(T1∩T2))

The output tree of the Unique Union operation is composed of contents of T1 and T2. In this union, the redundant items are deleted.

Intersection operator (T1∩T2)

The output tree of the Intersection operation is composed of the items contained in both T1 and T2; the non-redundant items are deleted. The algorithm proposed to the CUI level is presented in Fig. 7.

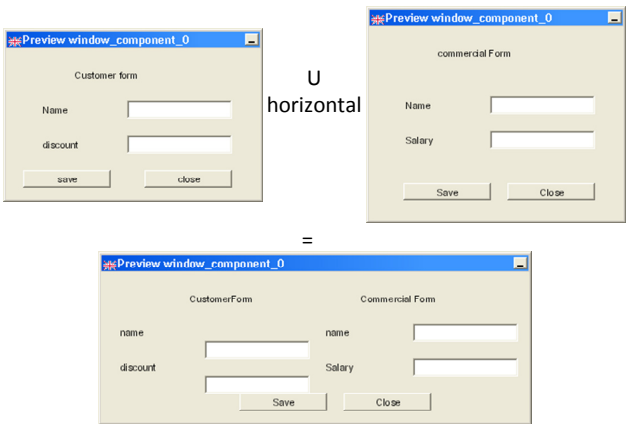


Figure 5. Normal Union operator applied on two UIs.

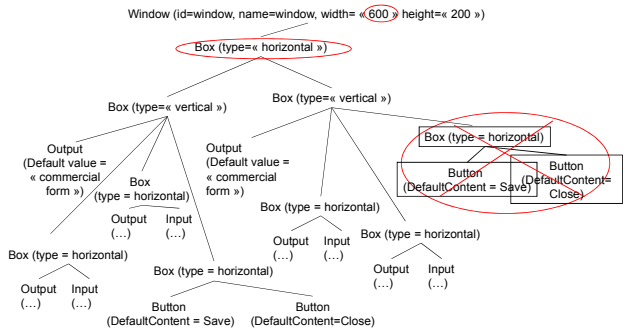


Figure 6. Normal Union operator – representation of result by tree.

```
For each component of T1,
        If (Component.defaultContent ∈ T2)
                // Hypothesis: Semantic alignment
                Add(Component, T3)
        Else {return false}
        If (Component.tag=InputText)
                // Component do not have a defaultcontent
                // IsIdentifiedBy function have to be used to know its role
        {X = Component.getIsIdentifiedBy()
         if (X ∈ T2)   Add (Component, T3)
                Else return False
        }
Return T3
```

Figure 7. Intersection on a tree algebra for CUI.

OPERATORS APPLIED ON USIXML LAYERS

This part is composed of three sections corresponding to the CUI layer, the AUI layer and the Task model layer. In each section, an illustration of some operators use is given.

Illustrations of operators use on a concrete UI

The above operators have been directly implemented in ComposiXML [20], a plug-in of GrafiXML, a graphical interface builder that automatically generates UsiXML specifications (Fig. 8) as opposed to final code for other builders. As they are an open source project regulated by Apache 2.0 open licence and available on SourceForge, they can be downloaded from http://www.usixml.org. GrafiXML is able to automatically generate code of a UI specified in UsiXML into (X)HTML or Java. For the purpose of the examples below, we will rely on the Java automated code generation.

Figure 8. User Interface of ComposiXML.

This part presents some illustrations of operators used at design time with ComposiXML. For example, they can be used by designers to create UIs which are developed for one or several applications within the same company, with the constraint of a corporate style guide. The designers can reuse some of the elements of the user interfaces thanks to operators. The second use case is at run time. It is integrated in the reuse issue which has introduced the component idea. The first issue in this domain is the composition of the components. If we consider the business component as a component with UIs, one issue in the domain of HCI is to compose the user interfaces of several business components. If we consider that the user interfaces are specified with UsiXML, the Union operator is particularly interesting for such composition of business components. Illustrations of operators are given for example in the domain of tourism. In this domain, it frequently happens that some parts of the same information should be reproduced in different UI for different events (e.g., hotel information, tourist trip including hotel booking, booking a hotel, etc.).

Fig. 9 (a) reproduces a screenshot of the preview in Java (obtained by Java automated code generation) of a Concrete User Interface created with the GraphiXML editor. Fig. 9 (b) reproduces another UI for an event management application. The two UIs only differ from a few fields, here the event dates (Fig. 9-T1) and the comment (Fig. 9-T2).

The Selection is used on the tourist application UI with a set of elements as input to give a UI subset (Fig. 9(a)).

Likewise, the Projection operator is used to extract a set of items according to their type (i.e., structure). Given as an example, the Fig. 9 (b) shows the result of the Projection operator on the tourist application with two parameters: outputText and button. The preview is in French.

Therefore, if we want to identify the common part of these two UIs, the Intersection operator performs the operation as defined previously to identify common parts of both trees and then rebuilds a new tree with the identified elements. This operator re-generates new UsiXML specifications. This intersection is reproduced in Fig. 9(c).

Note in Fig. 9 (c) that the designer did not have anything to do: all common elements were identified, a new layout was produced so as to mimic the initial one and all objects have been laid out and aligned to preserve the initial constraints.

Illustrations of operators use on abstract user interface

The AUI level allows specifying user interfaces without to know neither the modality nor the platform. It can be developed with a tool provided by UsiXML project named IdealXML. A representation of the task “order a pizza” is given in Fig. 10 (a) and its corresponding UsiXML in Fig. 10 (b). We notice that *abstractIndividualComponent* as “Choose quantity” is placed in an *abstractContainer* corresponding by “Choose a pizza”. The structure of user interface composed by container, component and relation between elements is here again respected by this model. Therefore, the operators, as at the CUI level, can be applied using tree algebra. The tree corresponding to the specification view of Fig. 10 (a,b) is in Fig. 10 (c) [18]. An example of operators using at this level is shown in the following part, in the section dealing with the multi-modality [19].

Illustrations of operators used on task tree

The operators applying on task level are similar to other levels because the UsiXML language provides a XML structure and the structure of UI follows also the set theory. However, the operators have to be adapted to take into account the relationship between tasks in the cases of fusion and Unions. Given as an example, Fig. 10 provides an illustration of the Union operator use on task trees.

Conclusion on the (de)composition operators

In this part, illustrations of operators used were given at each abstraction level according to the adaptation needs. Some works are not again completely and must be extended in the future but the illustrations give an idea of the interests to use operators to adapt and the possibility of using them at each abstraction level. The illustrations are given to describe the use of one or several operators on each level. In the following part, the adaptation needs and the use of associated operators are illustrated according to the 7 dimensions referred by UsiXML.

COMPOSE TO SUPPORT THE 7 DIMENSIONS

This part aims at illustrating the use of (de)composition operators to adapt the user interfaces according to the 7 dimensions: Multi-device Usage, Multi-User interface, Multi-linguality, Multi- organization, Multi-context usage, Multi-Modality usage, Multi-platform Usage.

Multi-device Usage

The simultaneous use of several devices sharing a same user interface implies the need to create FUI for each device. They are noted FUI'1 to *n* in Figure 11. The devices can possibly be distributed. In order to transform the initial user interface (i.e. source) into several UI parts, duplication or a new composition of UI parts are necessary. In order to decompose UI or to extract several components to be reused on other devices, the unary operators such as Selection or Projection are well suited. These operators can be applied at the CUI level (Figure 11). The decomposition may be use the task model in order to choose the part to extract/duplicate etc. At the end of this first step, le CUI' 1 to n are defined following each device. If the interface of one device must integrate elements provided by several services, then the Fusion or Union operators will be appropriated. For instance, Figure 11 illustrates the examples of a user who may control a music player running on a media center using a remote control on a handheld device, perhaps coupled with a Wii for volume control. This example is composed of three tasks: the Music controller task, the Sound player task and the volume controller task and Three devices: the media center which is used to realize the Music controller task and the Sound player task. The handheld device to make a remote control (sound controller and Music controller) and the Wii for the volume control. The source CUI is an existing UI dedicated to the Music Player task. Elements are extracted are distributed to the several devices described. We have used the CUI level to use operators because it is sufficient to this dimension but they could be also applied at the AUI level or at the task level if the UI repartition corresponds to the task(s).

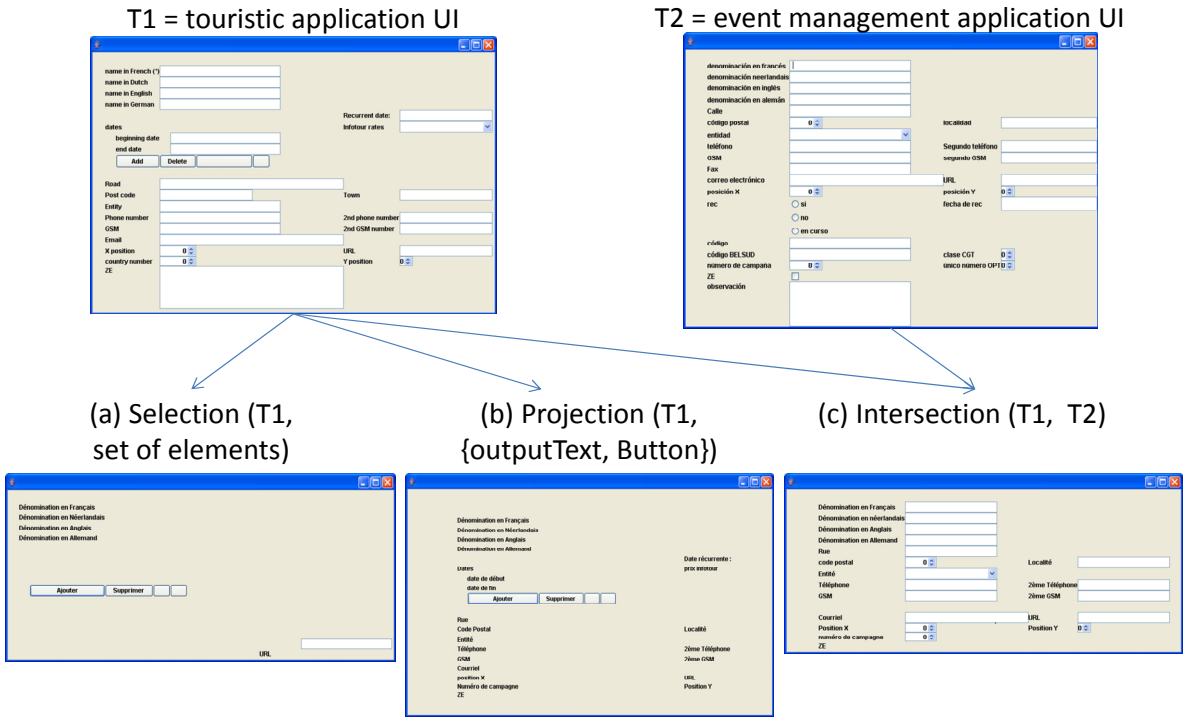
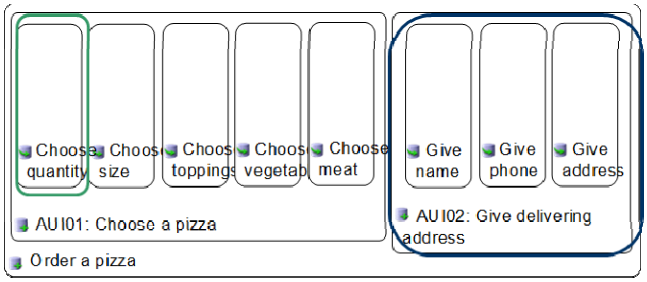


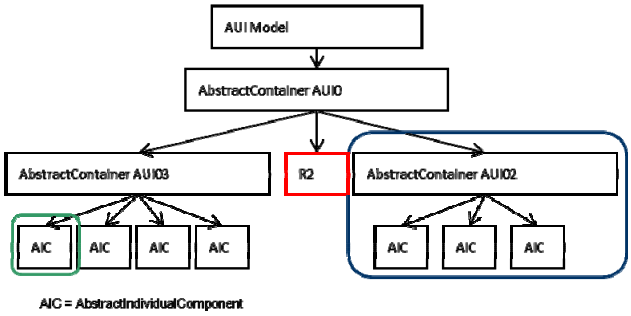
Figure 9. Illustrations of operators on CUI models.



(a) Graphical representation developed with IdealXML

```
<auimodel>
<abstractContainer id="idaio00" name="Order a pizza">
<abstractContainer id="idaio01" name="idaio01">
   <abstractIndividualComponent id="idaio02"
      name="Choose quantity">
   </abstractIndividualComponent>
   <abstractIndividualComponent id="idaio03"
      name="Choose size">
   </abstractIndividualComponent>
   …
   <abstractIndividualComponent id="idaio06"
      name="[Choose meat]">
   </abstractIndividualComponent>
</abstractContainer>
<abstractContainer id="idaio07" name="idaio07">
   <abstractIndividualComponent id="idaio08"
      name="Give name">
   </abstractIndividualComponent>
   …
</abstractContainer>
</abstractContainer>
</auimodel>
```

(b) UsiXML specification given by IdealXML



(c) Representation of the AUI model by tree

Figure 10. Three representations of the AUI model, developed with IdealXML, concerning a pizza order.

The tasks trees in the illustrations are created with the CTTE tool (http://giove.isti.cnr.it/tools/ctte/) [26]. Incidentally, they proposed to use CTT in order to adapt the UI following the device in [26]. Note that [7] propose composition-oriented user interface design patterns (CUIDP) with the same goal to reuse the UI knowledge. This work is also based on model-driven architecture and is based on UMLi to specify the UI.

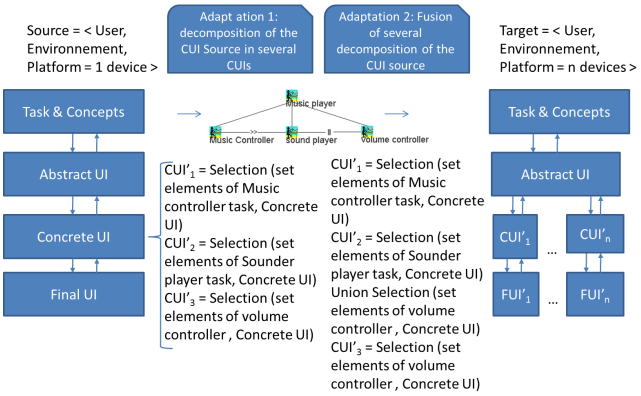


Figure 11. Definition of operators for Multi-device Usage.

Multi-User interface

Following this dimension, two adaptation propositions are possible. If the shared interface is co-localized, the users can simultaneously interact on one interface (for example a tactile multi-touch interface or tangible objects used [16]). In this case, it seems not necessary to adapt the interface by composition even if it should be adapted to the user (profiles, preferences, etc.). But, the problem is different if a user arrives in the collective work situation (or leaves it): the user needs a work space, with personal data and/or functions, and in a case of an interactive table for instance, the multi-user interface has to be adapted again, using composition/decomposition operators [17]. If the shared interface is distant, each user has a duplicate of this interface. In this case, duplication has to be carried out. The duplication operator is not proposed in this article (it can be viewed in [20] with application on a complete UI tree) but the Projection operator with the root node item as input allows to duplicate the whole arborescence. If a part only of UI must be duplicated, the input item of the Projection operator will be the item of the output root node. In parallel, the problem of the user rights or roles has also to be carefully considered. For instance in a brainwriting [14] session at a distance, the UI of the brainwriting moderator will be composed differently as the UI of the other participants.

Multi-linguality

Language is defined here as spoken language (not programming language). Several languages should not be used simultaneously within a same interface. Thus in many cases it seems not necessary to compose several parts of user interfaces. Nevertheless, if two similar interfaces are modeled at the CUI level (or higher) with specifications given for one (or more) language and another interface modeled with others languages, it can be possible to compose these interfaces to obtain one interface modeled for all these languages. Then, the UsiXML structure allows generating the final user interface in the selected language. But is it also important to consider translation considerations leading to necessities about composition/decomposition due to socio-cultural aspects (i.e. problems of designing UI for international use [20]): from a UI using a source language which

is read from left to right (and downwards; for instance, English), the problem is not trivial if the target UI has to use a language which is read from right to left (and downwards; for instance, Arabic), or upwards (for instance, Chinese). Indeed in several cases the different zones of the UI have to be composed differently.

Multi-organization

According to this dimension, the organization and task notions are essential. A task may be common to several organizations. The shared task must be associated to one interface. The Intersection operator allows detecting the common tasks whereas the Normal Union operator allows merging the two tasks without repetition of common parts (Fig. 12). These operators may be used at the task level of the UsiXML architecture [22]. We can note that Bihler *et al.* [2] implemented cross-application dynamic UI fusion in order to realize a task on a platform with a restricted graphical space. It is also important to recall that several organization models exist and are described in the literature (for instance in social sciences or multi-agent system domain) and lead to very different work methods. If several types of organizations are concerned, the composition/decomposition process may have to follow adapted rules or principles. An important research work has to be done on this subject.

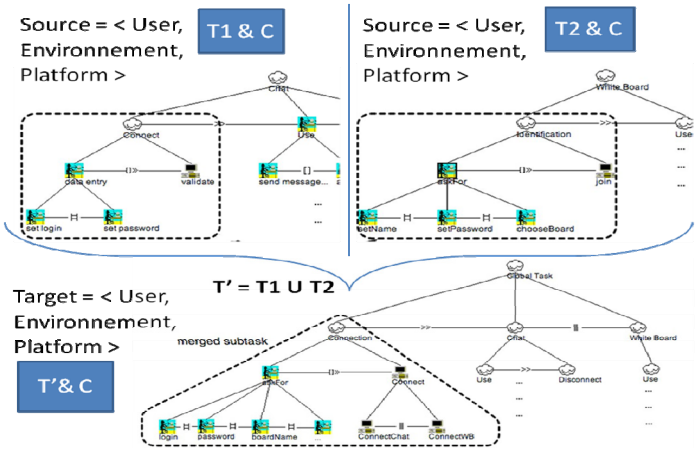


Figure 12.Normal Union operator for Multi-organization.

Multi-context usage

This dimension is integrated to UsiXML architecture since the structure with 4 levels and the transformation following the context change allow adapting user interfaces, whatever the abstraction level (Figure). A development methodology based on patterns, business component and learning, proposed in [13], is complementary to this architecture to generate and adapt the user interfaces according to the context evolution [13]. Another work focuses on the integration of the task model in the business component in order to compose complete application and to facilitate the co-evolution of the system [3]. A context modification can generate evolutions at each abstraction level. For example, a lighting change acts on the luminosity of the device (FUI) whereas the background noise change acts (or not) on the choice of the vocal modality. A modification of the user work brings modification of tasks and/or on the tasks planning.

Multi-Modality usage

The AUI layer allows specifying the UI independently of the modality (i.e., vocal, graphical, multimodal). According to needs brought by context change, a UI composition at this level (or at a higher level) allows merging interfaces which will be concretized by different modality in the lower level. For example, a food order can be realized with several modalities. Some tasks can be vocal in the case of phoning order or graphical in the case of an Internet connected device (ubiquitous or not) [19]. In this example, the Normal Union operator is used merging two applications existing partly. On one hand, in the application « order a pizza » which is illustrated above Fig. 10, the sub-task "Choose a pizza" is multimodal (see CUI model and FUI (XHTML+VoiceXML) in Fig. 13). On the other hand, the "Chinese food order" application is only graphically developed. The normal Union of these two applications gives a result which can be multimodal. The Normal Union avoids to repeat the sub-task "delivering address" which is common to the two applications (Fig. 14).

```
<outputText id="Ask_for_pizza_quantity"
        name="Ask_for_pizza_quantity" defaultContent="Quantity:"/>
    <inputText id="pizzaQuantity" name="quantity"
        defaultContent="1"
        voice_prompt="How many pizzas would you like?"
        voice_event_help="Say a number between one and twenty."
        voice_event_nomatch="Sorry I did not understand you."
        voice_event_noinput="You have to pronounce a quantity of
        pizza."/>
        <voice_quantity:grammar>
            <![CDATA[
                #JSGF V1.0;
                grammar pizza_quantity;
                public <quantity> = 1 | 2 | 3 | 4 | 5 | 6 | 7 |
        8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20
        ;
            ]]>
        </voice_quantity:grammar>
    </inputText>
<outputText id="Ask_for_pizza_size" name="Ask_for_pizza_size"
        defaultContent="Size:"/>
<group voice_prompt="What size would you like?">
        <radioButton id="radiobuttonSize1"
        name="radiobuttonSize" defaultContent="small" />
        <radioButton id="radiobuttonSize2"
        name="radiobuttonSize" defaultContent="medium"/>
        <radioButton id="radiobuttonSize3"
        name="radiobuttonSize" defaultContent="large"/>
        <voice_quantity:grammar>
            <![CDATA[
                #JSGF V1.0;
                grammar pizza_size;
                public <size> = small | medium | large ;
            ]]>
        </voice_quantity:grammar>
</group>
```

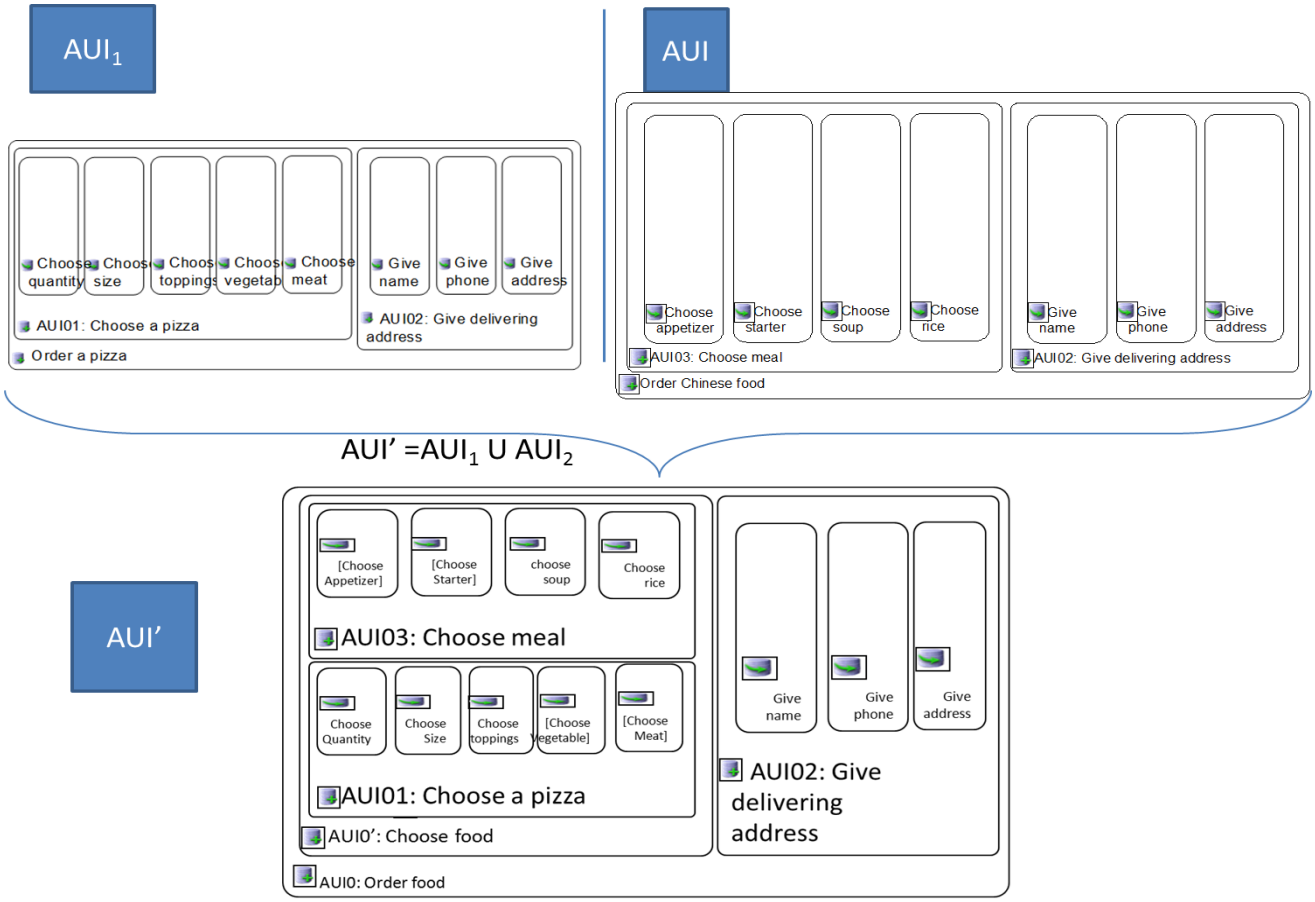


Figure 14. Normal Union (AUI (order a pizza), AUI (Order Chinese food))

Figure 13. A Multimodal CUI and FUI corresponding to the "Choose a pizza" sub-task.

Multi-platform Usage

The *multi-device usage* dimension often integrates the multi-platform dimension. So this part focuses on the adaptation of user interfaces centered on the platform only (not on the device). An example of multi-platform language is the java language. It can be executed on several platforms without changing model or implements. This dimension aims at doing the same from the user interfaces point of view. The UsiXML project allows for example to generate code from CUI model to FUI model. For example, GraphiXML (a graphical editor) allows generating java or XHTML code. In this case, we can say that the graphical user interface which is modeled with GraphiXML is multi-platform. This principle has to be generalized. According to this analysis, the composition operators are not needed into this process.

Conclusion on the support of the 7 dimensions of UsiXML

In the part here above, the adaptation of user interfaces following each of the dimensions has been explored using (de-)composition operators. Some dimensions do not require the use of (de-)composition operators whereas others have interest to use them. The analysis is realized for each dimension separately even if the 7 dimensions are complementary. Adaptations to criterion (dimension) can be done

to the detriment of the others. As a result, the adaptation must be global i.e. the adaptation has to take into account all the criteria simultaneously and not considered one by one. To go deeper, it is possible to do an analogy with McCall and his colleagues in 1977 [24] in software quality domain, considering that compromises between quality criteria are necessary from an application domain to another (for instance: efficiency is not always compatible with testability and portability). So we think that according to an application domain with its characteristics (real time or not, centralized or distributed, connectivity aspects, screen size, characteristics of the users well known or not (application for the general public), use frequency, …), compromises have to be made between the seven dimensions for insuring the best quality as possible of the final UI. More, research questions are open concerning (1) the one by one or simultaneous consideration of these dimensions, (2) the order in taking them into consideration.

CONCLUSION

On one hand, the article has presented operators which had been developed initially to compose at design time and at the CUI level. Afterwards, these operators have been used to each abstraction level of user interfaces. Likewise, some examples have shown that the operators could be used at the runtime. These operators are a support to the user interface adaptation and their use was illustrated specifically for each dimension; several research ways have been also suggested. As another perspective to this work, we propose an adaptation engine which takes into account several criteria simultaneously in order to perform this adaptation.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the ITEA2 Call 3 UsiXML project under reference 20080026 supported by the European Commission and Région Wallonne. This current research work has been partially supported by CISIT, the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the Ministry of Higher Education and Research, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions.

REFERENCES

1. Betermieux, S. and Bomsdorf, B. Task-Driven Composition of Web User Interfaces. In *Proc. of 6th Int. Conf. of Computer-Aided Design of User Interfaces CADUI'2006* (Bucharest, June 6-8, 2006). Information Systems Series, Springer-Verlag, Berlin (2007), pp. 233–244.
2. Bihler, P. and Kniesel, G. Seamless Cross-Application Workflow Support by User Interface Fusion. In *Multiple and Ubiquitous Interaction*, S. Boedker, C. Brodersen, C.N. Klokmose (eds.). DAIMI PB-581, University of Aarhus, 2007.
3. Bourguin, G., Lewandowski, A., and Tarby J.C. Defining Task Oriented Components. In *Proc. of the 6th Int. workshop on TAsk MOdels and DIAgrams TAMODIA'2007* (Toulouse, November 7-9, 2007). Lecture Notes in Computer Science, Vol. 4849. Springer-Verlag, Berlin (2007), pp. 170–183.
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers 15*(3), 2003, pp. 289–308.
5. Caramel, B., Joffroy, C., and Laguerre, M. De la composition de services à la composition d'interfaces homme-machine. In *Proc. of the 21st Int. Conf. on Association Francophone d'Interaction Homme-Machine IHM'2009* (Grenoble, October 13-16, 2009). ACM Int. Conf. Proc. Series, ACM Press, New York (2009), pp. 65–74.
6. Dery-Pinna, A.-M., Fierstone, J., and Picard, E. Component Model and Programming: a First Step to Manage Human-Computer Interaction Adaptation. In *Proc. of 5th Int. Symposium on Human-Computer Interaction with Mobile Devices and Services MobileHCI'2003* (Udine, September 8-11, 2003). Lecture Notes in Computer Science, Vol. 2795. Springer-Verlag, Berlin (2003), pp. 456–460.
7. Feng, S. and Wan, J. Multi-device User Interface Development with Composition-oriented User Interface Design Patterns. In *Proc. of the 8th ACIS Int. Conf. on Software Engineering, Artificial intelligence, Networking, and Parallel/Distributed Computing SNPD'2007* (July 30 - August 1, 2007). Volume 3. IEEE Computer Society, Washington (2007), pp. 605–610.
8. Gabillon, Y., Calvary, G., and Fiorino, H. Composing interactive systems by planning. In *Proc. of the 4th French-speaking Conference on Mobility and ubiquity computing UbiMob'2008*(Saint-Malo, May 28-30, 2008). ACM International Conference Proceeding Series, Vol. 277. ACM Press, New York (2008), pp. 37–40.
9. Gabillon, Y., Calvary, G., Mandran, N., and Fiorino, H. Composition dynamique d'interfaces homme-machine: besoin utilisateur ou défi de chercheur? In *Proc. of the 21st Int. Conf. on Association Francophone d'Interaction Homme-Machine IHM'2009* (Grenoble, October 13-16, 2009). ACM Int. Conf. Proc. Series, ACM Press, New York (2009), pp. 61–64.
10. Ginzburg, J., Rossi, G., Urbieta, M., and Distante, D. Transparent Interface Composition in Web Applications. In *Proc. of Int. Conf on Web Engineering ICWE'2007* (Como, July 16-20, 2000). Lecture Notes in Computer Science, Vol. 4607, Springer-Verlag, Berlin (2007), pp. 152–166.
11. Grundy, J.C. and Hosking, J.G. Developing Adaptable User Interfaces for Component-based Systems. *Interacting with Computers 14*(3), 2002, pp. 175–194.

12. Guerrero García, J., González Calleros, J.M., Vanderdonckt, J., and Muñoz Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHC'2009* (Merida, November 9-11, 2009), E. Chavez, E. Furtado, A. Moran (eds.). IEEE Computer Society Press, Los Alamitos (2009), pp. 36–43.
13. Hariri, M-A., Tabary, D., Lepreux, S., and Kolski C. Context aware Business adaptation toward User Interface adaptation. *Communications of SIWN 3*, 2008, pp. 46–52.
14. Heslin, P.A. Better than brainstorming? Potential contextual boundary conditions to brainwriting for idea generation in organizations. *Journal of Occupational and Organizational Psychology 82*(1), March 2009, pp. 129–145.
15. Jagadish, H.V., Lakshmanan, L.V.S., Srivastava, D., and Thompson, K. TAX : A Tree Algebra for XML. In *Proc. of 8th Int. Workshop on Database Programming Language DBPL'2001* (Frascati, September 8-10, 2001), G. Ghelli, G Grahne (eds.). Lecture Notes in Computer Science, Vol. 2397, Springer-Verlag, Berlin (2001), pp. 149–164.
16. Kubicki, S., Lepreux, S., Lebrun, Y., Dos Santos, P., Kolski, C., and Caelen, J. New Human-Computer Interactions Using Tangible Objects: Application on a Digital Tabletop with RFID Technology. In *Proc. of the 13th Int. Conf. on Human-Computer Interaction HCI'International'2009* (San Diego, 19-24 July 2009), J.A. Jacko (ed.), Part III. Lecture Notes in Computer Science, Vol. 5612. Springer-Verlag, Berlin (2009), pp. 446–455.
17. Kubicki, S., Lepreux, S., Kolski, C., and Caelen, J. Towards New Human-Machine Systems in contexts involving interactive table and tangible objects. In *Proc. of 11th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems* (Valenciennes, August 31 - September 3, 2010).
18. Lepreux, S., Kolski, C., and Abed, M. IHM et SIAD : vers une composition d'outils interactifs pour l'aide à la décision. In *Proc. of 16th Conf. on Association Francophone d'Interaction Homme-Machine IHM'2004* (Namur, August 30-September 3, 2004). ACM Int. Conf. Proc. Series, ACM Press, New York (2004), pp. 227–230.
19. Lepreux, S., Hariri, A., Rouillard, J., Tabary, D., Tarby, J.-C., and Kolski, Ch. Towards Multimodal User Interfaces Composition based on UsiXML and MBD principles. In *Proc. of 12th Int. Conf. on Human-Computer Interaction HCI International'2007* (Beijing, July 22-27, 2007), J.A Jacko (ed.), Part III. Lecture Notes in Computer Science, Vol. 4552. Springer-verlag, Berlin (2007), pp. 134–143.
20. Lepreux, S., Vanderdonckt, J., and Michotte, B. Visual Design of User Interfaces by (De)composition. In *Proc. of 13th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2006* (Dublin, July 26-28, 2006), G. Doherty, A. Blandford (eds.). Lecture Notes in Computer Science, Vol. 4323. Springer-Verlag, Berlin (2006), pp. 157–170.
21. Levenshtein, V., Efficient reconstruction of sequences from their subsequences or supersequences. *Journal of Combin. Theory, Ser. A, 93*(2), 2001, pp. 310–332.
22. Leventhal, E., and Grubis, Al. Composable User Interfaces. The MITRE Corporation, Bedford, 2004.
23. Lewandowski, A., Lepreux, S., and Bourguin G. Tasks models merging for high-level component composition. In *Proc. of 12th Int. Conf. on Human-Computer Interaction HCI International'2007* (Beijing, July 22-27, 2007). lecture Notes in Computer Science, Vol. 4550. Springer-Verlag, Berlin (2007), pp. 1129–1138.
24. McCall, J.A., Richards, P.K., and Walters J.F. *Factors in Software Quality*, 3 volumes. RADC-TR-77-369, 1977.
25. Nielsen, J. Designing User Interfaces for International Use. Elsevier Science Publishers, 1990.
26. Paternò, F., Mancini, C., and Meniconi, S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proc. of the IFIP TC13 Int. Conf. on Human-Computer interaction Interact'97* (Sydney, July 14-18, 1997). S. Howard, J. Hammond, G. Lindgaard (eds.). IFIP Conference Proceedings, vol. 96. Chapman & Hall Ltd., London (1997), pp. 362–369.
27. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008* (Iasi, September 18-19, 2008), S. Buraga, I. Juvina (eds.). Matrix ROM, Bucarest, 2008, pp. 1–10.
28. Vanderdonckt, J., Limbourg, Q., Florins, M. Deriving the Navigational Structure of a User Interface. *In Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003* (Zurich, 1-5 September 2003). IOS Press, Amsterdam, 2003, pp. 455-462.
29. Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. *Romanian Journal of Human-Computer Interaction* (Revista Romana de Interactiune Om-Calculator), Vol. 1, 2008, pp. 1–10.
30. Vanderdonckt, J., Coyette, A. Modèles, méthodes et outils de support au prototypage multi-fidélité des interfaces graphiques. *Revue d'Interaction Homme-Machine 8*, 1 (2007), pp. 91-123.
31. Vanderdonckt, J., Coutaz, J., Calvary, G., and Stanciulescu, A. Multimodality for Plastic User Interfaces: Models, Methods, and Principles. Chapter 4, in "Mul-

timodal user interfaces: signals and communication technology", D. Tzovaras (ed.), Lecture Notes in Electrical Engineering, Springer-Verlag, Berlin, 2007, pp. 61-84.

32. Vanderdonckt, J., Furtado, E., Furtado, V., Limbourg, Q., Silva, W., Rodrigues, D., Taddeo, L.. Multi-model and Multi-level Development of User Interfaces. Chapter 10, in Seffah, A. & Javahery, H. (Eds.), "Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces", John Wiley & Sons, New York, November 2003, pp. 193-216.