Chapter 19

TOWARDS A SUPPORT OF USER INTERFACE DESIGN BY COMPOSITION RULES

Sophie Lepreux[1,2] and Jean Vanderdonckt[2]
[1]Université de Valenciennes, LAMIH – RAIHM UMR CNRS 8530,
Campus du Mont-Houy – F- 59313 Valenciennes Cedex 9(France)
E-mail: sophie.lepreux@univ-valenciennes.fr
[2]School of Management, Université catholique de Louvain,
Place des Doyens 1 – B-1348 Louvain-La-Neuve (Belgium)
E-mail: {lepreux, vanderdonckt}@isys.ucl.ac.be
URL : http://www.isys.ucl.ac.be/bchi/members/{sle,jva}
Tel.: +32 10 47 85 25 – Fax : +32 10 47 83 24

Abstract The design of user interfaces is a step which takes a long time. The automatic generation of these interfaces induces shorter durations. With this automatic generation, the UIDLs have appeared. They allow specifying an interface using a Description Language. A step which also takes a long time is the redesigning of the user interface to take into accounts users remarks. We propose to use the operators of the tree algebra with a UIDL as UsiXML which is structured as a tree to improve this step of design. These operators help the designer to modify the interfaces and to reuse parts of interfaces. We have estimated the saving of time in two case studies.

Keywords: Tree algebra, User interfaces engineering, User interface extensible mark-up language.

1. INTRODUCTION

In general, the User Interface (UI) design step takes a long time. Once the specifications have been validated, the designer creates the UI mock-up and the prototype. Later, the validation step is again challenged by the users of the UI. Sometimes, it must be remade; often, it must be improved. Many design processes are iterative so this part is realized at several times [8]. Often the users request simple modifications as “move this part of the interface in another screen” or “add a field here with this information” and so on. These requests, which are simple in theory, are expensive to implement if the environment is not adapted to these needs [11]. We attempt to enrich the design environment by operators to simplify the redesign of the user interfaces. These operators are able to adjust the user interface after the deletion or the addition of an item. Consequently, they should bring a saving of time. The action analysis brings us a mean to estimate this saving of time [7]. Moreover, these operators allow the reuse of existing user interfaces. The operators are combined with an UIDL (User Interfaces Descriptor Language) to reuse parts or the whole of existing user interfaces. The proposed operators are used in this paper at the design time; however they can be used at the run time. For example, the Rainbow project [4] attempts to apply some operators as fusion by union or intersection at the run time in order to obtain a context-adapted user interface. They use the language SUNML which is a simple language, adapted to structural assembling of abstract widgets and so of HCI components. As far as we are concerned, we hope to use the operators at the runtime, i.e., UIDL files interpretation should contain the results section issue of operator, as well as at the design time within the editor. This article is focused on the design time. Indeed, another problem emerges at the run time about the data contained in the widgets.

The second part presents the UIDL which is used in this research, named UsiXML. The third part introduces the operators used to design and redesign UI and which are implemented in the editor associated to UsiXML named GrafiXML. The operators are evaluated with the GOMS method upon two case studies. The first one aims at illustrating the help brought by the operators to designing a UI and the second one aims at validating the operators to the redesign of UI. The paper ends with a conclusion and future work.

2. DESIGNING USER INTERFACES WITH USIXML

To allow high-level design operations on any GUI, we should rely on a high level description of the initial user interface. This description will be expressed in the UsiXML (User Interface eXtensible Markup Language – http://www.usixml.org [9]) UIDL. The principles set out below are, however, generally applicable to any UIDL such as UIML [1] or XIML [5]. UsiXML is structured according to the four abstraction levels of the CAMELEON reference framework [2] for multi-target UIs (Fig. 1). 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 (CIO), layout, navigation, and behavior. An *Abstract User Interface* (AUI) abstracts a CUI into a definition that is independent of any interaction modality [9].

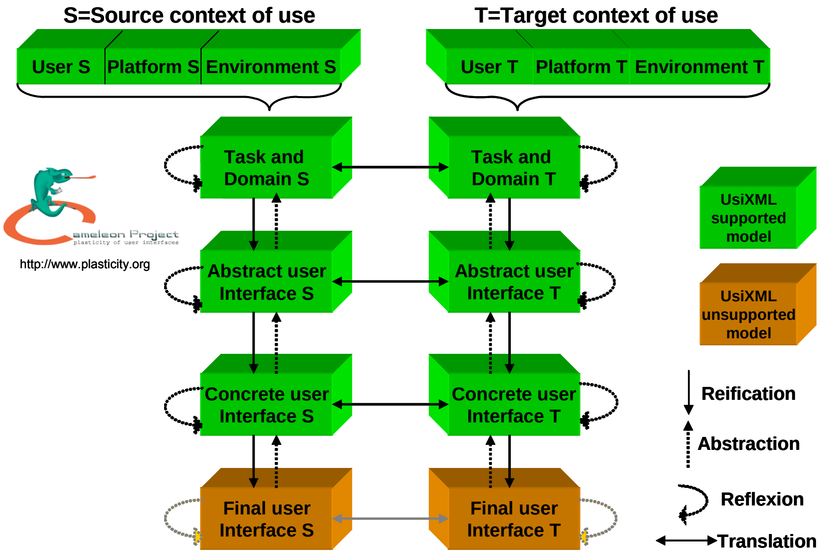


Figure 1. The four abstraction levels used in the framework.

The Tasks & Concepts level describes the interactive system specifications in terms of the user tasks to be carried out and the domain objects of these tasks. As the operators used to compose the user interface will be defined, in the first time, to be specific to one modality (e.g., graphical, vocal), the CUI level is the best candidate for a formal definition.

3. USING OF THE OPERATORS FOR THE USER INTERFACES COMPOSITION

To improve the development step of user interfaces corresponding at the time where the user asks the designer to modify the user interface, we propose to use a set of operations such as the union, the intersection, the selection, which is developed in the next part. These operators can also be used to reuse existing user interfaces. In general, the same information can be found in several user interfaces within the same application or the same domain.

3.1 Defining of the Operators

The operators are first defined, based on the UsiXML concepts of a Concrete User Interface. Then, the implementation of the operators in an existing editor is described. For the most important operators, we provide a complete formal definition which is defined thanks to a tree algebra for XML [6].

3.1.1 XML Document as a Tree Structure

Jagadish *et al.* define a data model [6]. 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-labeled and edge-labeled tree such that:

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

This pattern is used to define a database and to define the predicate used in the operations. This notation is specific to the database. So we propose 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 sub-element than that the window have a height=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. With this notation, the Selection, the Normal union and the Difference operators are formal defined here.

Selection. The selection is a unary operator. It takes a tree in input with a pattern tree and gives a tree in output. The definition brought by Jagadish *et al*. can be adapted. The output $\sigma(T, P)$ of the selection operator is a tree. The output is defined as follow. A node u in the input tree T belongs to the output iff u matches some pattern node in P, or u is a descendant of a node v in T which matches some pattern node w. Whenever nodes u, v belong to the output such that among the nodes retained in the output, u is the closest ancestor of v in the input, the output contains the edge (u,v). The relative order among nodes in the input is preserved in the output, i.e. for any two nodes u,v in the output, whenever u precedes v in the tree T, u precedes v in the tree of the output. For example, the selection operator applied with the pattern presented in Fig. 2b to the tree presented in Fig. 2a and is illustrated in Fig. 3.

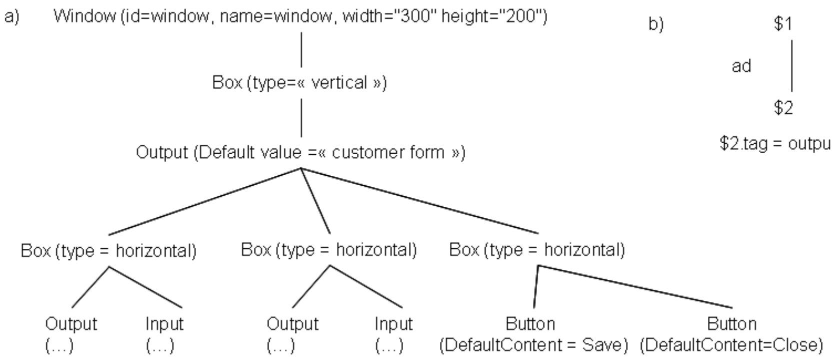

a)
Window (id=window, name=window, width="300" height="200")
Box (type=« vertical »)
Output (Default value =« customer form »)
Box (type = horizontal)
Box (type = horizontal)
Box (type = horizontal)
Output (…)
Input (…)
Output (…)
Input (…)
Button (DefaultContent = Save)
Button (DefaultContent=Close)
b)
$1
ad
$2
$2.tag = output

Figure 2. (a) A one-tree XML interface ; (b) a pattern tree.

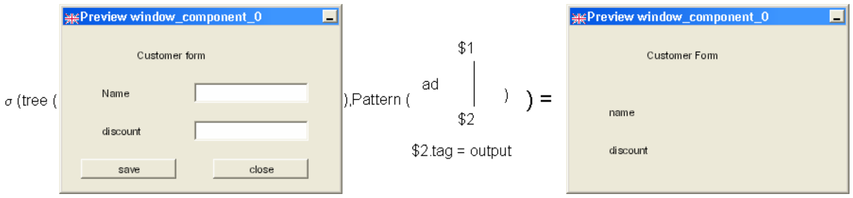


Figure 3. Selection of output in a user interface.

Normal Union. The Union operator takes a pair of trees T1 and T2 as input and produces an output tree as follows.

Firstly, the root of the output tree T3 is created:

If (T1.$1.tag == T2.$1.tag ==window) then T3.$1.tag = window with

If (horizontal Union) then content.width = T1.$1.content.width + T2.$1.content..width then (vertical) content width = max (T1.$1.width, T2.$1.width) and

If (vertical Union) then content.height = T1.$1.content.height + T2.$1.content.height else (horizontal Union) content.height = max(T1.$1.content.heiht, T2.$1.content.height)

If (T1.$1.tag == window && T2.$1.tag == box) then T3.$1 = T1.$1.

If (T1.$1.tag == box && T2.$1.tag == window) then T3.$1=T2.$1.

Then

To obtain a vertical Union: The child of the new root is added with tag = Box and type = “vertical”. Their left child(s) are the children of the root of T1 and their right child(s) are the children of the root of T2.

To obtain a horizontal Union: The child of the new root is added with tag = Box and type = “horizontal”. Their left child(s) are the children of the root of T1 and their right child(s) are the children of the root of T2 (Fig. 4).

Else T3.$1.tag = box with content.type = “horizontal” or “vertical” according to type of the Union. Their left child(s) are the children of the root of T1 and their right child(s) are the children of the root of T2.

For each node in the left and right subtrees of the new root node, all attribute values are the same as the input trees.

The duplicates must be deleted (this part is not detailed here).

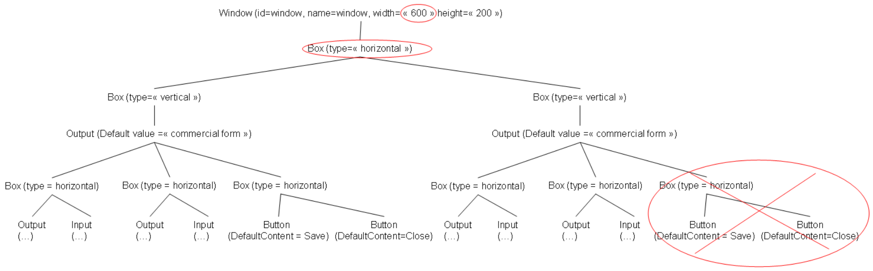

Window (id=window, name=window, width=« 600 » height=« 200 »)
Box (type=« horizontal »)
Box (type=« vertical »)
Box (type=« vertical »)
Output (Default value =« commercial form »)
Output (Default value =« commercial form »)
Box (type = horizontal)
Box (type = horizontal)
Box (type = horizontal)
Box (type = horizontal)
Box (type = horizontal)
Box (type = horizontal)
Output (…)
Input (…)
Output (…)
Input (…)
Button (DefaultContent = Save)
Button (DefaultContent=Close)
Output (…)
Input (…)
Output (…)
Input (…)
Button (DefaultContent = Save)
Button (DefaultContent=Close)

Figure 4. A Union Result.

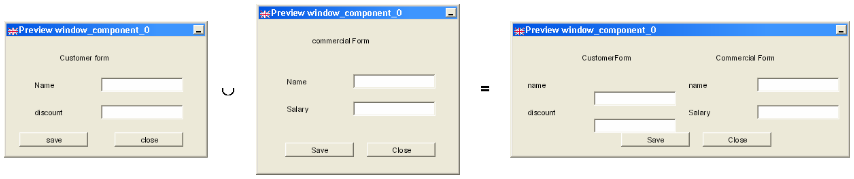


Figure 5. Union of two user interfaces.

Difference. The difference operation takes a pair of trees T1 and T2 as input and produces an output tree. The tree T2 is presented as pattern tree to identify the nodes of the second input which are in the first input tree. Then the identified nodes can be deleted. This operator corresponds to the node Deletion defined by Jagadish *et al.* [6] and adapted to the HCI domain. The delete operator takes a Tree as input and a pattern tree *P* and a delete specification (*DS*) is a sequence of expressions of the form $i or $i*, where $i is one of the node labels appearing in *P*. It generates a tree as output, as follows. Every node in *T* that matches some pattern node labeled $i in *P*, under some embedding, is marked *i*. The output tree is a copy of the marked input tree. Whenever a node *u* in the output corresponds to an input node marked *i* and the pattern node labeled $i in *P*, then

- If *DS* contains the expression $i*, the node *u* is deleted with all its descendants.
- If *DS* contains the expression $i, then node *u* is deleted, and each of its children is made a direct child of *u*'s parent. These children retain their relative order, and are inserted in the same position with respect to node *u*'s siblings as node *u* used to be.

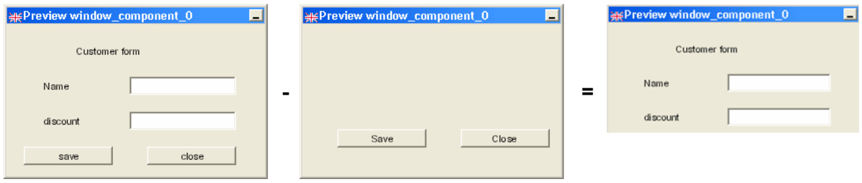

Preview window_component_0
Customer form
Name
discount
save
close
-
Preview window_component_0
Save
Close
=
Preview window_component_0
Customer form
Name
discount

Figure 6. Difference between two interfaces.

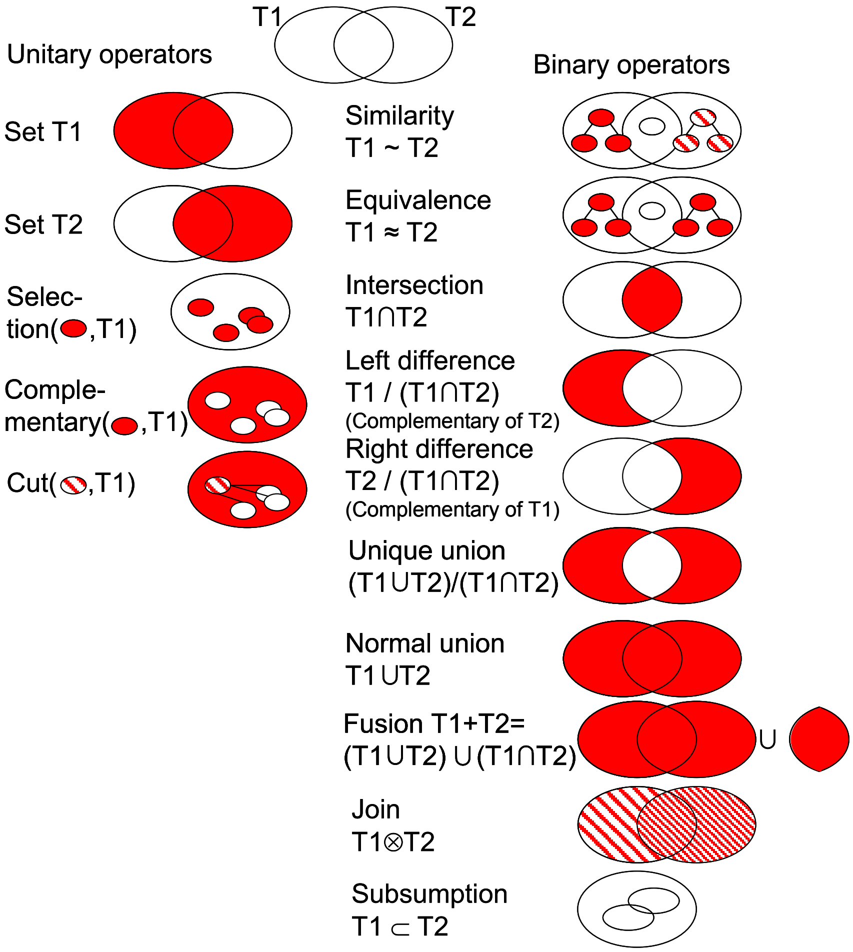


Figure 7. Illustration of composition/decomposition operators.

Due to space limitations, the definition of the remaining operators will be limited in natural language as they can be obtained by analogy to the previous ones. The set of operators is presented in Fig. 7. Each UI is shown as a set. The operators are unitary as: **Set**: The input of the set operator is one

XML tree. The output is the input tree. **Selection**: The input of the Selection operator is one XML tree and an expression. The output is a new XML document. The output tree is a set of elements which correspond to the expression. **Complementary**: The input of the complementary operator is one XML tree and an expression. The output tree is a new XML document corresponding to the input tree without the elements which correspond to the expression. **Cut**: The input of the cut operator is one XML tree and a parameter as a Node. The new output tree corresponds to the input tree without the elements which correspond to the parameter. **Projection**: The input of the projection operator is one XML tree and parameter as Node (*N*). The output tree corresponds to the search node and its child.

Among the binary operators (the input is two XML trees) are some operators aimed at comparing two UIs. They have a Boolean result: **Similarity** is used to compare the structure and not the data. **Equivalence** is used to compare the structure and the data. **Subsumption** is used to verify that one is a subset of the other.

The other binary operators allow the extracting of parts of the input trees: **Left or Right Difference** is used to extract the common part of two trees from one (right or left). **Fusion** is used to assembly the two trees with the repetition of the common part. **Normal Union** is used to assembly the two trees without repetition. **Unique Union** is used to assembly the two trees without the common part. **Intersection** is used to select only the common part of the two trees. **Join** is used to concatenate the set of nodes of the two input trees in function of the common nodes.

3.1.2 Implementation

Some of the above operations have been implemented in GrafiXML, a graphical interface builder that automatically generates UsiXML specifications as opposed to final code for other builders. GrafiXML has been implemented in Java 5.0 and today consists of more than 100,000 lines of Java code. It can be freely downloaded from http://www.usixml.org as it is an open source project regulated by Apache 2.0 open licence and available on SourceForge. 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.

3.2 Evaluation of the Benefits brought by the Operators

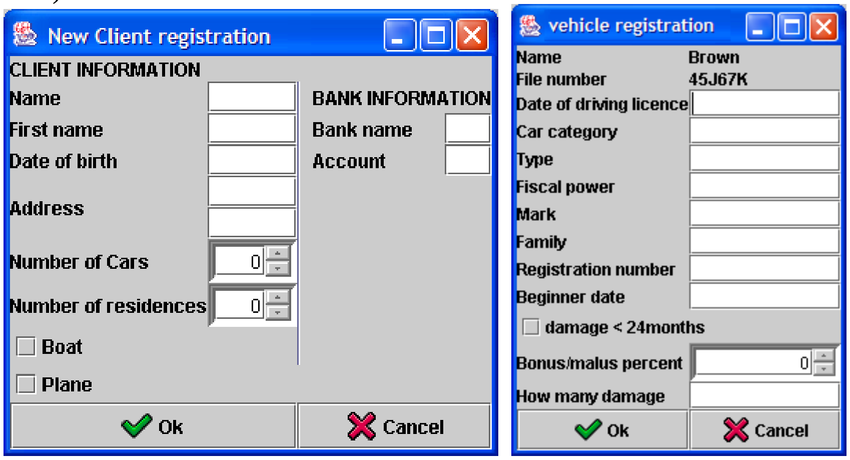
In this part, two case studies of UI design are presented. The first case study, in the insurance domain, aims at showing how the operators can be used to reuse the parts of one or more user interfaces. The second case study aims at presenting how to use the operators to modify an existing user interface according to user's suggestions. The GOMS (Goals, Operations, Methods and Selection rules) model is used to evaluate the saving in time (Table 1 [3,10]). For example, to modify a title of a window, the user right click on the window which costs 0.3s, to move hand to point device, 0.075s to execute a mental step, 1.5s to use a mouse and 0.075 to execute a mental step. Then, the user hits the title which costs 0.28s by letter, 0.075 to execute the mental step. To finish the user clicks on "validate". It costs 1.5s to use mouse and 0.075 for the mental step. The using of an operator costs 1.5 to use the mouse, 0.075 to execute the mental step, 1.2 to choose among methods and 0.075 to execute the mental step, the result cost is 2.85s.

Table 1. Average times for computer interface actions.

Physical Movements		
Enter one keystroke on a standard keyboard	.28 second	Ranges from .07 second for highly skilled typists doing transcription, to .2 second for an average 60-wpm typist, to over 1 second for a bad typist. Random sequences, formulas, and commands take longer than plain text.
Use mouse to point at object on screen	1.5 second	May be slightly lower – but still at least 1 second – for a small screen and a menu. Increases with larger screen, smaller objects.
Move hand to pointing device or function key	.3 second	Ranges from .21 second for cursor keys to .36 second for a mouse.
Mental actions		
Retrieve a simple item from a long-term memory	1.2 second	A typical item might be a command abbreviation ("dir"). Time is roughly halved if the same item needs to be retrieved again immediately.
Execute a mental "step"	.075 second	Ranges from .05 to .1 second, depending on what kind of mental step is being performed.
Choose among methods	1.2 second	Ranges from .06 to at least 1.8 seconds, depending on complexity of factors influencing the decision.

3.2.1 First Case Study: Operators for Reusing User Interface Parts

We consider an information system in which the same information can be found in several interfaces, as for the insurance application. The needs of insurance companies is to manage the relation with customers (registration, movements, terminate), to manage damages (report, negotiate with another company, compensate the client) and so on. To design the user interfaces of such an application, the designer begins by specifying the items to place, places the items which compose the interface, and finishes positioning and putting them together. The designer makes the two first initial user interfaces corresponding to the Client registration and the Vehicle registration (Fig. 6). From these first interfaces, others interfaces can be partly designed using the operators. The forms “Client Modifying” and “Vehicle Modifying” (Fig. 7) made with operators induce a cost of 35.025 seconds and 76.17 seconds, while those made without operators induce a cost of 37.875 seconds and 101.82 seconds (Table 2). The use of operators leads a benefit about 6% and 25% for the two considered forms. The using of the operators is still significant with the design of the bill UI (Fig. 8). The benefit equals 92.72 % (Table 3).

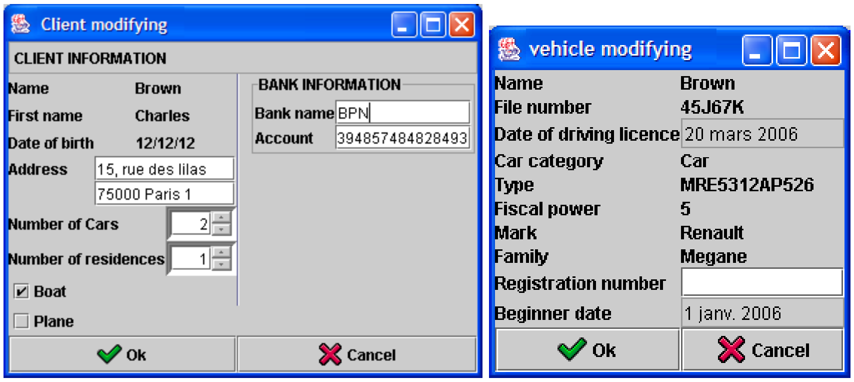


CLIENT INFORMATION
Name
First name
Date of birth
Address
Number of Cars 0
Number of residences 0
□ Boat
□ Plane

BANK INFORMATION
Bank name
Account

Name Brown
File number 45J67K
Date of driving licence
Car category
Type
Fiscal power
Mark
Family
Registration number
Beginner date
□ damage < 24months
Bonus/malus percent 0
How many damage

Figure 8. Initial User Interfaces for Client Registration and Vehicle Registration.



CLIENT INFORMATION
Name Brown
First name Charles
Date of birth 12/12/12
Address 15, rue des lilas
75000 Paris 1
Number of Cars 2
Number of residences 1
☑ Boat
□ Plane

BANK INFORMATION
Bank name BPN
Account 394857484828493

Name Brown
File number 45J67K
Date of driving licence 20 mars 2006
Car category Car
Type MRE5312AP526
Fiscal power 5
Mark Renault
Family Megane
Registration number
Beginner date 1 janv. 2006

Figure 9. Client Modifying and Vehicle Modifying UI Derived from the initial UI.

Table 2. Time evaluation to design UI.

Goal	Action	without operators	Time	with operators	Time
Create the Client Modifying	Modifying of the title	Right Click on the window Tape the new title	1.5+0.075+ 0.3+0.075+ 0.28*9+0.075+1	Right Click on the window	1.5+0.075+ 0.3+0.075+ 0.28*9+0.075

Goal	Action	without operators	Time	with operators	Time
UI from the Client Registration UI		Validate	.5+0.075 =6.12	Tape the new title Validate	+1.5+0.075 =6.12
	Modifying of 3 items from input to output	Point the input Right Click with the mouse to delete Select output Place the Output Click Tape Validate	3*(1.5+0.075+1.2+0.075+ 1.5+0.075+ 1.5+0.075+ 1.5+0.075+0.3+ 0.28*10+0.075+ 0.3+1.5+0.075) =**37.875**	Selection Difference Select output Place the Output Click Tape Validate	1.5+0.075+ 1.2+0.075+ 1.5+0.075+ 1.2+0.075+ 1.5+0.075+ 3*(1.5+0.075+ 1.5+0.075+ 1.5+0.075+0.3+0.28*10+0.075+0.3+1.5+0.075)=**35.025**
Create the Vehicle Modifying UI from the Vehicle Registration UI	Modifying of the title	Same as client	6.12	Same as Client	6.12
	Modifying of 6 items from input to output	The same as Client with 6 in place of 3	75.75	Selection Replace 3 by 6.	64.35
	Delete of 7 items	Point the input Right Click with the mouse to delete	7*(1.5+0.075+1.2+0.075) =19.95 **=101.82**	Selection and Difference	5.7 **=76.17**

Table 3. Time evaluation to design UIs.

Design	without operators	Time	with operators	Time
Create the Bill UI from the Vehicle Modifying UI and Client Modifying UI	Modifying of the title Copy of 14 items from Client UI Copy of 14 items from Vehicle UI Copy of 14 items from Vehicle UI Place all the items	4.72 32.4 32.4 32.4 119.7 =**221.62**	Modifying of the title Select Select Union Union	4.72 2.85+ 2.85+ 2.85+ 2.85+ = **16.12**

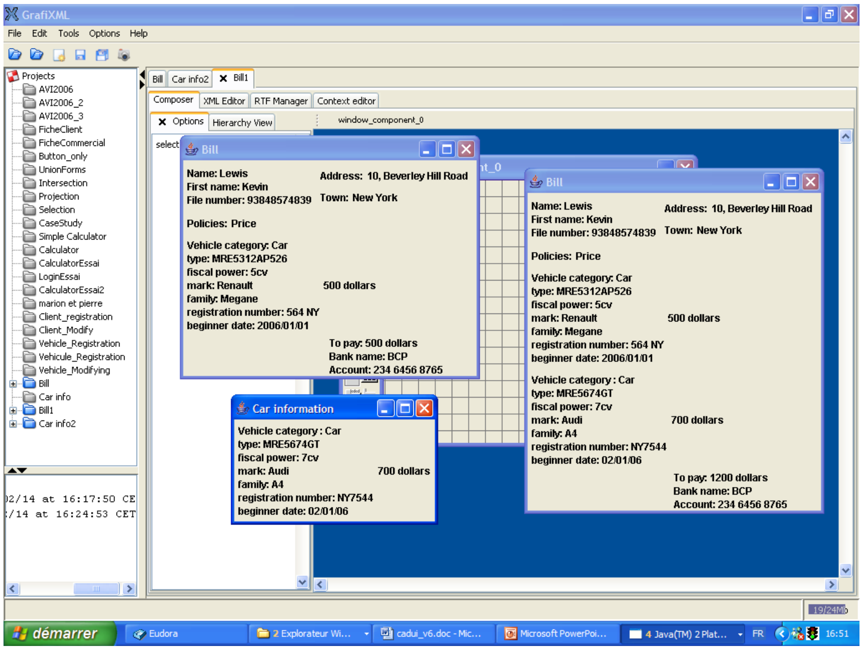
3.2.2 Second Case Study: Operators for Helping the Designer during the User Interface Design Process

This case study attempts to show the benefit in term of time induced by the using of the operators. Three cases are considered:

1. The user thinks that the user interface contains too much information. One solution is to place a set of information in a new interface. To illustrate with the insurance UIs, the user decides that the bank information

should be in another (a new) window. In this new window, the name of the client must appear.

2. It is the same case as the previous one, but the proposition is to place the information in another existing interface.
3. The UI of two interfaces are light so they could be gathered. For example, the “client registration” and the “vehicle registration” can be gathered in a new interface. The difference of time for each of the three cases is shown in Table 4. The times obtained without using operators are approximate because they depend on the number of items. In this case, the number of items is arbitrary fixed at 8 and the time to reorganize is fixed at 100s.



```
)2/14 at 16:17:50 CE
:/14 at 16:24:53 CET
```

Bill

Name: Lewis
First name: Kevin
File number: 93848574839
Address: 10, Beverley Hill Road
Town: New York

Policies: Price

Vehicle category: Car
type: MRE5312AP526
fiscal power: 5cv
mark: Renault 500 dollars
family: Megane
registration number: 564 NY
beginner date: 2006/01/01

To pay: 500 dollars
Bank name: BCP
Account: 234 6456 8765

Car information

Vehicle category : Car
type: MRE5674GT
fiscal power: 7cv
mark: Audi 700 dollars
family: A4
registration number: NY7544
beginner date: 02/01/06

Bill

Name: Lewis
First name: Kevin
File number: 93848574839
Address: 10, Beverley Hill Road
Town: New York

Policies: Price

Vehicle category: Car
type: MRE5312AP526
fiscal power: 5cv
mark: Renault 500 dollars
family: Megane
registration number: 564 NY
beginner date: 2006/01/01

Vehicle category : Car
type: MRE5674GT
fiscal power: 7cv
mark: Audi 700 dollars
family: A4
registration number: NY7544
beginner date: 02/01/06

To pay: 1200 dollars
Bank name: BCP
Account: 234 6456 8765

Figure 5. The design of the Bill UI with Union operator applied to mediate UIs.

Table 4. Time evaluation to modify the UIs.

Design	without operators	Time	with operators	Time
Case 1	Select and cut the items to ex-	18.51s	Select items to extract (with Se-	2.85s
	tract	1.275s	lection operator)	
	Create a new project	1.575s	Difference	2.85s
	Paste the items in the new	1.275s	Select items to duplicate (with	2.85s
	project		Selection operator)	
	Select and copy the items to	18.51s	Union	2.85s
	duplicate			=**11.4s**
	Paste the items	1.275		
	Reorganize the first interface	100s (arb)		

Design	without operators	Time	with operators	Time
	Reorganize the new interface	100s (arb) = **242.42s**		
Case 2	Select and Cut Paste in another UI Reorganize the first one Reorganize the second one	18.51s 1.275s 100s(arb) 100s(arb) =**219.785**	Select items to extract Difference Union	2.85s 2.85s 2.85s **=8.55s**
Case 3	Select all the items of one UI Paste in the other Reorganize the final UI	18.51s 1.275s 100 (arb) =**119.785**	Union	**2.85s**

4. CONCLUSION

This paper presented some composition operators coming from tree algebra which are interesting for the visual design of user interfaces. Moreover, at the design time, the work granularity is not only at the individual object level but also at a upper level of a coherent set of elements. These sets of elements correspond to tasks or sub-tasks, and often a leaf of the task tree. These operators have been coupled to the UIDL named UsiXML for two reasons: (1) UsiXML is a language adapted to the specification of user interfaces and (2) the operators are adapted to a tree structure which fits well the purpose of a XML-compliant language like UsiXML. The tree algebra is an advantage as it allows manipulating user interfaces structured as a set of elements. However we think that the operators could be divided into two groups. The first one is based on the set theory; the operators of this group manipulate the elements taken individually in interfaces. The second group is based on the tree algebra; the operators of this second group manipulate sets of elements and modify both the structure and the node of trees. Two case studies have presented the using of the operators for a simplified insurance application and the benefit of time has been evaluated with the GOMS method. The interest of the using of operators has been proved even if an evaluation with real designer in a real case is still to be done.

ACKNOWLEDGEMENTS

We gratefully thank the support from the SIMILAR network of excellence (The European research taskforce creating human-machine interfaces SIMILAR to human-human communication), supported by the 6th Framework Program of the European Commission, under contract FP6-IST1-2003-507609 (http://www.similar.cc). The authors thank also the Nord-Pas de Calais regional authority (Projects MIAOU and EUCUE) and the FEDER

(Fonds Européen de Développement Régional, European Fund for Regional Development) for supporting a part of this work.

REFERENCES

[1] Ali M.F., Pérez-Quiñones, M.A., and Abrams, M., *Building Multi-Platform User Interfaces with UIML*, in A. Seffah & H. Javahery (eds.), “Multiple User Interfaces: Engineering and Application Framework”, John Wiley, Chichester, 2004, pp. 95-118.

[2] 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, Vol. 15, No. 3, June 2003, pp. 289-308.

[3] Lewis, C. and Rieman, J., *Task-Centered User Interface Design, a Practical Introduction*, 1993, accessible at http://hcibib.org/tcuid/.

[4] Dery-Pinna, A.-M. and Fierstone, J., *Construction d'interfaces utilisateurs par fusion de composants d'IHM : un atout pour la mobilité*, in Proc. of Mobilité & Ubiquité'2004 (Nice, 1-3 June 2004), ACM Press, New York, 2004, pp. 60-65.

[5] Eisenstein, J., Vanderdonckt, J., and Puerta, A., *Model-Based User-Interface Development Techniques for Mobile Computing*, in Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI’2001 (Santa Fe, January 14-17, 2001), ACM Press, New York, 2001, pp. 69-76.

[6] Jagadish, H.V., Lakshmanan, L.V.S., Srivastava, D., and Thompson, K., *TAX : A Tree Algebra for XML*, in G. Ghelli, G Grahne (eds.), Proc. of 8th Int. Workshop on Database Programming Language DBPL’2001 (Frascati, 8-10 September 2001), Lecture Notes in Computer Science, Vol. ?, Springer-Verlag, Berlin, 2001, pp. 149-164.

[7] Kieras, D.E., *Towards a Practical GOMS Model Methodology for User Interface Design*, in M. Helander (ed.), “Handbook of Human-Computer Interaction”, Elsevier Science, Amsterdam, 1988.

[8] Lepreux, S., Abed, M., and Kolski, C., *A Human-Centred Methodology Applied to Decision Support System Design and Evaluation in a Railway Network Context*, Cognition Technology and Work, Vol. 5, 2003, pp. 248-271.

[9] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V., *UsiXML: a Language Supporting Multi-Path Development of User Interfaces*, in Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSV-IS’2004 (Hamburg, 11-13 July 2004), Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.

[10] Olson, J.R. and Olson, G.M., *The Growth of Cognitive Modelling In Human-Computer Interaction Since GOMS*, Human-Computer Interaction, Vol. 5, 1990, pp. 221-265.

[11] Vanderdonckt, J., *Visual Design Methods in interactive Applications*, Chapter 7, in M. Albers, B. Mazur (eds), “Content and Complexity: Information Design in Technical Communication”, Lawrence Erlbaum Associates, Mahwah, 2003, pp. 187-203.

[12] Vanderdonckt, J. and Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.

[13] Vanderdonckt, J., *A MDA-Compliant Environment for Developing User Interfaces of Information Systems*, in O. Pastor & J. Falcão e Cunha (eds.), Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31.