Web User Interface Generation for Multiple Platforms

Francisco J. Martínez-Ruiz[1], Jean Vanderdonckt[1], Jaime Muñoz Arteaga[2]
[1]Université catholique de Louvain, Louvain School of Management
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
[2]Universidad Autónoma de Aguascalientes, Centro de Ciencias Básicas.
Av. Universidad 940, CP. 20100, Aguascalientes, Mexico
martinez@isys.ucl.ac.be, jean.vanderdonckt@uclouvain.be, jmunozar@correo.uaa.mx

Abstract

In order to produce Web User Interfaces tailored for multiple platforms. This paper introduces an algorithm for semi-automated generation of user interface containers based on a task model. User interface containers are first derived from the configuration of a task model and then refined according to parameters characterizing user and computing platform. In this way, it is possible to render container structures for user interfaces in a specific language and platform.

1. Introduction

The design of User Interfaces involves a process of gathering tasks (i.e. they are used as building blocks in order to describe the goal pursuit by the software application). Task hierarchies are abstract representations. Therefore, they are translated into more physical structures. The arrangement of these structures is not a trivial process in most of the cases. Applications are distributed over a certain number of containment structures due to temporal, spatial and cognitive load limitations. Instead of reducing these constraints to mere boundaries is possible to extract relevant information that could guide our designing process. For instance, if the container generation is aware of platform requirements in early stages of development, then it could prevent the rupture of related task groups (or the gathering or unrelated ones). In this paper we tackle these problems taking into account the semantic information coming from a neutral description of the UI, applying a set of rules based on heuristic knowledge of the relationship between operators and the introduction of a metric for weighting abstract containment structures.

The design of a UI in the Web domain implies the division of the application into Web pages. Each one covers some tasks of the application. However, the page metaphor is moving to the Single Page Application approach (SPA) [17]. In this kind of web applications, the behavior and content of a single web page is changed though dynamic modifications of the Document Object Model (DOM) that represents the web page. That is, SPAs need to deal with container structures which are dynamically transiting from visible/focused to invisible/unfocused status. For instance, there are many item-grouping libraries in the GUI world (e.g. Java layout managers). These libraries follow a general schema that we depicted in fig. 1: First, we have a frontier-component that serves as a foundation canvas for the application. Second, an undefined number of containment elements following an initial order of presentation. They are ordered but they can be presented to the user in a rotation of states from visible/available to invisible/non available.

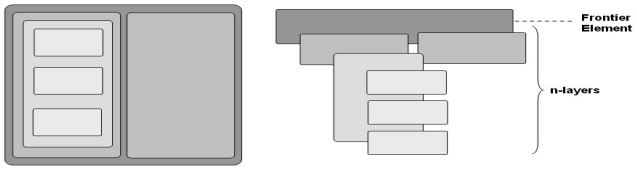

Figure 1. An example of the layers.

The rest of this paper is organized as follows: Section 2 discuss the state of the art in the creation of containers. Section 3 introduces some theory in Task models and model driven engineering domains. Then Section 4 covers the description of our method. And finally section 5 presents conclusions and future work.

2. State of the art

The difficulties that arise when you design containers include: First, how to solve the problem of distribution the UI over the available physical space since the size of the view is finite and as a consequence (in the non trivial UIs) we have to divide the UI in multiple views, this issue is

treated in [1, 4, 8]. Also this perspective tries to solve the container problem in terms of available space (i.e., geometrical constraints [15, 21] or more general restrictions [12]); the nature of the problem is NP-complex [8]. Second, how to determinate the acceptable frontier points that should be respected to create a coherent hierarchy of views. This process requires more knowledge (over the simple layout structure that is treated as proposed in [9, 10, 14] but with limitations in the recovery of information over the physical restrictions) to avoid grouping unrelated elements or breaking groups. The place to get this information in the following papers is a meta-description of the UI that is built in terms of a set of tasks: a task model. In [3, 5, 13, 20] is created a hierarchy of widgets to define in a device independent way the UI and use a bottom-up algorithm that is based in the condition of "splittable" or not of the nodes (The nodes of the tree can be labeled as splittable or un-splittable) and from there looking for the highest ancestor of the node and the resultant sub tree is marked as a page. Nevertheless, the process does not include temporal information from the task model and the division point is defined in a fixed way. In [18] the idea of using the task model is explored but in this case the temporal operators are key elements to provide information of how divide and create the containers. The task model is traversed in a Breadth-first search and through a set or principles proposes the way of reducing the UI from the less constrained platform to display it in devices with fewer capabilities in a process called "graceful degradation". In order to create the containers in [19] again the starting point is the task model and the relationship between tasks is extracted from the information enclosed in the domain model besides the identification of tasks related to fulfill user goal and supplementary tasks. In [13] temporal operators are used to propose a presentation where tasks that should be enabled at the same time are grouped in two sets: *first* and *body*. Here, the relevance of the so called first action is over-valuated while in the proposed algorithm the weighting metrics are based on more parameters (specifically, task types and operators besides the inclusion of the knowledge of the allowed deepness of hierarchies in a specific technology). The last container generation method is part of [20] which does not worry for space constraint because is oriented to discover and use the relationship between tasks and subtasks to create device-independent UIs. Also, there is work developed in this area focusing the problem as an optimization task [2, 6, 7] in our proposal we are using some heuristics based in the notion of strong repercussion of the temporal operators that are used, this is also present in [13, 20]. This paper proposes as solution in the next section: a model-based approach in order to create a feasible mapping between the finite layers and the task decomposition [13, 18, 19].

3. Model Driven Engineering Approach

Our methodology is supported by a Model driven engineering approach (http://www.omg.org/). We are going to present its core elements: the CAMELEON framework [22], UsiXML [20] and the CTT task model [13].

3.1 CAMELEON Framework

The design of UIs using a model based approach that includes features as Multi-level abstraction and Modality independence [16] requires the use of a framework to deal with the complexity of the process. We are using the CAMELEON framework [22]. This framework divides the development process in four successive levels of abstraction: Task and concepts (T&D), Abstract User Interface (AUI), Concrete User Interface (CUI) and Final User Interface (FUI). The UI is represented in the User Interface Description Language, UsiXML (UsiXML which stands for User Interface eXtensible Markup Language). This language provides the representation of the UI in the four levels of the framework, in a design independent way and over multiple contexts e.g., Character, vocal and Graphical User Interfaces among others.

3.2 CTT-based task models

The Concur Task Tree model (CTT) is a well known technique in Computer-Human Interaction to model an application in an independent platform way. The task model of UsiXML is implemented through CTTs. The objective of this model is to explain the work that the user pursuits as a hierarchy of tasks where each task is decomposed until arriving to basic tasks. The description below is very brief and a more detail description could be found in [13]. The sibling tasks (denoted as T) are related to each other through the following binary and unary operators: Concurrent Operators: These operators imply that T1 and T2 are performed in any order, in a concurrent order: |=|, ||| and |[]|. Sequential operators: [>, |>, >> and []>> these operators imply a strict sequence in the order of execution of the tasks. Selection operator: [] exclusive choice between T1 and T2. The unary operators include: The Optional operator [T] that implies the dispensable nature of some tasks. The Iterative operator T* that gives the model the faculty of describing cycles.

4. Method outline

The following section describes the proposed method to generate Web UI containers. Before going any further, we have to introduce some concepts: A level is a set of tasks recovered by an exploration of all nodes adjacent to the current task node in a breadth-first search. The root by

definition is the first level. Also we have to define the concept of layer which is a set conformed by each containment element that belongs to the same parent container e.g. in the right side of Fig. 1 are shown a UI with four layers.

In order to clarify the explanation a case study is presented (see Fig. 2). A sub-tree will be updated after each step of the method.

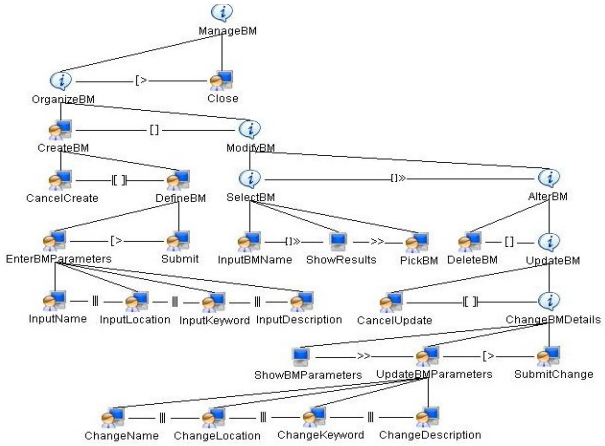

Figure 2. Task Tree Model of Bookmark manager.

4.1 Recovery of sub trees

The first step is the identification of levels (see Fig. 3). The case study includes eight levels (according to the given definition). Then sub-trees are created using as parameter the number of layers acceptable in the target platform. The procedure is as follows. The algorithm starts a bottom-up climbing of the tree searching the parent node at the nth layer. The starting point, called anchor node (see Fig. 3a) is the deepest and the most left positioned leaf node (in order to respect any possible sequence operator).

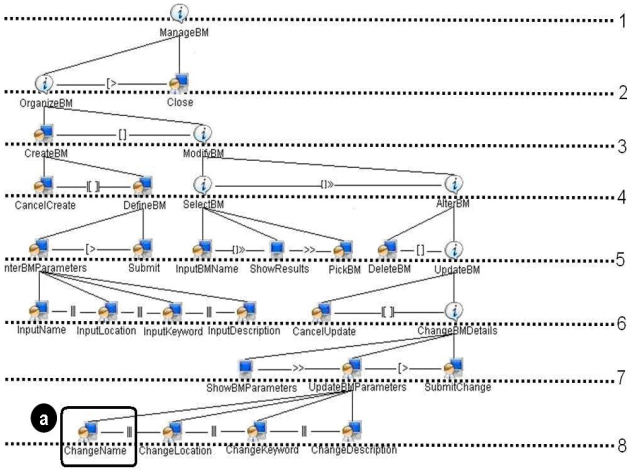

Figure 3. Levels and first anchor of task tree (a).

After that the procedure is repeated. The anchor node is relocated and the climbing restarts until it reaches the root node. The final product is a sub-tree called from now on: virtual container (VC). A formal description of the algorithm is presented in Fig. 5.

The definition of layers is done in terms of heuristic notions. Most of platforms do not impose a fixed nor static number of layers then we have to define an approximated value. For instance, in Fig. 4 the possible number of layers of four devices is presented as a guide to the designer. Next we present the result for the case of three layers in Fig. 6 (in this case the procedure delivers four containers).

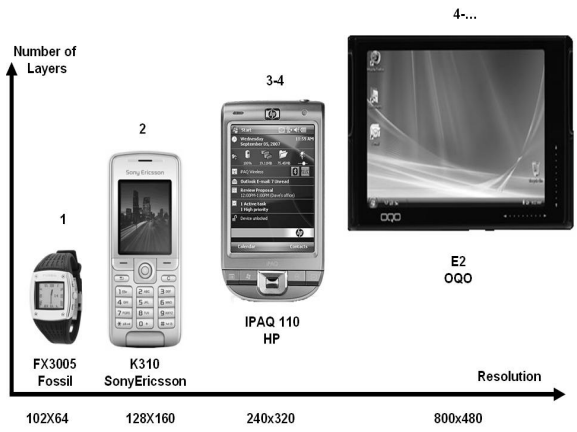

Figure 4. Features of four platforms.

```
Function CreateNextContainer(anchor, numberLayers)
        returns SubTreeStructure or failure
Layers=0
reviewedNode=anchor
While   Layers is less than numberLayers or
        reviewedNode is not rootNode do
        reviewedNode = ParentNode(reviewedNode)
        Layers = Layers + 1
If not rootNode then
        return GetSubTree(reviewedNode)
else
        return GetSubTree(rootNode)
```

Figure 5. Algorithm for generating Virtual Containers.

4.2 The generation of the internal structure

The second step is the evaluation of the internal structure of each container. This in turn provides us with the required information to generate the hierarchy of inner containers. This process is based on the generation of abstract containers [20] and we have to remember that the root task since is an inner node is marked as container not as a work to do (for instance, get or retrieve a value). Now in order to reduce the complexity of the process we are going to mark each level as a set of inner nodes if any sibling is a branch parent (e.g. Fig. 7, sections 7a and 7b are inner nodes), otherwise they are marked as leaves (see Fig. 7c). Then (1) is applied to each set of nodes (for instance Figs. 7a to 7c).

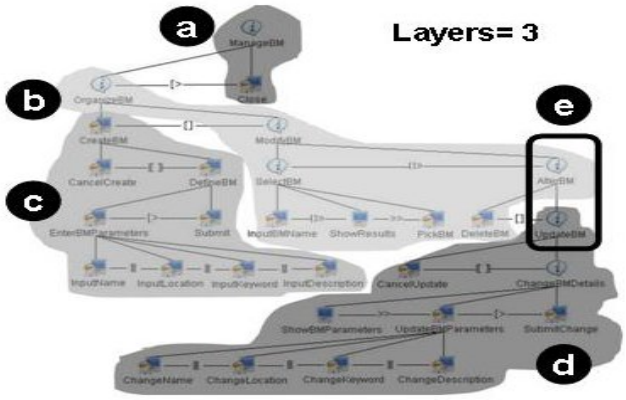

Figure 6. Virtual Containers generated.

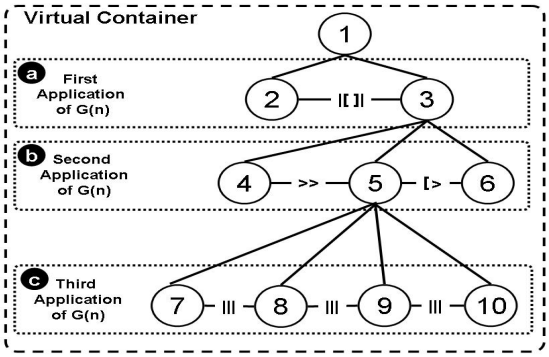

Figure 7. A simplified version of a VC (Fig. 6d).

Let **I** and **L** denote the sets of inner and leafs nodes, respectively. Let op denote the operator set formed by {C, F, and S} Where **C** is the set of all concurrent operators. **F** is the Selection operator and **S** is the set of sequential operators. Let **T** denote the analyzed task set. Finally, let **n** denote the amount of generated containers.

$$G(n)=\begin{cases} B(n)\cup 0 & \leftarrow \ T\in I \wedge op\in C \\ 0 & \leftarrow \ T\in L \wedge op\in C \\ n & \leftarrow \ T\in I \wedge op\in F \\ 1 & \leftarrow \ T\in L \wedge op\in F \\ n & \leftarrow \ T\in I \wedge op\in S \\ 0 & \leftarrow \ T\in L \wedge op\in S \end{cases} \quad (1)$$

Without the presence of any restriction the number of configurations to generate is equivalent to the problem of location of elements in a set of boxes (see Bell numbers algorithm). For instance, the hypothetical VC from Fig. 8a with three concurrent tasks (A, B and C) has six possible containment configurations according to (1). This process is presented as a formal algorithm in Fig. 9.

Now consider the VC (Fig. 6d) of the case study and in this situation is possible to deliver three configurations (see Figs. 10, 11 and 12). Another point of interest is the process of propagation of the control widgets e.g. the "close task" that is member of the first container (Fig. 6a).

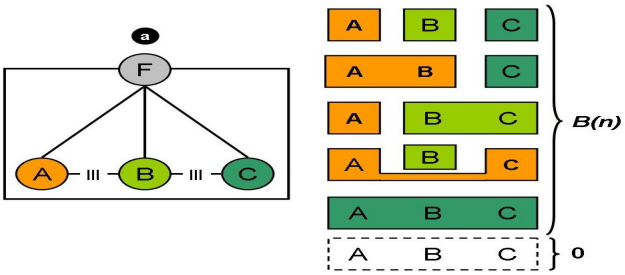

Figure 8. Example of application of G(n).

```
Function ApplyGenerationSchema(SubTreeStructure)
        returns containerStructure or failure

ParentNode = GetRootNode(SubTreeStructure)
loop do
        chooseNext(ParentNode) and expand its children
        if there are not nodes for expansion then return exit
        For each childNode do
            add childNode to ListOfNodes
            if isParent(childNode) is true then
                innerNodeFlag = true
        AddTo(GenerationFn(ListOfNodes, innerNodeFlag),VC)
return  VC
```

Figure 9: The process of generating containers.

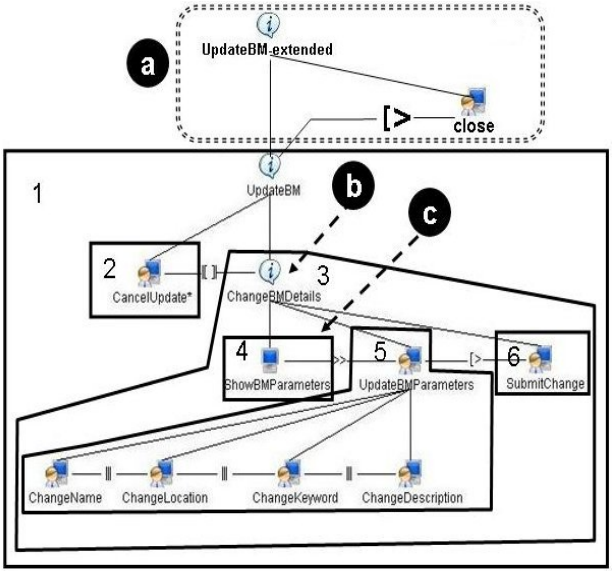

Figure 10. A configuration with 6 container units (C1).

This task should be available in all the UI then it should be propagated. The process is straightforward: the task is integrated to each children container (see Fig. 10a). It is important to remember that inner nodes as ChangeBMDetails in Fig. 10b are removed but her name should be propagated in their children containers (e.g, container 10c could be named ChangeBMDetails.showBMParameters in order to preserve information of the task hierarchy).

4.3 Choosing the configuration

After the generation of the container structures comes a weighting process, based on the values of table 1. Once again the process involves exploiting semantic information (now from task types and operators).

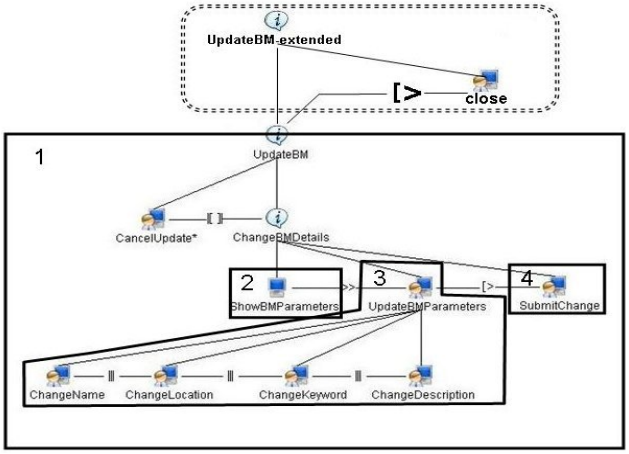

Figure 11. A configuration with 4 containers (C2).

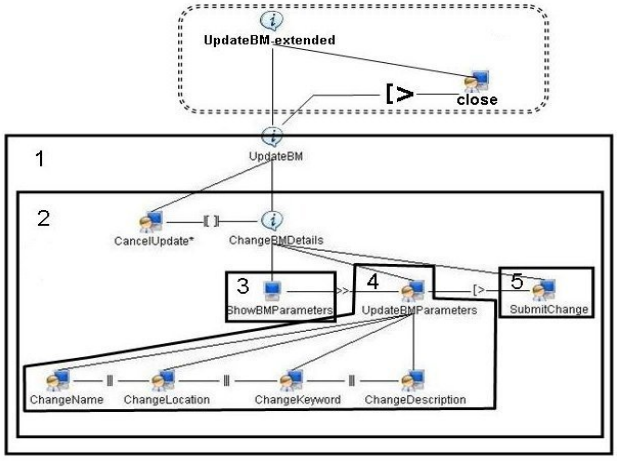

Figure 12. A configuration with 5 containers (C3).

Then we count all the exposed items of the container (leaf task and operators) using a breadth-first walk while the inner containers would be seen as black boxes and dismissed in order to apply (2).

$$value = \sum tasks \times w + \sum operators \times w \qquad (2)$$

It is worth noting that weight values are based on heuristics notions of the importance and complexity of the task types and operators and it is a pending task an evaluation of current values (as well as the layer weights). Now, it is presented in Fig. 13 the three configurations (Figs. 10, 11 and 12) as simplified weight trees. Nodes are labeled with their weight and the external value indicates the container label (Note: Fig. 13a show the weighting process).

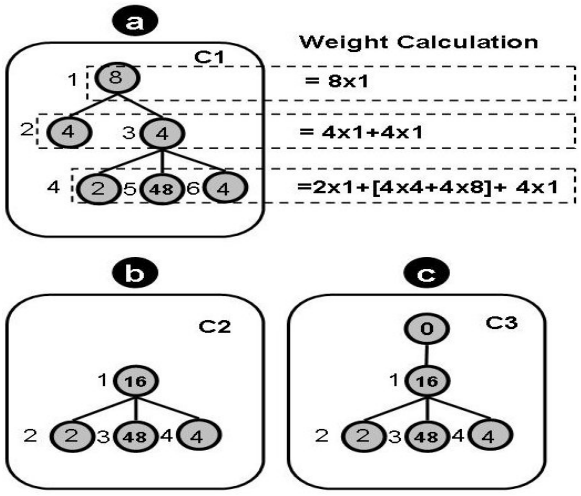

Fig. 13. Cost of each container in updateBM VC.

Now, we have to present to the designer the most suitable configuration and for that we are going to use a classic metric, the weighted average. Then, the configurations with lower cost are presented to the user. The weights are related with the number of layers that previously we have defined as constraint of the VC containers. Finally, according to the result of table 2 we should suggest our user/designer the configuration C2 (Fig. 11). The process of transformation from task model to AUI is out of the scope of this paper since it is discussed in [3, 20].

4.4 Navigational tasks

The method at this point could deliver three AUIs connected with navigation elements. For instance, the way of connected the UIs could be seen more clearly in the problem of the fragmentation of the sub tree depicted in Fig. 6e, there the task Update is a sub task of an upper tree besides it is the root node of other container. Then, the approach taken to resolve this situation is the introduction in the upper container of a navigation component pointing to the lower one. A fundamental consideration is the fact that the proposed method (for the moment) is not looking optimization. Instead of that it wants to provide the designer with plausible scenarios.

5. Conclusion and future work

In this paper we have presented an alternative method for the semi-automatic generation of the hierarchy of containers that compose a UI. This method is based on the recovery of semantic information extracted from the tree structure and the operator types. The data is extracted from the topology of the tree and the operators interacting with the tasks. The result is a feasible UI well balanced over the layers also flexible enough to allow the developer select alternative configurations.

Table 1. Weight of task tree elements.

Weight	Items to process	
	Task Type	Operator Type
8	-	Concurrent
4	Interactive	Choice
2	Application	Sequence

Table 2. Weighted average of the containers.

Layers	Weight	Calculated cost per level		
		C1	C2	C3
1	3	8	16	0
2	1	8	54	16
3	2	54	0	54
Weighted average		23.33	17	20.66

This method could help in fast prototyping as well as in exploration of news containment dispositions. Additionally, this method is platform independent since we only deal with an abstract definition of the UI. Finally, we are going to explore the integration of the algorithm into an editor tool in order to make extensive tests.

Acknowledgements: This work was supported by Alβan, the European Union Program (www.programalban.org) of High Level Scholarships for Latin America, under reference E06D101371MX and by the SIMILAR network of excellence, the European research taskforce creating human-machine interfaces SIMILAR to human-human communication under reference FP6-IST1-2003-507609 (www.similar.cc).

References

[1] Badros, G.J. Borning, A., and Stuckey, P.J. The Cassowary Linear Arithmetic Constraint Solving Algorithm, *ACM Trans. on CHI 8*, 4 (2001) pp. 267-306.

[2] Borning, A. and Duisberg, R. Constraint-based Tools for Building User Interfaces. ACM Transactions on Graphics 5, 4 (1986) 345-374.

[3] Bouillon, L. and Vanderdonckt, J. Retargeting Web Pages to other Computing Platforms with VAQUITA. In Proc. of WCRE'2002, IEEE Computer Society Press, Los Alamitos (2002), pp. 339-348.

[4] Chen, Y., Xie, X., Ma, W.-Y., and Zhang, H.-J. Adapting Web Pages for Small-Screen Devices. *IEEE Internet Computing* 9, 1 (2005), 50-56.

[5] Chu, H., Childreng, H., Wong, C., Kurakake, S., and Katagiri, M. Roam, a Seamless Application Framework. *Journal of System and Software* 69, 3 (2004), 209-226.

[6] Fogarty, J. and Hudchildren, S.E. GADGET: A Toolkit for Optimization-based Approaches to Interface and Display Generation. In *Proc. of UIST'03*, ACM Press (2003).

[7] Gajos, K. and Weld, D.S. SUPPLE: Automatically Generating User Interfaces. In *Proc. of IUI'2004*, ACM Press (2004), 93-100.

[8] Lim, A. and Zhang, X. The Container Loading Problem. In *Proc. of SAC'05*, ACM Press (2005).

[9] Lok, S. and Feiner, S.K. The AIL Automated Interface Layout System. In *Proc. of IUI'02*, pp. 202-203.

[10] Lok, S., Feiner, S., and Ngai, G. Automated User Interface Generation: Evaluation of Visual Balance for Automated Layout. In *Proc. of IUI'04*, pp. 101-108.

[11] Mori, G., Paternò, F., and Santoro, C. Design and Development of Multi-device User Interfaces through Multiple Logical Descriptions. *IEEE Trans. Software Eng.* 30, 8 (2004), pp. 507-520.

[12] Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., and Pignol, M. Generating Remote Control Interfaces for Complex Appliances. In *Proc. of UIST'02*, ACM Press (2002).

[13] Paternò, F., *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, 1999.

[14] Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., and Winograd, T. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Proc. of Ubicomp' 2001*, Springer-Verlag (2001) pp. 56-75.

[15] Sears, A. AIDE: A Step Toward Metric-based Interface Development Tools. In *Proc. of UIST'95*, pp. 101-110.

[16] Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In Proc. of *CAiSE'05*, Springer-Verlag, (2005), pp. 16-31.

[17] Mahemoff, M., Ajax Design Patterns. O'Reilly & Associates, Inc., USA, 2006.

[18] Florins, M., Simarro, F. M., Vanderdonckt, J., and Michotte, B. 2006. Splitting rules for graceful degradation of user interfaces. In Proc. of the Working Conference on Advanced Visual interfaces. AVI '06. ACM Press, New York.

[19] C. Pribeanu, J. Vanderdonckt, Exploring Design Heuristics for User Interface Derivation from Task and Domain Models, Chapter 9, in *Proceedings of 4th Int. Conf. on Computer-Aided Design of User Interfaces* CADUI'2002, Kluwe Academics Pub., Dordrecht, 2002.

[20] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces, *Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interact.Sys.* EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). LNCS, Vol. 3425, Springer, 2005.

[21] Bodart, F., Hennebert, A., Leheureux, J., and Vanderdonckt, J. 1994. Towards a dynamic strategy for computer-aided visual placement. In *Proceedings of the Workshop on Advanced Visual interfaces* (Bari, Italy, June 01 - 04, 1994). M. F. Costabile, T. Catarci, S. Levialdi, and G. Santucci, Eds. AVI '94. ACM Press, New York, NY, 78-87.

[22] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, A Unifying Reference Framework for Multi-Target User Interfaces, Interacting with comp., Vol. 15, No. 3, June 2003, pp. 289-308.