Context-aware Generation of User Interface Containers for Mobile Devices

Francisco J. Martinez-Ruiz, Jean Vanderdonckt, *Senior Member IEEE* and Jaime Muñoz Arteaga.

,

***Abstract*—The world of the mobile devices involves restrictive features. Multiple efforts have been done to define the User Interface. Here, it is combined a neutral description of the UI with a semantic recovery of information. The focus of this paper is the generation of the containment structure based on the context of the application. The process includes three main tasks: first a fixed division of the neutral definition of the UI (a task tree), Then heuristic rules based on the vicinity are applied and finally a set of container patterns is provided to polish the UI. Alternative configurations are delivered and weighted. Since multiple metrics has been proposed a comparison of them was delivered. The results of the comparative analysis suggest that some metrics, although more sophisticated and fine-grained, do not necessarily improve significantly the quality of determining user interface containers, therefore stemming for simple, yet efficient, metrics used to reach a threshold. Finally, a set of plausible UIs is delivered.**

***Index Terms*—Context-awareness, Mobil Devices, Model based engineering, user interfaces generation**

I. Introduction

The design of User Interfaces involves a process of gathering building blocks. These hierarchies of elements are the representation of the goal pursuit by the application. Also, the design of applications is affected by the utilization context which is conformed by three core components: the user, the platform and the environment. In this work, the focus is in the platform which is the supporting structure of the applications. The specific features of a context lead designers to take particular design decisions. For instance, the Mobile World introduces interesting challenges in terms of the context. According to [28], the context is any piece of data that could guide us in the characterization of the situation of elements such as temporal or spatial disposition. The status of these elements is needed to understand the interaction between user and application. In our case, these pieces of data include: the number of tiers in containment structures defined by a platform (for instance the API of a language), and the information recovered from the task tree. But usually, this is taken into account in the last steps of the design when the user interface is already done. Furthermore, the design of the containment structures of a UI is not a trivial task in most of the scenarios, so it is pertinent to define a method to build the skeleton of containers. Also, relevant information is retrieved from temporal and spatial restrictions in order to suggest alternative configurations. In summary, if the container generation is aware of platform requirements in early stages of development, then more specialized UIs are delivered.

This work proposes a method to design the UI in terms of hierarchies of containers. For this purpose, semantic information is collected from a neutral description of the UI (a Task Tree). From this tree is possible to recover the temporal and spatial constraints needed for grouping related parts of the UI. The selected case study comes from the Web domain where the paradigm is switching from the multiple web page application to the Single Page Application. These kinds of applications retrieve most of the content in a single download. Then with the aid of a scripting language, the Document Object Model (DOM) is updated in order to present (or hide) sections of the web page [10].

A. The Mobile Platform

The mobile platform is a very interesting environment to verify the awareness of the container structure. There are notable differences/constraints in the world of small devices and the most perceivable change among the desktop/laptop PCs and mobile devices is the screen size [16] also we have to take into account: limited capacities in terms of processor speed, memory and storage [17]. In the mobile devices world, basically there are four schemas for displaying content [18]:

1. **Native single and multiple authoring**. These options could deliver sophisticated results but implies the work of specialized designers for one or a set of specific devices and platforms.
2. **Automatic reconstruction**. This approach has more practical uses since we have a lot of web pages already done which require being migrated/delivered to small devices. One of the possible solutions is the web page summarization.
3. **Client-side navigation**. This approach allows the user

Manuscript received June 9, 2008. This work was supported by AlBan, 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).

F. J. Martinez-Ruiz is with The Université catholique de Louvain, Louvain School of Management. Place des Doyens, 1 – B-1348 Louvain-la-Neuve. Belgium. Also He is with the University of Zacatecas, Mexico. (e-mail: jamaru@acm.org).

J. Vanderdonckt. is with The Université catholique de Louvain, Louvain School of Management. Place des Doyens, 1 – B-1348 Louvain-la-Neuve. Belgium. (e-mail: jean.vanderdonckt@uclouvain.be).

J. Muñoz Arteaga is with the Universidad Autónoma de Aguascalientes, Centro de Ciencias Básicas. Av. Universidad, Aguascalientes. Mexico. (e-mail: jmunozar@correo.uaa.mx).

1550-4069/08 $25.00 © 2008 IEEE
DOI 10.1109/ENC.2008.34

the capability of navigating inside the web page with multiples techniques e.g. scrolling or zooming.

Since the proposed method is based in the second approach (automatic reconstruction/construction), it is important to go into detail. There are multiple dimensions to consider: syntactic vs. semantic and transformation vs. elision. The first is related with the structure, the second with content understanding, the third involves a modification of the presentation and finally, the fourth implies information removing. The first three are going to be combined in order to produce acceptable containers.

The rest of this paper is organized as follows: Section 2 discusses the state of the art in the creation of containers for applications. Section 3 introduces some theory in Task models and Model Driven Engineering domains. Section 4 covers the description of the proposed method. This section is divided in four subsections: classification of containers, generation of the Abstract User Interface (AUI), selection of UI complexity metrics, and a proposal of container patterns. Finally, section 5 presents conclusions and future work.

II. STATE OF THE ART

Several approaches have been proposed in order to tackle the problem of developing UIs for mobile devices (especially as web pages), some techniques has been presented in the previous section. Current approaches are lacking of a neutral definition and are depending on some specific technology. For instance, zooming over the web page through the mobile web browser facilities [21], the generation of summaries [20] also other methods deliver a series of tiles [22] since there are more limitations in terms of space in the mobile devices. This produces an increment in the number of pages delivered from a less restrictive environment.

Some architectures has been proposed in order to support UI description for multiple devices [19], [23] and [24] however the resultant UIs are limited to text or form based structures since more complex structures are not considered.

Multiple problems are derived from the design of containment structures. The problem of dividing the UI over a physical space is treated in [1], [7] and [3] where the problem is resolved by the creation of multiple views (i.e. sub divisions of the original web page). Meanwhile, the problem is resolved taking into account physical aspects (e.g., geometrical constraints) in [9] and [14].

Other direction is the analysis of the UI expressed as a neutral definition (usually based on a XML description). Some descriptions are based on Task Trees [2], [4]and [8]. In [13] is proposed a method near to this paper since it also used a bottom-up analysis of the task tree. Here, some nodes are marked as splittable or not in order to indicate the "cutting point" to produce a new web page. However, their method lacks the recovery of information from temporal operators. In the other hand, in [11] the temporal operators are taking into account but the goal of is a process of degrading (or reduction combined with division) to fit in more limited devices instead of proposing a creation/design method. Also in [8] the task model is used to create a presentation. Here it is defined a sets of enabled tasks which are active at the same time. The source of information is the domain model (a data model that the UI should update) besides the user's goals in [12].

III. MODEL DRIVEN ENGINEERING APPROACH

Our methodology is supported by a Model driven engineering approach [13], [14] and [15]; we are going to present its core elements: the CAMELEON framework [15], UsiXML [13] and the CTT task model [8].

1) CAMELEON Framework

The design of UIs using a model based approach that includes features as Multi-level abstraction and Modality independence [25] requires the use of a framework to deal with the complexity of the process. We are using the CAMELEON framework [15]. 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.

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 [8]. 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.

IV. METHOD OUTLINE

The method steps to generate context-aware UI containers are described in the following diagram (Fig. 1). Each time that the UI is refined more details from the context are added. The first one is the knowledge about the number of layers. Next, the vicinity of the containers and finally, the container patterns applicable to the specific platform. Now, each one of the process is explained in detail. It is worth noting that this proposal is using some heuristics that have been presented in [8]and [13].

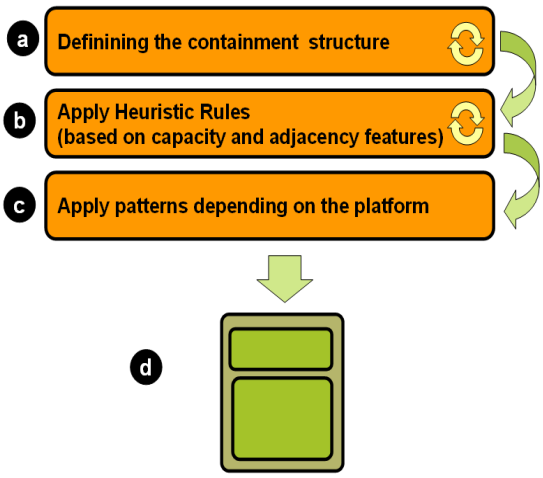

Fig. 1: Diagram of the proposed method. The steps can be refined multiple times (this is depicted with the arrows inside the blocks) and after that applied a new refinement. In (d) it is depicted the final result of the procedure, a plausible container structure.

A. Defining the Containment Structure

The next section describes the proposed method for producing a containment structure. The revised application is a minimalist weather application from the catalog of demos from open Laszlo (www.openlaszlo.org). Her goal is very straightforward: deliver weather and forecast information about an American city. (See Fig. 2). The neutral and simplified version (as a Task Tree) of this application is presented in Fig. 3. There, the UI model is divided into two main sections: ***Rain and shine*** and ***Show Results*** which are also divided in more subtasks. Finally, the execution path is represented by temporal operators that show the running order inside the task tree.

Fig. 2: Screenshot of the case study, in left the zip code is requested and the right shows a series of data of the weather.

Definitions of some terms introduced later in the paper are presented here: 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 (see Fig. 4). The root by definition is the first level. Also we have to define the concept of ***layer*** which is a tier from a hierarchy of containment elements in a specific platform (e.g. a Java component **Jpanel** is capable of carrying inside other components, so if we have a panel with a button inside this is a one-layer application). Since we are dealing with Task trees a level is renamed layer in order to represent a tier of a specific platform in the moment of evaluating the tree in order to create sub trees (container hierarchies).

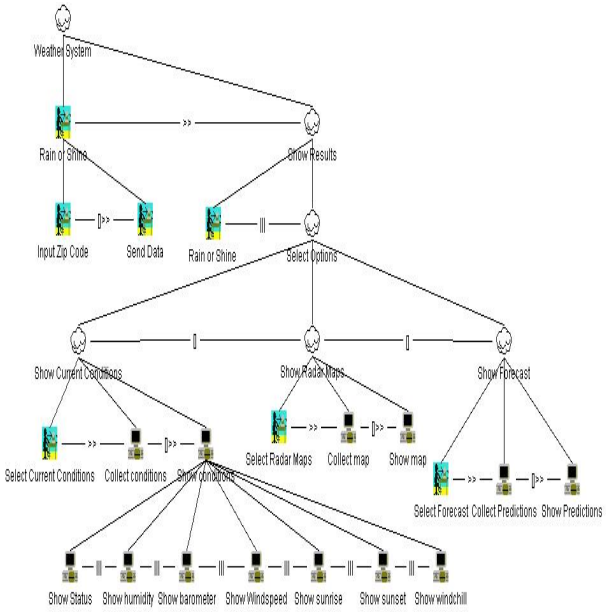

Fig. 3: Task Tree Model of weather Application.

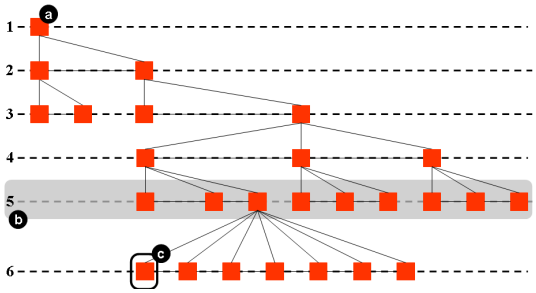

Fig. 4: Describing elements: Root (a), a level (b) and first anchor of the task tree (c).

1) Recovery of sub trees

First, we have to identify the levels (see Fig. 4). The case study produces six levels. The procedure starts a bottom-up climbing in order to search for the ***nth*** layer. The starting point is called anchor node (see Fig. 4 section c) and it's the deepest node of the tree. This process is repeated until the root node is reached. Each sub tree is called Virtual Container (VC). Note: the formal definition of the algorithm to recover VCs is presented in the Fig. 5.

```
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)
```

Fig. 5: Algorithm for generating Virtual Containers.

The number of layers is neither fixed nor static in most of the platforms. Nevertheless, it is possible to define an approximate value in order to guide the work of designers. This is more accurate in mobile devices where the restrictions are more explicit. For instance, in Fig. 6 features of a sample of mobile devices are presented. In the proposed case study, if four layers are selected then two containers are produced (see Fig. 7).

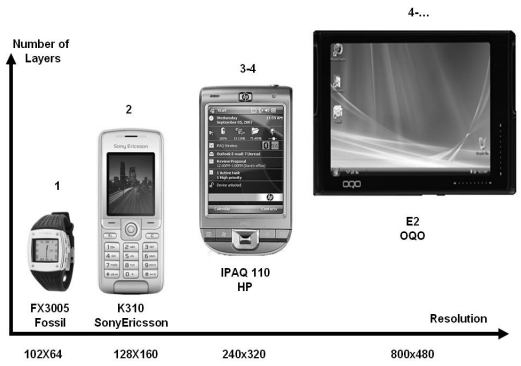

Fig. 6: Some features of four platforms.

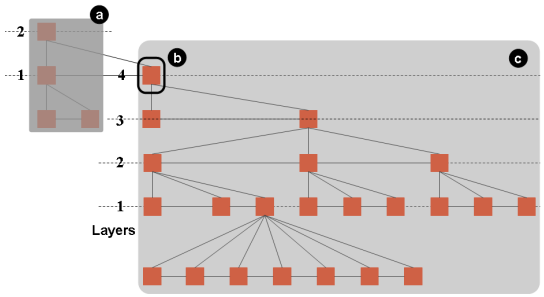

Fig. 7: Virtual Containers generated for the case study.

2) The generation of the internal structure of containers

Now, the next process is the evaluation of the internal structure of each one of the sub-trees (Fig. 7 sections a and c). This procedure is based on the generation of abstract containers of **[13]**.

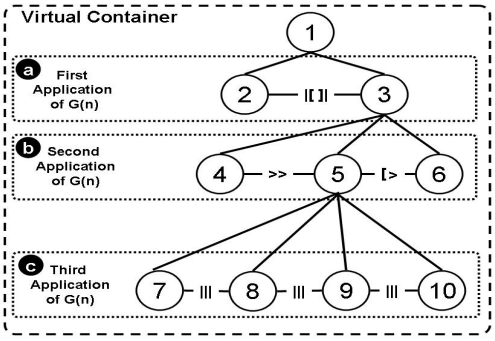

Fig. 8: Walk inside a VC in order to process its internal components.

The nodes are classified in two types: **inner nodes** and **leaf nodes**. The first category involves any sibling of a set where at least one of the nodes is itself a parent node (e.g. Fig. 8 section a). Otherwise, the nodes are marked as members of the second category (e.g. Fig. 8 section c).Then (1) is applied to each set of nodes (for instance, Fig. 8 sections a to c). The generation of containers without restrictions is equal to the calculation of the number of sub-sets or partitions of a set (Bell numbers). For instance, in Fig. 9 are depicted three inner tasks and the result of the application of (1) which delivers six possible subsets (note, the original formula of the Bell numbers does not include the empty set).

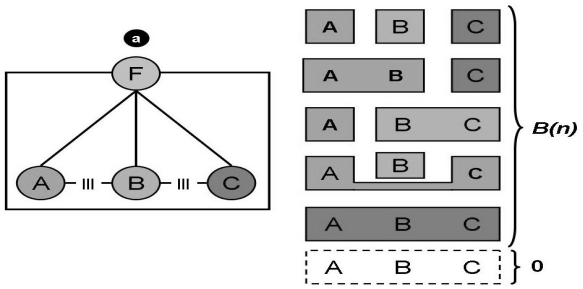

Fig. 9: example of application of the generation formula over a VC with three inner tasks.

The next paragraph describes (1): 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.

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

The formal description of this method is shown in Fig. 10. Here again with a breadth-first walk we are going to search for

parent nodes in order to analyze their children. First, marking them as inner or leaf nodes and after that to generate a containment structure depending on the application of (1).

```
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 10: The process of generating containers.

It is possible to deliver three scenarios from the VC depicted in Fig. 7 section c (these configurations are presented in Figs. 11, 12 and 13). In Fig. 13 the abstract container select options is omitted. But in order to retain the relationship between their children they are renamed. For instance, the forth container of Fig. 13 should be renamed: **SelectOptions.ShowForecast**.

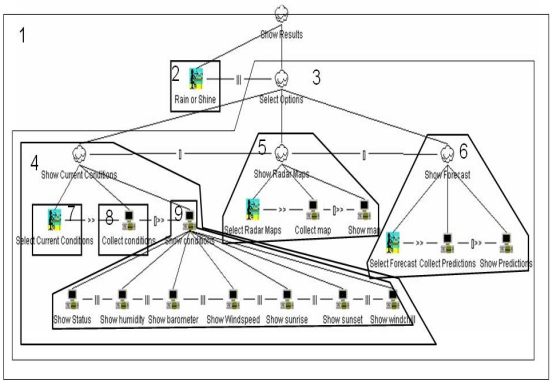

Fig. 11: A feasible configuration with 9 container units (C1).

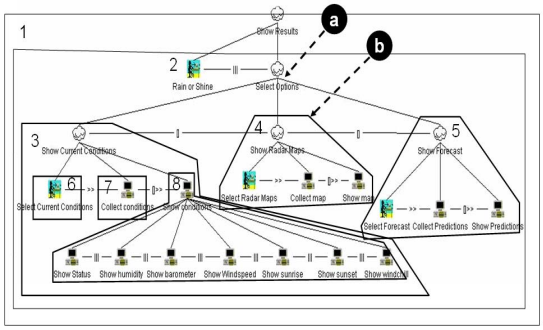

Fig. 12: A feasible configuration with 8 containers (C2).

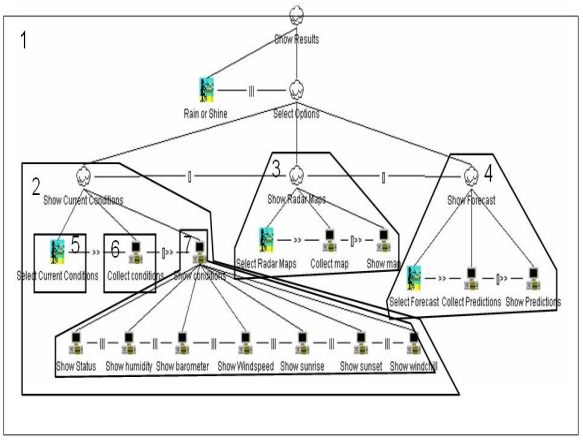

Fig. 13: A feasible configuration with 7 containers (C3).

3) Choosing the configurations

In order to choose the most suitable configuration the next step is a weighting process, based on the values of table 1. The weight is calculated taking into account their operators and tasks. The counting (2) only include the exposed elements (to reduce the complexity of the calculation). In Fig. 14 the result of this calculation is shown as reduced weighting trees.

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

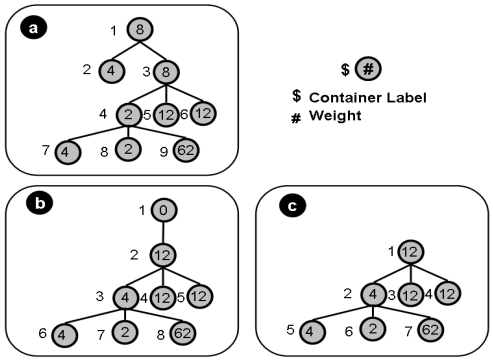

Fig. 14: Cost of each container in the Show-Results VC.

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

Table 1: Weight of task tree elements.

In table 2, it is presented the result of the calculation. Each layer has a specific weight. It is worth noting that the selection of the weight could lead to specific configurations in this case the tasks in the deep layers are penalized with more weight. Note: the process of selecting weights is based on empirical notions (indeed, the definition of a normalized process is a work in progress). According to table 2 the best configuration is **C2** (Fig. 14 section b).

Layers	**Weight**	**Calculated cost per level**		
		C1	C2	C3
1	2	8	0	12
2	1	12	12	28
3	3	28	28	68
4	2	68	68	-
Weighted average		31	29	32

Table 2: Result of weighting process

4) Navigational tasks

The VCs are connected through navigation elements following the procedure described in **[13]**. A special case is the fragmentation produced by the method for instance if a VC's root node is member of another VC the approach taken in this method is the introduction of a navigation component from one VC to the other one (at the level of the AUI definition).

B. Classification of Containers

Other source of information that it is going to give us more details to support the container building is the structure that could take our task tree. Even if the amount of combinations is big, it does not exclude the possibility of proposing a categorization of the containers in terms of a series of features that will be described below:

1) Capacity

That is the amount of embedded tasks/elements inside a container. (Also it is possible to understand it as the number of children nodes under a parent node i.e. the cardinality). In order to describe the capacity we have to specify two conditions:

First, the case of (1) which involves the use of Bell numbers (the conditions for this are the existence of concurrent operators in inner nodes).This could be described in a more formal way as:

$$P \subseteq S(n,k) \left| (1 < k < n) \wedge \left| p_i \right| > 1 \right. \qquad (3)$$

$$Cap(n,k) = \begin{cases} 1 & \leftarrow \quad S(n,k) \wedge k = n \\ P & \leftarrow \quad 1 < k < n \\ n & \leftarrow \quad S(n,k) \wedge k = 1 \end{cases} \qquad (4)$$

Let ***P*** denote the set of all possible partitions (3) where p_i indicates a specific partition, here it is exploited its cardinality to exclude sets with one elements. Let ***n*** denote the number of nodes in a set. Let ***k*** denote the number of non-empty subsets that are possible to deliver over the partition of a set of ***n*** elements. Finally, let ***S*** denote Stirling numbers of the second kind (i.e. the equation to calculate these partitions).

Second, the other alternatives of (1) are correlated with the cardinality of the children sets (for instance, with four inner nodes connected by sequential operators the capacity of each container is one).

2) Adjacent Disposition

This feature is the relation of adjacency between tasks. Since all the children tasks of each sub-tree could be ordered as a sequence from left to right (this disposition does not affect the real attributes of the tree, because we are only dealing with the graphical representation). Then, two categories are possible:

- ***Contiguous***. The children tasks are located in an order that follows the natural numbers (**N**).
- ***Non-contiguous***. This disposition breaks the natural order also depending on the process of generation of containers could relate geometrically distant tasks of the task tree.

Based on these features is possible to imply some heuristic rules over the container structure again with the information provided by the operators; the first one,

HR1: Sequential operators could be associated with contiguous configurations since they are obliged to follow **N** (See Fig. 15). Then all alternative configurations are valid and it could be possible to reduce the number of containers. Another interesting consideration is that, every container of capacity equal to one is contiguous therefore; the order should be respect over container sets.

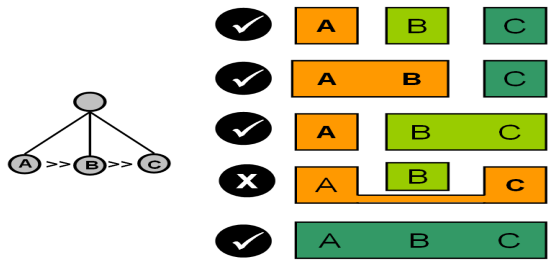

Fig. 15: Cost of each container in the showResults VC.

HR2: the discontinuous configuration should be associated

to choice or concurrent operators. Now, we are going to present the result over one part of the case study.

The application of the rules in different ways could produce different scenarios. For instance, in Fig. 14 containers five, six and seven could be merged according to the **HR1** into configurations which respect the rule. The results from the application of (1) are fixed but though **HR1** and **HR2** they could be updated since our goal is the delivery of plausible containment scenarios instead of optimal ones. Also, we are not dealing with internal spatial location of the widgets (this issue is outside the scope of this paper).

3) The generation of the AUI

In the previous step the second configuration has been presented to the designer. The process of transformation from task model to AUI is out of the scope of this paper since it's discussed in [2] and [13]. The final process would deliver the AUI depicted in Fig. 16 for the container **C2**. The method at this point would provide us with two AUIs that should be connected with the introduction of 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. 6 section b. There, the task **ShowResults** is a sub task of an upper tree besides it's the root node of other container. Then, the approach taken to resolve this situation is the introduction in the upper container a navigation component pointing to the other container.

Fig. 16. Conversion to AUI of the container C2.

At last, as an example of the final result (see Fig. 17), we include a possible FUI of the Container that was labeled as **C2** (Note: this step involves the transformation of the CUI definition to a FUI one through the information available in table 3 which shows a fragment of the mapping process between CUI to FUI elements [2], [13]).

Possible mappings	**Widget**	
Type of container	CUI: UsiXML	FUI : GWT widgets
Top level	Window, dialogbox, flowbox,gridbox,borderbox	AbsolutePanel, DialogBox, DockPanel, FlowPanel, HorizontalPanel, VerticalPanel, TabPanel, DialogBox,FlexTable
Low level	groupbox, Box,table,	FormPanel, Frame, Grid, TextArea, TextBox

Table 3: Mapping between CUI and FUI elements.

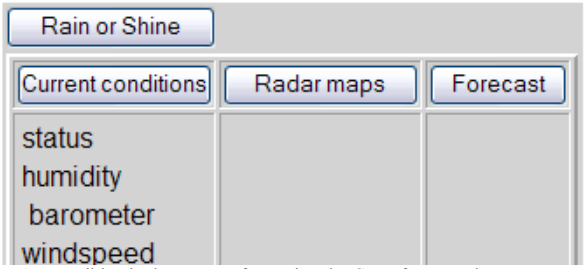

Fig. 17: Possible Final User Interface using the GWT framework.

C. The selection of the UI Complexity metrics

After the process of selecting a configuration that will be used to update the CUI model in terms of containment structure. We arrive to this point where it's possible to leave the semantic level and deal with layout constraints. In order to understand better the process behind the layout generation and UI complexity we are going to compare the different metrics used in some of the algorithms described above. We are going to review four metrics: μ1 (based on amount of items), μ2 (based on weight of items), μ3 (based on widget surfaces) and μ4 (based on cognitive load).

1) μ1 = Based on amount of items

This is the simplest approach to the problem and in fact, it's a simplification of the actual scenario of the UI. The process is straightforward: first we have to define a max capacity of our container and then the amount of items from a UI is counted. If the maximum is reached then another container is instantiated [34].

2) μ2 = Based on weight of items

This approach to the problem requires the definition of an attribute weight to evaluate in a better way the widgets I each UI. We are going to associate the purpose of the elements of UI according to five roles: input elements, output elements,

navigation elements or control elements [13] also we added the number of boxes to take into account the general geometrical structure. Each element has a specific value (see table 4) that is used to calculate the weight. The selection of the weight is based on heuristics about the complexity of the elements.

Role of element	Weight
Navigation (N)	5
Output (O)	1
Boxes (B)	10
Input (I)	3
Control (C)	5

Table 4: Weight values of the elements of the UI.

3) μ3 = Based on widget surfaces

This approach takes into account the visual design of the UI and the spatial constraints of the system. It's deeply discussed in [13] and [14]. Here we use bins of the same height and width (20 pixels) in order to simplify the counting process.

4) μ4 = Based on cognitive load

This approach uses the action analysis ideas from [27] to understand the processes behind the actions of the user in order to accomplish a task through a UI. Each element has a specific value that is used to calculate the weight. We have used the values proposed in [26] where it's defined a Perceive Constant (PC) of 100ms; the cost of understanding a widget is 230ms and 340ms for written text.

5) Description of the used examples

In order to test these metrics we have selected a group of six UIs: first, a login process, then a RSS reader, a shopping cart, an address book, a movie catalog, and finally a reservation system (see Fig. 18). All of them created with the Google Web Toolkit (GWT). The normalized results are shown in table 4.

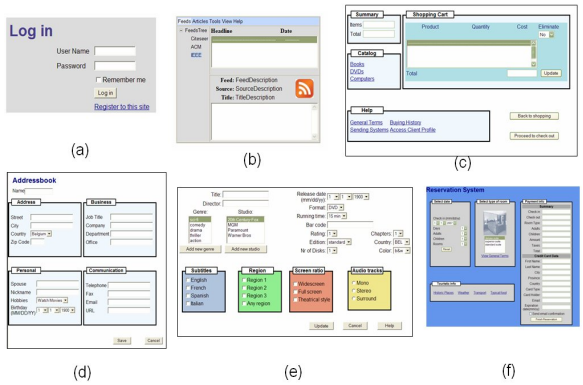

Fig. 18: The generated UIs with the GWT as examples.

#	M1	M2	M3	M4
1	8	6.4	0.98	10.48
2	14	7.6	3.23	17.27
3	26	19	17.8	27.67
4	43	25	17.7	45.79
5	53	34	10.1	62.33
6	63	35	32.8	66.91

Table 4: Weight values of the elements of the UI.

The result of this could be seen in this following graphic (Fig. 19). Note: each line correspond to each metric but in a normalized presentation. It is interesting to notice that even with more sophisticated metrics the tendency was very similar and we did not gain more information. Then we should infer that in the construction of containers is not necessary to involve complex metrics.

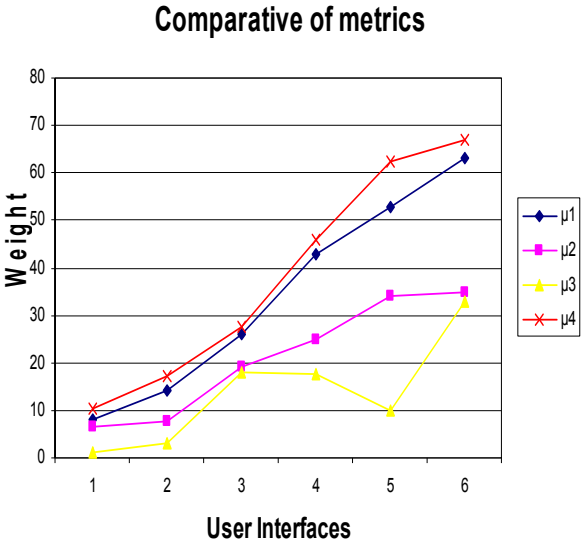

Fig.19: The result of the different metrics.

This is especially useful for the web applications (as the UIs generated with GWT that we have presented) because time would be used in other issues not in an expensive calculation not acceptable for the latency of the mobile domain.

D. Inclusion of Container Patterns

The deployment of the CUI into a specific FUI for mobile devices requires more knowledge of the condition of these devices. And table 5 shows some of the possible mappings to deliver the FUI.

Device	Container Patterns					
	1	2	3	4	5	6
A	X					
B	X		X		X	
C	X	X	X		X	
D	X	X	X	X	X	X

Table 5: Mapping between devices/patterns.

In this step is possible to introduce some container patterns (see Figs. 20 and 21) that are proposed. This classification of container patterns could be applied depending on the device's features (see table 5). With this information is possible to refine even more the containment structure derived until now (The list of devices correspond to those presented in Fig. 6 from left to right are labeled: A,B,C and D in table 5).

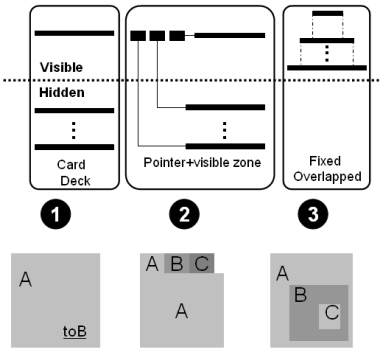

Fig. 20: Proposed Container patterns.

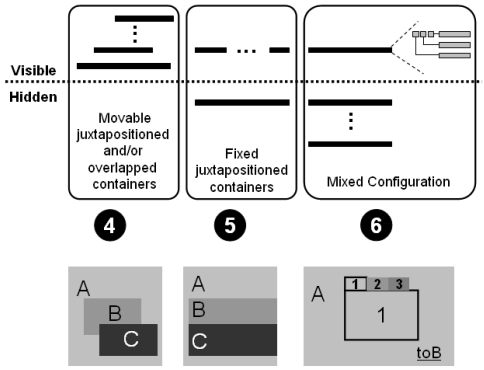

Fig. 21: Proposed Container pattern (continuation).

For instance, in the case study with four layers and taking as the delivering platform a PDA (table 5, last row: device D) It could used the full gamma of patterns. In Fig. 17, the selected pattern is first one. Nevertheless, if the device has enough space, the pattern number forth is a more interesting option.

V. Conclusion and Directions for Future work

In this paper, it has been described an iterative method to help UI designers in the process of defining containment structures in the mobile domain. This method takes into account contextual information (the platform, temporal relationships between tasks and vicinity of the tasks). The starting point is a task tree which is used as the skeleton base in order to support a structure of containers. Furthermore, the utilization of task trees delivers a neutral definition that allows the method to stay platform independent. The designs could be refined multiple times (different scenarios) in order to explore new configurations. For this purpose a set of metrics has been proposed and reviewed and the results obtained suggest that some of them even if they seem more complex and fine-grained do not compulsorily ameliorate the process of finding the UI structure as consequence it's better to use simple metrics in order to fulfill a threshold to construct the containers. Finally, A fundamental consideration is the fact that our method for the moment is not looking optimization as in [5] and [6] instead of that, it provides plausible scenarios to the designer.

A. Future work

A main issue is the refinement of the technique for selecting weights (now, this process is done based on empirical notions). Also, we are working in a better system of pruning low quality configurations in order to show the designer the most acceptable ones. This last task implies the development of an application to help the designer with this task.

REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] 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).

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

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

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

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

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

[11] Florins, M., Simarro, F. M., Vanderdonckt, J., and Michotte, B. 2006. Splitting rules for graceful degradation of user interfaces. In Proceedings of the Working Conference on Advanced Visual interfaces (Venezia, Italy, May 23 - 26, 2006). AVI '06. ACM Press, New York, NY, 59-66.

[12] 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 (Valenciennes, 15-17 mai 2002), Kluwe Academics Pub., Dordrecht, 2002, pp. 103-110.

[13] 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 Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems* EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 207-228.

[14] 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.

[15] 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.

[16] Hürst, W., Lauer, T., and Nold, E. 2007. A study of algorithm animations on mobile devices. In Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (Covington, Kentucky, USA, March 07 - 11, 2007). SIGCSE '07. ACM, New York, NY, 160-164.

[17] Zhao, D., Grundy, J., & Hosking, J. (2006). Generating mobile device user interfaces for diagram-based model-ling tools. Proceedings of the 7th Australasian User in-terface conference - Volume 50, 101-108.

[18] Bickmore, T., and Schilit, B, Digestor: Device-Independent Access to the World Wide Web. In Proc. Seventh Intl. WWW Conference 1997, pp. 655–663.

[19] Bonifati, A., Ceri, S., Fraternali, P., Maurino, A.(2000) : Building multi-device, content-centric applications using WebML and the W3I3 Tool Suite, Proc. Conceptual Modelling for E-Business and the Web, LNCS 1921, pp. 64-75.

[20] Chen, Y., Ma, W.Y., and Zhang, H.J. (2003): Detecting Webpage Structure for Adaptive Viewing on Small Form Factor Devices. In Proc. WWW'03, 20-24 May 2003 Budapest, Hungary, pp 225–233.

[21] Eisenstein, J. and Puerta, A. (2000): Adaptation in automated user-interface design, Proc. 2000 Conference on Intelligent User Interfaces, New Orleans, 9-12 January 2000, ACM Press, pp. 74-81.

[22] Baudisch1, P., Xie, X., Wang C., and Ma, W.Y. (2004): Collapse-to-Zoom: Viewing Web Pages on Small Screen Devices by Interactively Removing Irrelevant Content. In Proc.UIST '04.

[23] Palm Corp. (2001): Web Clipping services, www.palm.com, 2001.

[24] Grundy, J.C. and Zhou, W. (2003): Building multidevice, adaptive thin-client web user interfaces with Extended Java Server Pages, In Cross-platform and Multi-device User Interfaces, Wiley, 2003.

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

[26] Montero, F., López-Jaquero, V., Lozano, M., González, P., IdealXML: un entorno para la gestión de experiencia relacionada con el desarrollo hipermedial, in *ADACO: Ingeniería de la usabilidad en nuevos paradigmas aplicados a entornos web colaborativos y adaptativos*, Proyecto Cicyt TEN2004-08000-C03-03, Taller celebrado en Granada, September 2005.

[27] J.R.Olson, G. Olson. The Growth of Cognitive Modeling in Human-Computer Interaction since GOMS. Human-Computer Interaction, 1990,Volume 5, pp. 221-265, Lawrence Erlbaum, 1990.

[28] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. 1999. Towards a Better Understanding of Context and Context-Awareness. In Proceedings of the 1st international Symposium on Handheld and Ubiquitous Computing (Karlsruhe, Germany, September 27 - 29, 1999). H. Gellersen, Ed. Lecture Notes In Computer Science, vol. 1707. Springer-Verlag, London, 304-307.