# Integrating UIML, Task and Dialogs with Layout Patterns for Multi-Device User Interface Design

*Kris Luyten*[†], *Karin Coninx*[†] *and Marc Abrams*[‡]

[†] Expertise Center for Digital Media
Limburgs Universitair Centrum
Wetenschapspark 2, Diepenbeek, Belgium
{kris.luyten,karin.coninx}@luc.ac.be

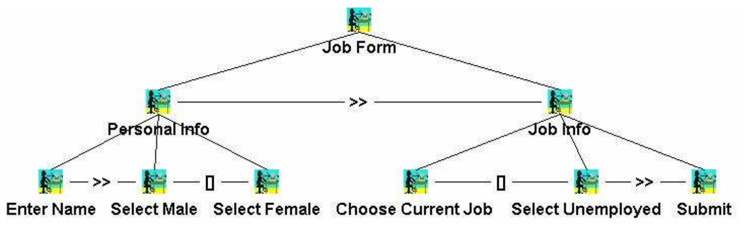
[‡] Harmonia, Inc.
Virginia Tech Corporate, Research Center
1715 Pratt Drive Suite 2820 - PO Box 11282
Blacksburg VA, 24062, USA
mabrams@harmonia.com

## Abstract

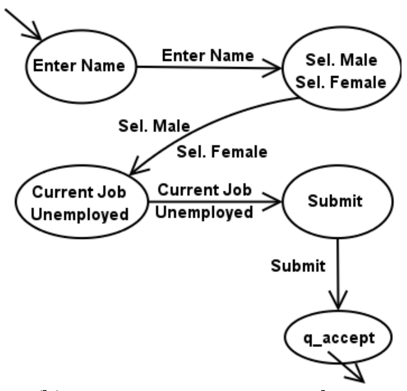
HCI Patterns represent design knowledge that can be reapplied in different situations where the same type of tasks has to be represented. This can be done easily if the devices that support these tasks do not differ in input and output capabilities. A pattern described with the traditional Alexandrian notation does not contain sufficient information "as is" to be flexible enough to support multi-device user interface design and automatically adapt according to the context of use. On the other hand, a degree of plasticity for a pattern can be defined that determines the varying contexts where the pattern can be applied. We show how a certain type of interaction patterns, *layout patterns*, can be combined with a model-based interface development process to support multi-device user interface design. The suitability of a certain layout pattern can be automatically determined by combining information contained in the various models and the description of the pattern.

## 1 Introduction

XML-based User Interface Description Languages (UIDLs) have become very popular since a few years now, mainly driven by the emerging requirement of supporting multi-device user interface (UI) design and development. Most solutions that can be found try to support multi-device UI design by introducing a high level XML-compliant language to describe the UI in terms of Abstract Interaction Objects (AIOs) (Vanderdonckt and Bodart, 1993). While there are many advantages to this approach there are two major drawbacks. The first drawback is the *lack of expressiveness* of most UIDLs to make use of different modalities or take full advantage of a target widget set. A modality refers to the type of communication channel used to communicate with the user and is reflected in the presentation of the user interface: speech and graphical are two familiar modalities. Most UIDLs define a set of abstractions that can be used to describe the UI; this limits the expressivity of the language to this particular predefined set. The second drawback is the *complexity* of the process that transforms the abstract UI description into a concrete working UI for a particular platform. Both problems can be tackled by using a task-centered design approach combined with a powerful UIDL. Task-centered design is an approach that supports a clear abstraction of the interactive system without omitting the significant details that matter in UI design. In our approach the ConcurTaskTrees (CTT) notation is used; a graphical notation for task specification that supports concurrent tasks introduced by Fabio Paternò (Paternò, 2000). Figure 1(a) shows task specification specified with the CTT notation.



(a) ConcurTaskTrees diagram



(b) State Transition Network

Figure 1: A task (1(a)) and a dialog (1(b)) specification.

The interactive tasks in a task specification can be related with user interface building blocks: reusable UI descriptions that specify the interface for a particular task in an abstract way. We use the User Interface Markup Language (UIML) (Abrams et al., 1999; Abrams and Helms, 2004) to describe the UI because UIML is not limited to any modality or widget set in any way and offers us the desired level of abstraction. UIML offers a way to define custom abstractions by means of user-defined vocabularies. A mechanism is provided in this language to use AIOs to describe the interface and define a set of mappings (vocabularies) to convert the AIOs into a Concrete Interaction Objects (CIO). The same interface structure can be reused with different presentation styles and mappings, which is essential for multi-device UI development.

The remainder of this paper is structured as follows: section 2 gives an overview of the related work. Next, in section 3 we show how layout patterns and a model-based interface development process can be combined. Section 4 provides more insight in layout patterns and their properties. Finally, section 5 concludes the paper with the contributions presented here.

## 2 Related Work

Building multi-device user interfaces can be considered one of the main purposes of UIML and has been covered in various publications. For example, the solution that is presented in (Ali et al., 2004) is based on a *generic vocabulary*: a set of predefined abstractions that can be used across different GUI toolkits and devices. This requires an extra processing stage where the interface can be specialized for each device. This can happen either manually or using templates that are related to particular (families of) devices. The approach here differs with respect to the generic vocabulary by promoting integrating between task, context-of-use and presentation.

The most common approach to support multi-device user interface development is a model-based interface design (MBID) approach where the task model often is the most prominent model. The Teresa environment (Mori et al., 2003) is an example of a tool that supports this approach where task, presentation and context are combined to generate multi-device user interfaces. Teresa is based on an XML-based UIDL to describe AIOs; this language is limited to a predefined set of abstractions (Berti et al., 2004). The process presented here has some similarities with the process that is used in Teresa to generate a user interface for multiple devices.

The third and last approach that is valuable for this work is pattern-driven interface design. Although there is no agreement on the definition of an HCI pattern, the term is commonly considered as an artifact that captures reusable design knowledge suitable for a particular context-of-use. We consider interaction patterns as defined by Welie and Trætteberg (Welie and Trætteberg, 2000) to be an appropriate representation of what a HCI pattern that captures design knowledge exactly should represent. In their definition interaction patterns are task-related, categorized according to the usage problem they address and characterized by the following elements: problem description, usability principles, context of use, solution, rationale and examples. Javahery et al. have shown that patterns can be used for multi-device user interface development and that the same patterns can be reused or transformed across several devices (Javahery et al., 2004).

## 3 Task-Driven Presentation Selection

This section shows how UIML can be used as a *presentation specification* in the Dygimes framework and run-time environment (Coninx et al., 2003). Dygimes is a model-based environment that has been developed for multi-device UI creation. With the approach presented here, we provide UIML with a dialog and task model without changing or extending the UIML language itself. While some other approaches try to integrate this kind of models in the UIML specification itself, this approach does not require extending UIML itself and will constrain UIML to the description of single dialogs.

## 3.1 Integration with the task specification

The starting point in this approach is the creation of a task specification with the CTT notation (e.g. figure 1(a)). A leaf task from a hierarchical task description is visualized by a *user interface building block*, which is a presentation expressed in UIML. This user interface building block contains an abstract description of a part of the user interface that can be considered as a "unit": it would not be meaningful to subdivide this user interface unit any further in subparts. Notice the granularity of a building block is directly related to the way the interface designer thinks about the tasks the interface should support. In reality an interface designer often prefers to design an interface for each device to obtain the most "pleasing" results: the task-based approach can support this and still optimize reusability. First, tasks can be constrained to a certain families of devices (e.g. PDA and desktop, but not cell-phone): when generating the interface for a particular target device the tasks that do not apply can be filtered out (thus omitting the related building blocks). Second, a task specification can contain decision nodes (Clerckx et al., 2004): a decision node is just another type of node besides the existing CTT node types that provides a selection mechanism inside the task specification.  Decision nodes allow selecting an appropriate sub-tree in the task specification for a given context-of-use.

Let us reconsider the simple task specification in figure 1(a). A user interface building block for each leaf task that should be visualized is created to obtain a complete user interface for the task specification. This gives us the set of task-related user interface building blocks as shown in table 1. For each task there is a building block represented by a UIML document that visualizes the task. For clarity, only the structure fragment of the various building blocks is shown in table 1.

## 3.2 Generation of the dialog model

The set of dialogs necessary to have a full coverage of the user interface, for all possible tasks, are generated automatically from the task specification.  Full dialog coverage indicates that every task from a task specification is reachable by interacting with a sequence of dialogs. This is accomplished by the algorithm introduced in (Luyten et al., 2003). Since a dialog is actually a mapping from a set of leaf tasks onto a presentation unit, the building blocks attached to these tasks exactly make up the presentation unit.

Next, using the terminology of (Paternò, 2000) the following enabled task sets (ETS) for the task specification in figure 1(a) can be identified: $ETS_1$ *= {Enter Name}*, $ETS_2$ *= {Select Male, Select Female}*, $ETS_3$ *= {Choose Current Job, Select Unemployed}* and $ETS_4$ *= {Submit}*. An enabled task sets specifies which tasks will be active during the same period of time; exactly these tasks should be presented to the user all at once. We have to provide four different dialogs to obtain complete dialog coverage for this task specification (one for each ETS). Take $ETS_3$ for example: it has two tasks, which should be visualized in the same period of time. Merging the user interface building blocks for these tasks is a matter of selecting an appropriate layout pattern that serves as a container. Adding the two building blocks in this container makes up a new UIML document that can be rendered as the final interface. Table 2 shows two very simple layout patterns for illustration purposes: the left container just uses a horizontal layout and the container on the right uses tabbed pages. In the example the UIML building block of the task "Choose Current Job" is inserted first, and the task "Select Unemployed" is inserted second in the container. The UIML properties are not shown in the example because merging them can be straightforward: the list of properties of the building blocks can be concatenated as can the behavior rules. A more sophisticated merge could be provided by making use of certain elements provided by UIML like the "source" attribute or "restructure" tag. Containers are simple layout patterns that could be pre-generated for the designer and customized afterward. In this example we made the completion of the enabled task set $ETS_3$ explicit in the container by adding a part "confirm" that maps to a button to indicate the information for this dialog is complete.

This approach can be extended for context-sensitive user interfaces: e.g. the task specification can take into account the different constraints that occur when other interaction devices are used to complete the task(s). In (Clerckx et al., 2004) we showed a simple but effective realization of the integration of context information in the task specification. Context is represented by decision nodes in the task specification: a decision node is just another type of node besides the existing CTT node types that provides a selection mechanism inside the task specification. This can easily be extended to select a layout pattern amongst the available patterns according to the context-of-use the task(s) will be executed. This results in a way to specify which layout pattern is more suitable in a particular context by the designer. Figure 2 shows an example of this solution.

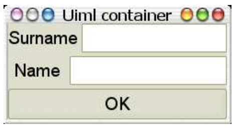
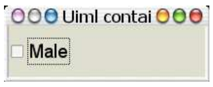
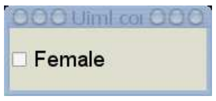
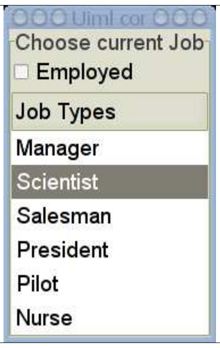
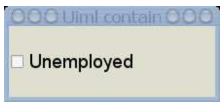
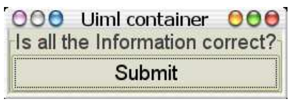
| *Task* | *UIML structure* | *UI building block* |
|---|---|---|
| Enter Name | `<part class="VBox"> <part class="HBox"> <part class="Label" id="lsurname"/> <part class="Entry" id="surname"/> </part> <part class="HBox"> <part class="Label" id="lname"/> <part class="Entry" id="name"/> </part> <part class="Button" id="ok"/> </part>` |  |
| Select Male | `<part class="HBox"> <part class="CheckButton" id="male"/> </part>` |  |
| Select Female | `<part class="HBox"> <part class="CheckButton" id="female"/> </part>` |  |
| Choose Current Job | `<part id="Top" class="Frame"> <part class="VBox"> <part class="CheckButton" id="lcurrent"/> <part id="currentjob" class="List"/> </part> </part>` |   Uiml cor Choose current Job Employed Job Types Manager Scientist Salesman President Pilot Nurse |
| Select Unemployed | `<part class="HBox"> <part class="CheckButton" id="unemployed"/> </part>` |  |
| Submit | `<part id="stop" class="Frame"> <part id="submit" class="Button"/> </part>` |  |

Table 1: User Interface building blocks for each leaf task from figure 1(a)

## 4 Layout Patterns: A GUI Floor Plan

A layout pattern is probably one of the simplest patterns: it can be compared to a floor plan of a building. Such a floor plan describes where the different types of rooms are located in the building, often spread over several floors. The same concepts can be found in a layout pattern: the representations of different tasks are allocated to the different available spaces in the pattern. The main difference is the plasticity of a layout pattern with respect to a floor plan: a layout pattern is reusable for a certain ranges of surface areas. Figure 2 shows a visual representation of a layout pattern as a simple building plan. Notice there are two “floors”, e.g. the two floors can be mapped on tabs in the final interface, like shown in column 2 of table 2. Layout patterns are proven solutions that can be applied easily for an application. In contrast with (Javahery et al., 2004) where XUL templates are proposed to express the layout of a GUI, spatial constraints can offer more plasticity since the layout will be suitable as long as the constraints are satisfied. Spatial constraints are a simple yet effective way for describing the layout of an interface by specifying the spatial relations (left-of, right-of, above, under, next-level, previous-level) between different rooms in the pattern. Furthermore the minimal and maximal width and height can be specified for each room, which gives us a mean to calculate whether the chosen pattern is “plastic enough” for a given screen space.

Once a pattern is chosen, we need to know which “room” in the pattern should be used for each task. This can be specified by the designer in the dialog model but a manual approach can not cover all possibilities. Since we started from the CTT notation, several concepts that are used in Teresa’s abstract user interface language (Berti et al., 2004) can be reused here. E.g. a place for a building block in the layout pattern can be reserved according to the interaction or output type of the task that is related with the building block. The possible interaction and output types are selection, control, editing, text, feedback… Figure 2 shows a visual representation of a layout pattern including indications of which type of building block belongs where.

Finally, there is an opportunity to integrate this approach with pattern repositories such as MOUDIL (Gaffar et al., 2003) given the repository has some predefined schema that can be queried in a structured and well-defined way. According to the context-of-use (in our case defined by the tasks that need to be presented together with the interaction device that is used) an appropriate pattern can be chosen to visualize a set of tasks.

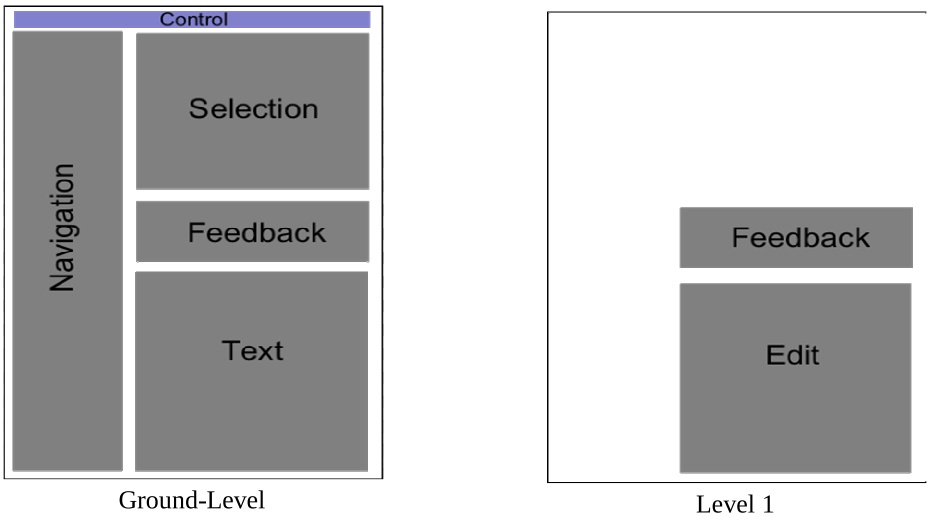


Figure 2: Visual Representation of a Layout Pattern

## 5 Conclusions

We presented a way to integrate layout patterns with task modeling to obtain multi-device user interfaces. This integration takes into account the different types of patterns that occur in the design of a user interface: task patterns, layout patterns, interaction patterns… The required flexibility to support multi-device user interface design and development can not be obtained from layout patterns itself, because they are directly related to screen space usage, but can be obtained by combining layout patterns with the other design artifacts that are suitable for multi-device user interface design. This paper defines a simple process to support task-based design of an interactive system while using the full power of the XML-based UI description language UIML. A common problem arising with the (semi-)automatic generation of UI dialogs, namely the inability to influence the organization of the final presentation, is tackled by container templates that represent the layout patterns. Most importantly; this process can be used with different XML-based UI description languages depending on the expressiveness that is required to target a particular widget set.

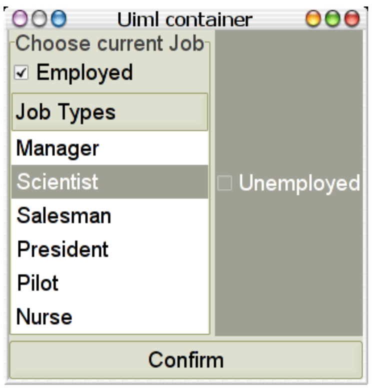
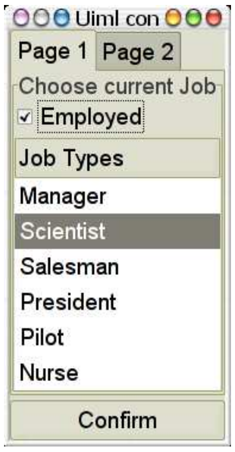
| Vertical Container | Tabbed Container |
|---|---|
| <part class="VBox"><br>    <part class="HBox"><br>        <!-- insert UIML structure<br>                first task here --><br>        <!-- insert UIML structure<br>                second task here --><br>    </part><br>    <part class="Button" id="confirm"/><br></part> | <part class="VBox"><br>    <part id="tabs" class="Tabs"><br>        <part id="Tab1" class="TabPage"><br>            <!-- insert UIML structure<br>                    first task here --><br>        </part><br>        <part id="Tab2" class="TabPage"><br>            <!-- insert UIML structure<br>                    second task here --><br>          </part><br>    <part class="Button" id="confirm"/><br></part> |
|  Uiml container<br>Choose current Job<br>Employed<br>Job Types<br>Manager<br>Scientist<br>Salesman<br>President<br>Pilot<br>Nurse<br>Unemployed<br>Confirm |  Uiml con<br>Page 1<br>Page 2<br>Choose current Job<br>Employed<br>Job Types<br>Manager<br>Scientist<br>Salesman<br>President<br>Pilot<br>Nurse<br>Confirm |

Table 2: Two examples of merging UIML building blocks from an enabled task set with a predefined container.

## Acknowledgments

The research at the Expertise Center for Digital Media (EDM/LUC) is partly funded by the Flemish government and EFRO (European Fund for Regional Development). Marc Abrams is supported by Small Business Innovative Research contract N68335-05-C-0029 from the U.S. Navy.

## References

Abrams, M. and Helms, J. (2004). Retrospective on UI Description Languages, Based on 7 years Experience with the User Interface Markup Language (UIML). In (Luyten et al., 2004), pages 1–8.

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E. (1999). UIML: An Appliance-Independent XML User Interface Language. *WWW8 / Computer Networks*, 31(11-16):1695–1708.

Ali, M. F., P´erez-Qui˜ones, M. A., and Abrams, M. (2004). *Building Multi-Platform User Interfaces with UIML*, pages 95–116. In (Seffah and Javahery, 2004).

Berti, S., Correanim, F., Patern`o, F., and Santoro, C. (2004). The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels. In (Luyten et al., 2004), pages 103–110.

Clerckx, T., Luyten, K., and Coninx, K. (2004). Generating Context-Sensitive Multiple Device Interfaces from Design. In Limbourg, Q., Jacob, R., and Vanderdonckt, J., editors, *CADUI 2004*, volume 4. Kluwer Academic.

Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J., and Creemers, B. (2003). Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In Chittaro, L., editor, *Mobile HCI*, volume 2795 of *Lecture Notes in Computer Science*, pages 256–270. Springer.

Gaffar, A., Javahery, H., Sinnig, D., and Seffah, A. (2003). MOUDIL: A Comprehensive Framework for Disseminating and Sharing HCI Patterns. In *CHI'03 Workshop on Perspectives on HCI patterns: Concepts and Tools*. `http://hci.cs. concordia.ca/moudil/`.

Javahery, H., Seffah, A., Engelberg, D., and Sinnig, D. (2004). *Migrating User Interfaces Across Platforms Using HCI Patterns*, pages 241–259. In (Seffah and Javahery, 2004).

Luyten, K., Abrams, M., Limbourg, Q., and Vanderdonckt, J., editors (2004). *Developing User Interfaces with XML: Advances on User Interface Description Languages*. Sattelite workshop of Advanced Visual Interfaces (AVI) 2004, Expertise Centre for Digital Media.

Luyten, K., Clerckx, T., Coninx, K., and Vanderdonckt, J. (2003). Derivation of a Dialog Model for a Task Model by Activity Chain Extraction. In Jorge, J. A., Nunes, N. J., and Falcão e Cunha, J., editors, *DSV-IS*, volume 2844 of *Lecture Notes in Computer Science*, pages 203–217. Springer.

Mori, G., Paternò, F., and Santoro, C. (2003). Tool Support for Designing Nomadic Applications. In *Proceedings of the 2003 international conference on Intelligent user interfaces*, pages 141–148, Miami, Florida, USA.

Paternò, F. (2000). *Model-Based Design and Evaluation of Interactive Applications*. Springer.

Seffah, A. and Javahery, H., editors (2004). *Multiple User Interfaces – Cross-platform Applications and Context-aware Interfaces*. Wiley.

Vanderdonckt, J. and Bodart, F. (1993). Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *ACM Conference on Human Aspects in Computing Systems InterCHI'93*, pages 424–429. Addison Wesley.

Welie, M. V. and Trætteberg, H. (2000). Interaction Patterns in User Interfaces. In *Proceedings of the 7th Pattern Languages of Programs Conference*.