Multi-device Layout Management for Mobile Computing Devices

Kris Luyten Bert Creemers Karin Coninx

Technical Report TR-LUC-EDM-0301
Expertise Centre for Digital Media
Limburgs Universitair Centrum
Wetenschapspark 2
B-3590 Diepenbeek
{kris.luyten, bert.creemers, karin.coninx}@luc.ac.be

Abstract

As the computing market is evolving toward mobile computing systems and embedded or ubiquitous systems, developing user interfaces becomes a tedious job. Taking into account all possible combinations of device constraints has become rather complex, and UIs are still developed toward specific devices. This paper discusses some techniques for multi-device interface development. In particular we describe how abstract UI descriptions and constraint-based layout management systems can be combined for developing adaptive UIs for a wide range of devices. We believe this kind of techniques will gain importance when the development techniques for Mobile User Interfaces become more mature.

1 Introduction

The growing diversity of programmable devices on the market raises new demands for user interface designers. Whereas they traditionally could focus on one user interface for a particular platform, focus has shifted on making multi-device user interfaces. The Internet is the most well-known example of this evolution: the old HTML pages which were only suitable for desktop web-browsers are evolving toward dynamically generated pages which are adapted to the target browser. The most well-known example of this practice is the combination of the eXtensible Markup Language (XML)(consortium 2001*a*) in combination with eXtensible Stylesheet Language Transformation (XSLT) documents (consortium 2001*b*). While the XML document describes the content, the platform and/or device specific transformation into a suitable presentation is done by the provided XSLT documents.

The fact that different XSLT documents should be provided for each kind of transformation before the content can be deployed is an important drawback. Although this approach can be made generic, it does not anticipate unknown target devices and platforms. When new devices have restrictions which were not known when the transformation was defined, it is possible that the content cannot be delivered to the newly introduced target in the intended manner.

This paper introduces parts of a framework for making multi-device user interfaces. It describes how a constraint-based layout management system enables the GUI designer to deploy its interface to a wide range of devices. A layout management system takes care of positioning components of the GUI. While the variety of mobile computing systems grows, the development techniques for providing UIs targeted at these systems are maturing slowly. Because of the many available programming languages and UI toolkits for creating UIs for mobile systems, there is a need to design the UI independent of these choices. On the other hand the approach should be practical

to reduce the time-to-market.

The next section gives an overview of the related work. Continuing with section 3 we explain how high-level, abstract UI descriptions can be used to build UIs for mobile computing devices. Section 4 will discuss the usage of constraint solving and logical grouping in the presented layout manager. Details about the actual layout adaption process taking into account available screen space are explained in section 5. Next, section 6 describes how the same technique can be applied when the UI has to be spread over several devices. Finally some conclusions with respect to the current work are given in section 7, followed by an overview of the future work.

2 Related Work

Closely related work can be found in (Olsen, Jefferies, Nielsen, Moyes & Fredrickson 2000). This system provides multi-device interfaces using several techniques to adapt the layout of the UI to the device. This system relies more on the scalability of the widgets (widgets provide a minimum, preferred and maximum size) and less on the logical relations between groups of widgets then the system we present. (Olsen et al. 2000) does provide a protocol and specification for describing the interfaces and allows less intervention of the UI designer for managing the layout. As we will show in section 5.1 the specification of the UI structure and the specification of the layout can be separated, and allows easy manipulation by the UI designer.

Most noticeable are the existing layout managers like those used by the Java AWT and Swing GUI toolkits (Walrath & Campione n.d.) or Gtk (Main & Team 2002). They provide an hierarchical approach, where layouts can be nested in other layouts. This has proven to be intuitive and flexible. For example, Java-based GUIs can already be shown on different devices whilst adapting to their new environment. Unfortunately, this approach lacks flexibility when the constraints of the new environment become more extreme. The GUI does not have the possibility to "regroup" itself in another presentation structure for better presentation while respecting the constraints. Other, more flexible approaches are also under investigation.

(Lok & Feiner 2001) provides a survey of work on automated layout for presentation models. A presentation model is the actual user interface that is generated for the application logic. Most of the research on automated layout management is concentrated on constraint-based layout systems. The characteristics of these systems are summarized in (Lok & Feiner 2001). Two approaches can be identified: spatial layout constraints and abstract constraints. Spatial constraints express the positioning of components with respect to each other. Abstract contstraints express a logical relation between components. For example, "caption A is left of list B" is a spatial constraint and "caption A describes the content of list B" is an abstract constraint. Abstract constraints are always transformed into spatial constraints before they can be used.

(Bodart, Hennebert, Leheureux & Vanderdonckt 1994) and (Vanderdonckt 1995) describe some techniques for automated layout management, more in particular the dynamic right-bottom and the static two-column strategy. These techniques are not constraint-based, as opposed to the approach we present in this paper. In the past, there have been several attempts to create constraint-based layout managers, like interfaces made with Thinglab (Borning 1979, J, A & BN 1989) and the application of the DeltaBlue algorithm in (Sannella, Maloney, Freeman-Benson & Borning 1993). Our approach is partially inspired by the work presented in (Sannella et al. 1993). Two dimensional Graphical UIs are often described with linear constraints. Because constraint satisfaction is a difficult problem to solve, some research was conducted toward efficient algorithms for constraint satisfaction problems using linear constraints (Borning, Marriott, Stuckey & Xiao 1997).

When transporting a GUI to an other device with different constraints, a suitable presentation model has to be generated. (Eisenstein, Vanderdonckt & Puerta 2001) describes a method to generate and select a presentation model for GUIs on mobile computing devices. It also describes the influence of interactor selection on the allocated screen space. Au-

tomatic interactor selection based on selection rules is described in (Vanderdonckt & Bodart 1993). Its purpose is to select an optimal set of Abstract Interaction Objects (AIOs) presenting some functionality for a specific target platform.

3 Multi-device Interfaces with UiBuilder

When GUIs travel from one device to another or existing GUI implementations are reused on other devices, they are subject to certain transformations. Several decisions have to be made about the new presentation model which is used when targeting a new device. New devices from which the constraints are unknown at design-time should also be targetable. This means the migration process has to provide some rules or algorithms deciding about the new presentation structure at runtime.

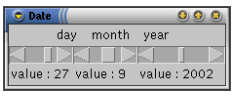
A GUI can be described on a "higher level" using an abstract description language for the GUI. This approach is also known as describing the AIOs and providing mappings on Concrete Interaction Objects (CIOs) (Vanderdonckt & Bodart 1993). Our UI description language uses an XML[1]-based syntax to describe the abstract AIOs. This description can be converted into a device-specific User Interface at runtime using the UiBuilder library developed for this purpose. This library takes UI descriptions as input, together with a target widget set (AWT, HTML or Swing for example), and will render a working User Interface from this description. More information about the system and differences with other similar systems can be found in (Luyten & Coninx 2001, Luyten, Vandervelpen & Coninx 2002). To give the reader an idea of how the system works, a minimal example is given in listing 1, and two possible resulting interfaces are shown in figure 1.

Listing 1: A date group

```
<group name="date">
 <interactor>
  <info name="label">...</info>
 </interactor>
 <interactor>
  <range name="day">...</range>
 </interactor>
 <interactor>
  <range name="month">...</range>
 </interactor>
 <interactor>
  <range name="year">...</range>
 </interactor>
</group>
```

[1] eXtensible Markup Language; (consortium 2001*a*)

(a) In a browser



(b) Java AWT

Figure 1: Two different views on listing 1, both automatically generated

As one can see in listing 1, a *group* tag groups all widgets which logically belong together. At the lowest level, all widgets in a group should always be presented together to the user. The hierarchical structure of the UI description allows to recursively group parts of the UI, i.e. groups can contain other groups, which on their turn can contain other groups themselves...

When developing a UI description language which enables us to render the UI presentation on several different devices, a more flexible approach for laying out concrete widgets than the traditional layout management is necessary. In the past, several approaches for automatic layout management were proposed. Traditional approaches are insufficient to handle the constraints for multi-device layout management. For example, when a graphical UI designed for a desktop system has to be rendered for use on a very small screen space (like on a mobile phone) most techniques fail to present a usable interface.

4 Constraint Satisfaction and Layout Management

As a first step toward constraint-based layout management, we use four simple linear spatial constraints to describe the positioning of the widgets in respect to each other: *left-of*, *right-of*, *above* and *below*. In addition the available space to lay out the widgets is divided in a grid. Each bucket in the grid is uniquely identified by its x and y position within the grid. Notice the linear constraints can be expressed in a mathematical form: take the constraint *widget A left-of widget B* for example. If widget A is put in bucket X with coordinates (x_1, y_1) and widget B is placed in bucket Y with coordinates (x_2, y_2). The constraint can also be expressed as $x_1 < x_2$.

Several traditional layout managers do offer some kind of spatial constraints, like the GridBagLayout of the Java GUI toolkit(Walrath & Campione n.d.) or the "packing" containers provided by the GIMP Toolkit(Main & Team 2002).

However, our approach differs with traditional approaches because we also use the hierarchy as described by the *abstract UI description* (see section 3) instead of directly implementing the hierarchy in the programming code. Constraints are only defined between siblings in the description tree. This is shown in figure 2. The hierarchy divides the interface in groups. These groups can be subdivided in other groups and so on. All widgets part of the same group, have a logical relation with respect to each other. Some rules can be applied here:

- A group describes a set of **logically related** abstract interactors or groups of abstract interactors. The designer should decide which widgets are gathered in a group.
- A group can be specified **splittable**. this specifier allows the layout manager to show the abstract interactors or groups of abstract interactors in separate spaces.
- The group specifier **non-splittable** forces the layout manager to show the children of the group as a whole to make sense to the user. Notice non-splittable is only valid for the direct children of the group, and does not constrain the further offspring.

In contrast with the traditional layout managers, our system does not rely on a particular programming language to produce the GUI. In combination with the high level UI descriptions introduced in section 3, we get a *very loosely-coupled* UI for the application.

5 Calculating Presentation Structures

As mentioned before, only spatial constraints will be employed in our implemented system. The use of only this kind of constraints is sufficient because they directly determine the geometric structure of the layout. Abstract constraints which describe a high-level relation between two components are omitted in our approach because they are always transformed to spatial constraints in a later stage.

5.1 Describing spatial constraints

A simple XML-based syntax is used for describing constraints between two components. A formulation of a constraint network for listing 1 is depicted in listing 2. The first constraint in this network implies that *label* will appear somewhere above *date* in the resulting layout. If the set of constraints is sufficiently large, there is a strong likelihood that conflicts will arise: for example, some constraints may contradict others and possibly make the set of constraints unsolvable. For handling these kind of inconsistencies, priorities are introduced into our system. A priority, represented by an integer value, can be applied to each constraint. Higher values indicate stronger constraints and lower values represent weaker constraints.

Listing 2: A XML constraint description for a UI.

```
<constraints>
  <constraint type="above" priority="5">
    <interactor name="label"/>
    <interactor name="date"/>
  </constraint>
  <constraint type="left" priority="7">
```

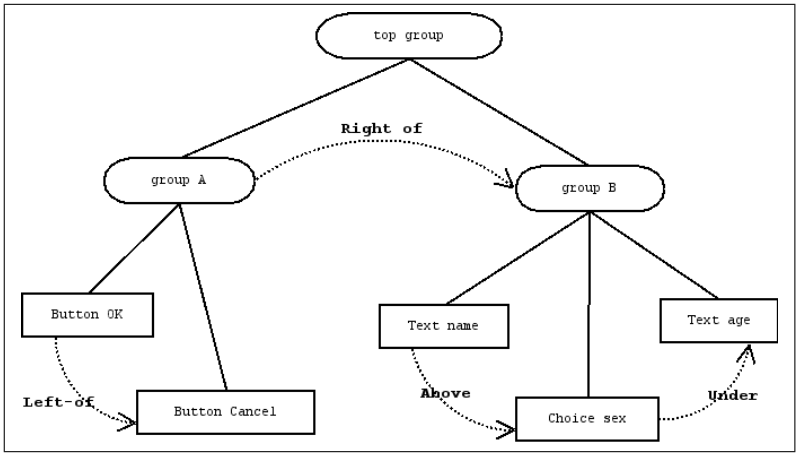


Figure 2: A visual representation of the constraint definition

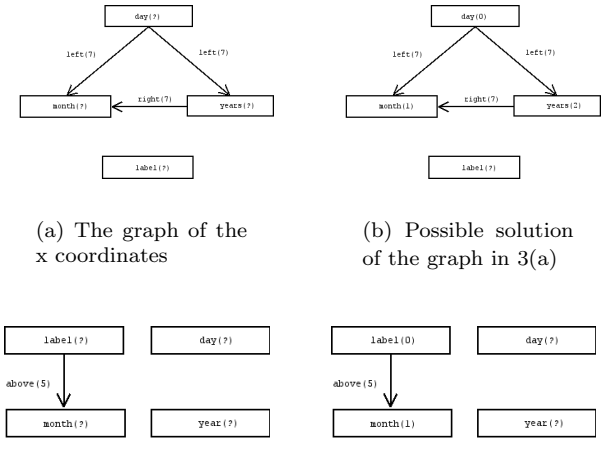
```
    <interactor name="day"/>
    <interactor name="month"/>
  </constraint>
  <constraint type="right" priority="7">
    <interactor name="year"/>
    <interactor name="month"/>
  </constraint>
  <constraint type="left" priority="7">
    <interactor name="day"/>
    <interactor name="year"/>
  </constraint>
</constraints>
```

5.2 Building the layout description graph

Because constraints are only allowed between siblings, each component of the group will be associated by a unique bucket in the grid. Our implemented algorithm will initially solve the x coordinates of the components (coordinates indicate the position in the grid, *not* the absolute coordinates on the screen). A graph will be composed where each node represents a component with his x coordinate and each edge represents a constraint between the two components connected by the edge. Considering the constraints in listing 2, the resulting graph will have an outcome as depicted in figure 3(a). Like mentioned earlier, each edge with the label *left* can be expressed as $x_1 < x_2$. A possible solution for the constraints represented in the graph is depicted in figure 3(b). Single nodes represent components from which the x coordinates are not affected by the constraints. The x coordinates for these components will be determined at a later stage in the algorithm. After the solution has been calculated for the x coordinates, the same strategy will be applied for the y coordinates. The resulting graph and a possible solution for the y coordinates are depicted in figure 3(c) and figure 3(d).

5.3 Calculating widget positions

The results of the two proceeding steps will be combined. This leads to a general solution where for example the component *month* has the coordinates (1,1). The final stage of the algorithm consists of assigning a value to the still unassigned coordinate values. When the component *day* with coordinates (?,0) is considered, the UI designer has the responsibility to locate a free bucket in column 0 to place the component in. '?' reflects that every free place in column 0 can be used. This can be filled in automatically when required or can be chosen by the designer.



(a) The graph of the x coordinates

(b) Possible solution of the graph in 3(a)

(c) The graph of the y coordinates

(d) Possible solution of the graph in 3(c)

Figure 3: The calculation of the presentation structure

5.4 Conflict handling

The occurrence of cycles in the graph implies the the presence of conflicting constraints. Each edge in a cycle represents a constraint that conflicts with another one in the cycle. The constraint with the weakest priority will be removed to break the cycle. If the constraints have equal priority, the first one will be removed and the other will be maintained. The presence of multiple edges between two nodes also implies a conflict. A component can not be placed at the same time at the right and left side of another component. The edge with the highest priority will not be eliminated. All the other edges width lower priorities will be removed.

5.5 Further screen space reduction strategies

After the presentation structure which is defined by the UI designer is calculated, the possibility exists that the layout does not fit on the screen. A layout adapter is imposed to resolve this problem by adapting the presentation structure to the limited screen size of the target platform. One of the strategies employed by the adapter to shrink the presentation consists of reducing the sizes of the textfields. Several similar rules can be applied (e.g. figures can be shrinked). The functionality of the user interface remains intact.

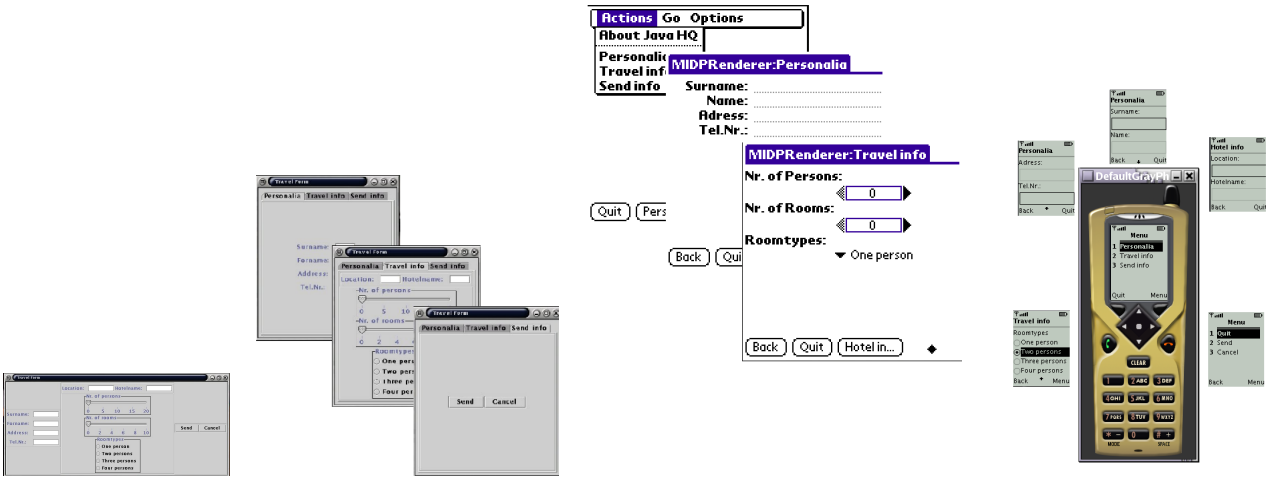
A more powerful strategy to reduce screen space consists of placing the components of a splittable group after each other in a card layout or with tabbed panes, like shown in figure 4. This strategy takes benefit from the hierarchical nature of the UI specification.

It is important to notice that after applying this method the constraints between the children of the group are no longer valid. The adapter applies these strategies iteratively on the presentation structure and after each iteration will be checked if the layout fits the screen. However sometimes it is impossible to shrink the layout to the size of the screen of the target platform. An appropriate warning will be displayed if the UI cannot be rendered.

6 Multiple-device Layout Management

As with “traditional” layout managers, like the ones used in Java AWT and Swing, different layout managers can be used for subparts of the GUI. This is an effective approach when the GUI can be spread over several output devices. Imagine you have two PDA’s or a mobile phone and a PDA. When they become aware of each other, it becomes possible to spread the GUI over those devices.

In section 3 the hierarchical approach of structuring the GUI was explained. As a consequence, when the GUI has to be split up to be presented over several devices, a subtree of the GUI description is cut-



(a) Desktop (b) IPaq 3970 (c) Palm IIIc (d) Cell phone

Figure 4: Hotel registration form

off and this part of the GUI can be assigned to a specific device. This device is responsible for showing its assigned part of the interface. Communication between subtrees (subtrees are part of the complete GUI) takes place using the built-in communication mechanisms, which is based on web-service communication mechanisms, like XML-RPC and SOAP (Vandervelpen, Luyten & Coninx 2003). User actions in one part of the interface can cause updates in other parts of the interface (rendered on the same or another device).

7 Conclusions and Future Work

This paper presented how UI development for mobile computing systems and automatic layout management can be combined. Reusing existing techniques, like constraint-based layout management in combination with high-level UI descriptions proved to be a powerful tool to create reusable and flexible UIs for embedded systems and mobile computing systems. Because of the various different presentations the UI can have, the layout management technique should be able to operate on these different presentations.

We do not believe our approach is an optimal one, but the proposed technique has proven to be usable in various situations. The techniques are only suitable for form-based GUIs which is the case for the majority of contemporary mobile applications. Several interfaces were built successfully using this system. One of these interfaces was a graphical UI for controlling a camera surveillance system.

An important aspect that was not discussed in this paper but which is certainly a topic that deserves our interest, is how we can manage to "spread" the User Interface over several output devices, where a variety of modalities are offered. Especially interesting here is how parts can be rendered as speech and other parts as a GUI.

8 Acknowledgments

Our research is partly funded by the Flemish government and EFRO[2]. The SEESCOA project IWT 980374 is directly funded by the IWT[3].

References

Bodart, F., Hennebert, A.-M., Leheureux, J.-M. & Vanderdonckt, J. (1994), 'Towards a dynamic strategy for computer-aided visual placement'.

Borning, A. (1979), ThingLab – A Constraint-Oriented Simulation Laboratory, Technical report, XEROX PARC. report SSL-79-3.

Borning, A., Marriott, K., Stuckey, P. & Xiao, Y. (1997), Solving Lineair Arithmetic Constraints for User Interface Applications, *in* 'Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-97)'.

consortium, W. W. W. (2001*a*), *Extensible Markup Language (XML)*, World Wide Web, `http://www.w3.org/XML/`.

consortium, W. W. W. (2001*b*), *The Extensible Stylesheet Language (XSL)*, World Wide Web, `http://www.w3.org/Style/XSL/`.

Eisenstein, J., Vanderdonckt, J. & Puerta, A. (2001), Applying Model-Based Techniques to the Development of UIs for Mobile Computers, *in* 'IUI 2001 International Conference on Intelligent User Interfaces', pp. 69–76.

J, M., A, B. & BN, F.-B. (1989), Constraint Technology for User Interface Construction in ThingLab II, *in* 'OOPSLA'.

Lok, S. & Feiner, S. (2001), A Survey of Automated Layout Techniques for Information Presentations, *in* 'Proceedings of SmartGraphics 2001'.

[2]European Fund for Regional Development

[3]Flemish subsidy organization

Luyten, K. & Coninx, K. (2001), An XML-based runtime user interface description language for mobile computing devices, *in* C. Johnson, ed., 'Interactive Systems: Design, Specification, and Verification', Vol. 2220 of *Lecture Notes in Computer Science*, Springer, pp. 17–29.

Luyten, K., Vandervelpen, C. & Coninx, K. (2002), Migratable User Interface Descriptions in Component-Based Development, *in* P. Forbrig, Q. Limbourg, B. Urban & J. Vanderdonckt, eds, 'Interactive Systems: Design, Specification, and Verification', Vol. 2221 of *Lecture Notes in Computer Science*, Springer, pp. 62–76.

Main, I. & Team, T. G. (2002), *GTK+ 2.0 Tutorial*, World Wide Web, `http://www.gtk.org/tutorial`.

Olsen, D. R., Jefferies, S., Nielsen, T., Moyes, W. & Fredrickson, P. (2000), Cross-modal interaction using XWeb, *in* 'Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)', ACM Press, N.Y., pp. 191–200.

Sannella, M., Maloney, J., Freeman-Benson, B. & Borning, A. (1993), 'Multi-way versus One-way Constraints in User Interfaces: Experience with the DeltaBlue Algorithm', *Software - Practice and Experience* **23**(5), 529–566.

Vanderdonckt, J. (1995), Knowledge-Based Systems for Automated User Interface Generation : the TRIDENT Experience, *in* 'Proceedings of the CHI '95 Workshop on Knowledge-Based Support for the User Interface Design Process'.

Vanderdonckt, J. & Bodart, F. (1993), Encapsulating knowledge for intelligent automatic interaction objects selection, *in* 'ACM Conference on Human Aspects in Computing Systems InterCHI'93', Addison Wesley, pp. 424–429.

Vandervelpen, C., Luyten, K. & Coninx, K. (2003), Location Transparant User Interaction for Heterogeneous Environments , *in* 'HCI International'. Accepted for publication.

Walrath, K. & Campione, M. (n.d.), *The Swing Tutorial*, World Wide Web, `http://java.sun.com/docs/books/tutorial/books/swing/index.html`.