Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems

Karin Coninx, Kris Luyten, Chris Vandervelpen, Jan Van den Bergh, and Bert Creemers

Limburgs Universitair Centrum
Expertise Center for Digital Media
Universitaire Campus
B-3590 Diepenbeek, Belgium
http://www.edm.luc.ac.be
{karin.coninx, kris.luyten, chris.vandervelpen, jan.vandenbergh, bert.creemers}@luc.ac.be

Abstract. Constructing multi-device interfaces still presents major challenges, despite all efforts of the industry and several academic initiatives to develop usable solutions. One approach which is finding its way into general use, is XML-based User Interface descriptions to generate suitable User Interfaces for embedded systems and mobile computing devices. Another important solution is Model-based User Interface design, which evolved into a very suitable but academic approach for designing multi-device interfaces. We introduce a framework, Dygimes, which uses XML-based User Interface descriptions in combination with selected models, to generate User Interfaces for different kinds of devices at runtime. With this framework task specifications are combined with XML-based User Interface building blocks to generate User Interfaces that can adapt to the context of use. The design of the User Interface and the implementation of the application code can be separated, while smooth integration of the functionality and the User Interface is supported. The resulting interface is location independent: it can migrate over devices while invoking functionality using standard protocols.

1 Introduction

A variety of new techniques for creating User Interfaces (UIs) for deployment on several different devices are emerging. Model-Based User Interface (MBUI) design is evolving from an academic solution into a practical software engineering methodology for designing multi-device interfaces. XML-based UI descriptions have matured and several toolkits allow building extensive interfaces based on XML documents.

This paper presents a framework for dynamically generating User Interfaces for embedded systems and mobile computing devices. The main purpose of the

framework is to ease the work of the mobile and embedded UI designer and implementor. Often the software implementor of an embedded system or mobile computing device also takes care of the design and implementation of the UI for the system. This is mainly due to the device specific constraints that have to be taken into account: a thorough knowledge of the device is necessary. The Dygimes framework is conceived to ease the creation of the UI without the need for specific knowledge of the hardware or software platform. Runtime transformations of UIs for adaptation to the target device are also supported.

The next section introduces the UI creation process, and discusses the required components to build multi-device UIs for embedded systems and mobile computing devices. These components are discussed into detail in the following sections. First, XML-based UI descriptions are introduced in section 3 as the basic building blocks for the process. Continuing with section 4, the use of a task specification will be explained, and its relation to the XML-based UI descriptions. Section 5 shows how the created UI can be attached (or "glued") to the interfaced functionality it presents in a location-independent manner. Section 6 and 7 show respectively how the system can ensure consistent UIs and how the resulting (concrete) UI can be tailored for a more appealing result. Finally, section 9 discusses the applicability of the system and the obtained results, followed by an overview of the future work.

2 Dygimes process

As stated in the introduction, the main purpose of the framework is to ease the work of the UI designer as well as the work of the application implementor. At the same time a clear separation between the work of the designer and the work of the implementor is supported. This is desirable because of the pitfalls involved in implementing UIs for embedded systems and mobile computing devices, which require specific knowledge about the device and the software platform available for that device.

Throughout this paper we will use a case study, managing a simple publication database, to show how the framework helps the UI designers and system implementors. The database requires the user to login before using the system. The system offers roughly two different kinds of tasks: adding a paper to the database or searching for a paper in the database. Both require some information to complete the tasks successfully. We kept the example intentionally simple for illustration purposes. The next sections explain how we can develop a multi-device UI for this task using the Dygimes framework.

A task specification for this task is developed, enriched with the UI building blocks. This will be sufficient to generate prototype UIs useful in a user-centered design process. The necessary time to create these prototypes is extremely short because many of the steps the designer had to do manually with traditional GUI building toolkits are now automated by the framework. For example; the transformation from the task specification to the resulting functional UIs built by the UI designer is done automatically. A micro-runtime environment offers support

for rendering the created UIs independent of the chosen widget toolkit. Section 4 will show how all UIs stay consistent with regard to the task specification. A graphical overview of the UI construction and rendering process is shown in Fig. 1, all parts (including how the actual communication with the functionality takes place) will be explained in the following sections.

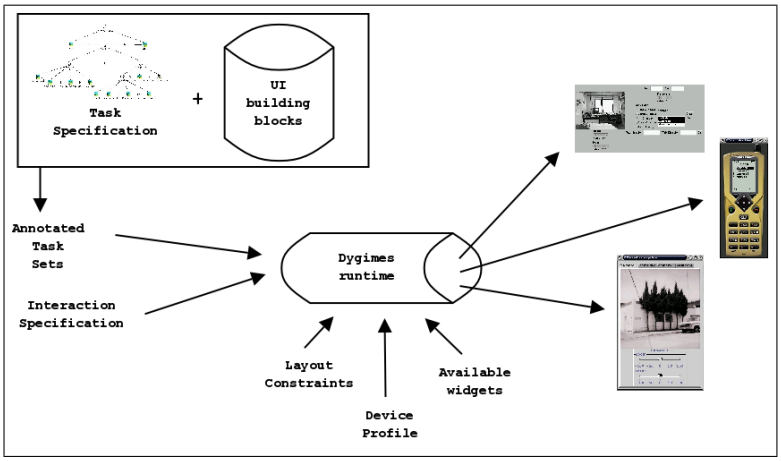


Fig. 1. The process for creating mobile and multi-device User Interfaces.

3 XML-based User Interface Descriptions

In accordance with the recent growth in mobile computing devices usage, the demand for more suitable multi-device UI building toolkits also increases. The reuse of existing UI designs for new devices is problematic: new devices have other or less constraints making the reuse difficult. In contrast consistent look-and-feel is very important, as it contributes to creating a "brand" for the products and makes it easier for customers to use the new device. To enable flexible reusability of existing designs, we need to abstract the way the UI is created for a device in a way it becomes less dependent on device-specific properties.

One way of doing this is the use of high-level XML[1]-based UI descriptions. There are already several propositions and real world examples of the usage of XML to describe UIs for multiple devices: [1, 7, 12–14]. To give the reader an idea of which kind of XML-based UI descriptions are used in our system, listing 1.1 shows the specification of a simple login-dialog. For simplicity, the interaction glue and spatial constraints are omitted from the description. Section 5 discusses how generated events are handled and section 6 discusses the spatial layout constraints. When the renderer (the runtime environment) processes the

[1] eXtensible Markup Language, `http://www.w3.org/XML/`

description shown in listing 1.1, it can produce concrete UIs for different target platforms (shown in Fig. 2) *without* any human intervention.

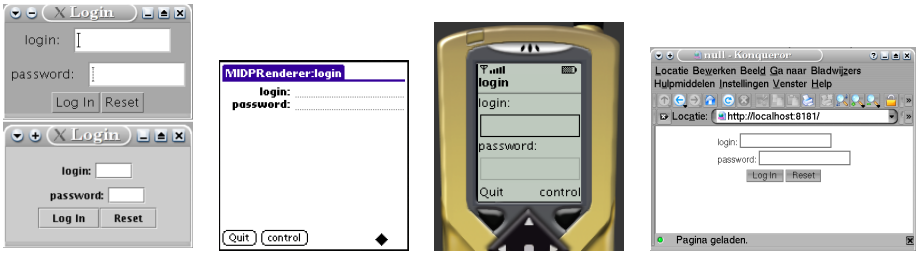
Listing 1.1. The login dialog UI description

```
<ui>
  <group name="login">
   <group name="userinfo">
     <interactor>
      <textfield name="login">
        <info>login</info><text size="10"/>
     </textfield>
     </interactor>
     <interactor>
      <textfield name="passwd">
        <info>password</info><text size="10"/>
      </textfield>
     </interactor>
    </group>
    <group name="control">
     <interactor>
      <button name="in"><info>Log In</info></button>
     </interactor>
     <interactor>
      <button name="reset"><info>Reset</info></button>
    </interactor>
   </group>
  </group>
</ui>
```

Notice the XML description allows to hierarchically group widgets using the "group" tag: this way all groups of widgets that logically belong together are put in the same physical space (e.g. in the same panel or window). At the lowest level, all widgets in a group should always be presented to the user together. 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.

4 Task Model

The design of a consistent interface starts at the task level. There are several advantages of using a task specification: better requirements capturing, consistent and detailed interface design and better integration with real-life situations [6]. Nevertheless, software developers seldom use task specifications to develop UIs for embedded systems and mobile computing devices. One of the main reasons is the wide gap between the implementation of the UI with its specific device-dependent constraints, and the task specification. To make task modeling more



(a) AWT + Swing (b) Palm IIIc (c) Cell phone (d) Browser

Fig. 2. The login dialog, rendered for several devices

attractive there should be a glue to overcome the gap between the technical challenge of realizing the concrete UI and designing it with help from a task model. The framework presented here will offer such functionality.

The framework discussed in this paper uses the ConcurTaskTree task model (CTT) proposed by Fabio Paternò [15]. This notation offers a graphical syntax, an hierarchical structure and a notation to specify the temporal relation between activities. For illustration purpose, a simple CTT of the paper-database example is shown in Fig. 3. A lot of current research extends the ConcurTaskTrees

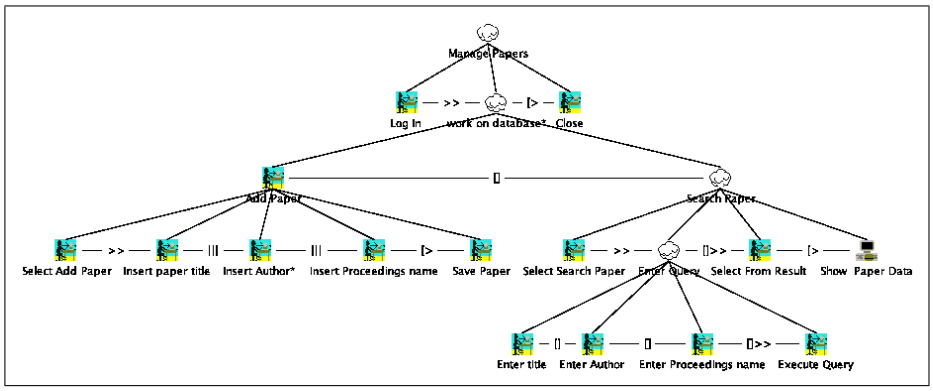


Fig. 3. Managing a simple publication database

notation for multi-device task specifications. For a good understanding of this paper, it suffices for the reader to know that siblings in the tree can be connected

with a temporal relation, such as a disabling operator, independent concurrency between two tasks or an enabling operator.

To convert the task specification into a concrete UI, the *Enabled Task Sets* (ETS) have to be calculated. An Enabled Task Set is defined in [15] as:

> a set of tasks that are logically enabled to start their performance during the same period of time.

The ETS generation is used as a glue between the realization of the UI and the task specification. This approach has also been described in [11, 16] where the focus is on the design of UIs, whereas we concentrate on runtime support for creating the UIs dynamically using widget toolkits as well as markup languages for the resulting UI.

For this purpose we implemented a tool chain for preparing the UI starting from a ConcurTaskTree task specification. The first graphical tool in the tool chain allows to attach the XML-based UI descriptions to leafs in the CTT tree. Next, a tool is offered to describe spatial layout constraints, so the designer can make sure the interface is rendered in a visually consistent manner. Section 6 elaborates on the use of the layout constraints. The last tool is the runtime library: it reads the constructed UI specification (including the task specification), adapts it to the target platform and renders it. The library will stay in memory where it captures events from the UI and handles these in a location transparent way (see Sect. 5). An algorithm for calculating the ETS is included in the framework; the different task sets are generated by the runtime environment. Although the task models can be made with the ConcurTaskTrees tool[15], we can not use their ETS calculation algorithm: the CTT model is annotated with extra information to work with our system. The designer does not have to check whether every possible UI is created for covering each aspect modeled in the task specification: this is done automatically at runtime by the framework through the use of the ETS[10].

The paper database example illustrates this: its CTT specification is saved as an XML file and loaded in the annotation tool where the specification can be decorated with XML-based UI descriptions. Fig. 4 shows the appropriate building block linked to the “Log In” task.

5 The System Glue: an Interaction Model

Once the designer is satisfied with the UI, the next step is to “attach the UI to the application”. More particulary we need to provide a mechanism in which the user can interact with application logic through the generated UI. An important property for this interaction mechanism is the support for *location transparency*: when a mobile device is used, the implementation of the functionality does not have to be on the same device as the UI presenting this functionality. Section 2 also emphasized that we intend to enable the separation of the UI design and system implementation of embedded systems and mobile computing devices. This

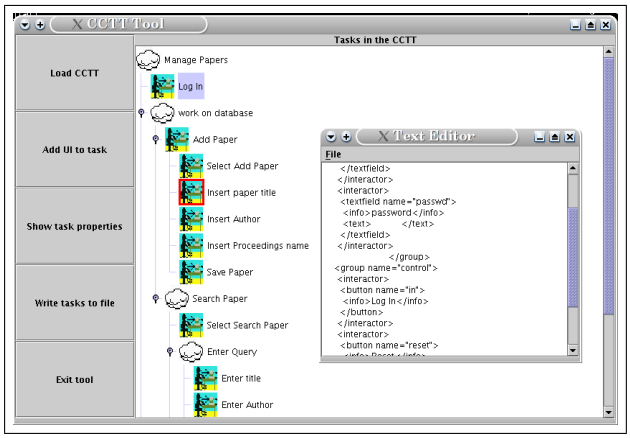


Fig. 4. The CTT annotation tool

means we also need to support several ways to exchange interactive messages between the UI and the application logic, which could be local or remote.

To overcome these problems, the framework offers an extensible "action-handling" mechanism [19]. This mechanism is the glue between the UI and the functionality that will be invoked by the UI. Because a clear separation between the UI and the application logic is needed, Dygimes only needs to know which functionality can be invoked on behalf of the logic and which interactors can be used to execute interactions. It does not need to know how the logic implements the functionality (code encapsulation) or where the implementation resides (location transparency). Even the used technique to invoke the functionality needs to be adaptable. To accomplish these goals, we use interaction descriptions that represent the functionality offered by the application.

In Dygimes, an interaction description is based on the Web Services Description Language (WSDL)[2]. This technology allows the application logic implementor to describe the operations, messages and data types that are supported by the application while existing WSDL editing tools can be used. However, Dygimes also needs a binding between the interaction description and the abstract UI. This binding provides Dygimes with the information needed to determine what must happen when a particular event occurs. For this reason we added a section to the interaction description that describes in what way the generated UI will be bound to the application logic. Suppose, for example, a user pushes the "in" button from listing 1.1. Listing 1.2 then shows what should happen as a response to this action. In this case the loginProcedure operation is sent to the service. This operation is defined in the "paperDBport" portType of the WSDL document. The <uib:parameter> tags describe the parameters. In this case, their values will be extracted from the "login" and "passwd" interactors.

[2] `http://www.w3.org/TR/wsdl`

It is clear that this kind of description separates the development of the UI and the application logic and that it supports location-transparent late binding.

Listing 1.2. The binding between an abstract UI and the application logic

```
<uib:uibinding name="actionbinding" type="paperDBport">
  <uib:interactorbinding name="in">
      <uib:operationlink name="loginProcedure">
          <uib:parameter name="login"/>
          <uib:parameter name="passwd"/>
      </uib:operationlink>
  </uib:interactorbinding>
</uib:uibinding>
```

Dygimes supports different methods to carry out the specified interactions. First, Direct Method Invocation (DMI) can be used to invoke functionality on the application. DMI has the benifit of being fast. However, the drawback with this technique is that DMI can only be used for local invocation with applications implemented in a programming language supporting a reflection mechanism (such as Java). To overcome this problem and to enable location transparency, we make use of web service messaging protocols. These protocols enable us to deploy Remote Procedure Calls (RPC) in an XML-syntax to invoke application functionality. An example of such a technology is the Simple Object Access Protocol (SOAP)[3]. This protocol uses an XML-syntax to describe which method needs to be invoked upon a web service, together with the method's actual parameters. Those parameters are marshalled from language constructs to XML by using particular serializers. Dygimes also supports XML-RPC[4], which is a more efficient implementation of XML-based RPC. Figure 5 shows the extensible architecture of Dygimes enhanced with an interaction model.

A WSDL-based interaction description together with XML-based messaging protocols offer the following benefits:

- Applications become web services-aware through the SOAP implementation. This will be an important advantage in the near future;
- The used approach is device and programming language independent. Java Remote Method Invocation (RMI) for example would restrict the use to Java implemented applications only;
- Interaction with remote logic that runs behind company firewalls is supported;
- Common standards for handling interaction are used, namely XML and web services;
- The automatic generation of functional UIs for remote applications in a location transparent manner is supported.

[3] http://www.w3.org/TR/SOAP/
[4] http://www.xml-rpc.com

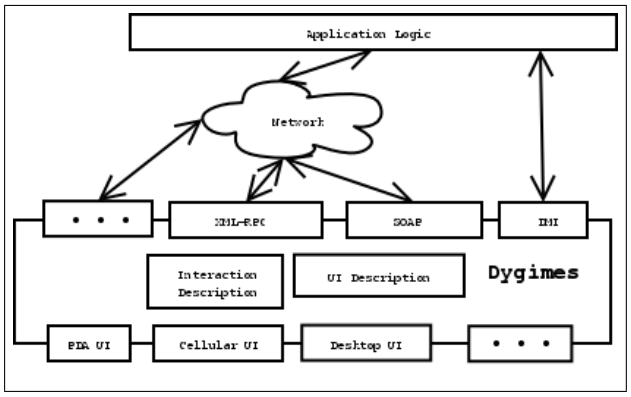


Fig. 5. The location-transparent action handling glue

6 Automatic Layout Management

Abstract UI descriptions and a constraint-based layout management system are combined in our Dygimes framework for developing adaptive UIs for a wide range of devices. When developing a UI description language that 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 techniques offer is necessary. We focus on screen size constraints because this is the most stringent device constraint in this matter. 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. The usage of spatial layout constraints can help the designer in these situations: the consistency of the UI is enforced, yet the UI is flexible enough for large differences in available screen size. There exist several other constraint-based layout management systems like the ones presented in [2, 8, 17], but none really focus on providing a flexible layout for embedded systems and mobile computing devices. The layout manager presented here does not guarantee a "visually pleasing" UI, but makes sure the UI is suitable and consistent on different devices. Adding (platform dependent) placement strategies, like the ones presented in [4], is planned.

Four simple linear spatial constraints, which are described in a simple XML based syntax, are used to express the positioning of components with respect to each other: *left-of*, *right-of*, *above* and *below*. In addition, the available space to lay out the components is divided over 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: assume the constraint *widget A right-of widget B* for example. This means widget A is put in a bucket X with coordinates (x_1, y_1) and widget B is placed in a bucket Y with coordinates (x_2, y_2): the constraint can be expressed as $x_1 > x_2$. A simple constraint solving

algorithm can be used to solve these constraints, we refer to [9] for a survey of several methodologies used for information presentation.

Only constraints between siblings[5] are allowed in our framework: each component of the group will be associated with a unique bucket in the grid. Our layout system 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. As mentioned before, each edge with the label *right* can be expressed as $x_1 > x_2$. A possible solution for the x coordinates will be calculated. The same strategy will be applied for the y coordinates. The results of the two previous steps will be combined in a general solution. The possibility exists however that cycles or multiple edges occur in the graph which imply the presence of conflicting constraints. For handling this kind of inconsistencies, priorities are introduced into our system. The conflicting constraints with the highest priorities will survive. Necessary tool support for specifying constraints is provided in the framework, by means of a specialised constraint editing tool (Fig. 6).

After the presentation structure is calculated, the possibility exists that the layout does not fit on the screen. A layout adapter is imposed to resolve this problem by rearranging and adapting the presentation structure to the screen size of the target device. One of the strategies employed by the adapter consists of placing the components of a splittable group behind each other in a card layout or with tabbed panes. However, sometimes it is impossible to shrink the layout to the size of the screen of the target device. An appropriate warning will be shown to the UI designer at design time.

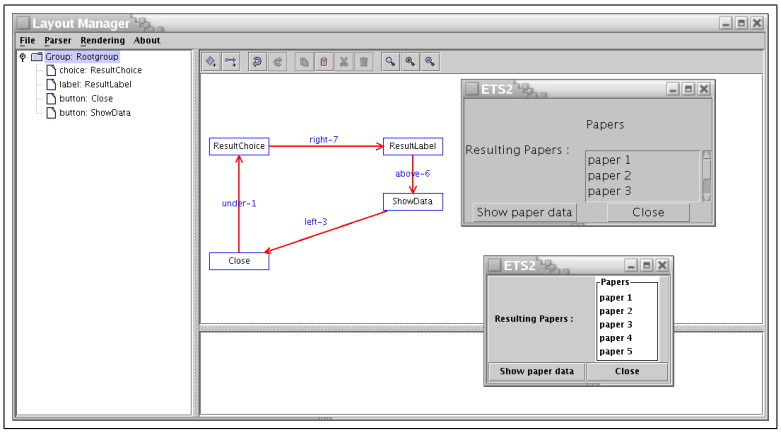


Fig. 6. Tool for managing spatial constraints. Preview of the UI is supported.

[5] The UI descriptions are XML-based: they can be structured as a tree

7 Customization and Templating

The previous sections explained that the framework uses abstract User Interface (AUI) descriptions with constraints to render the user interface. The translation from the abstract interaction objects (AIO) into concrete interaction objects (CIO)[18] can be done fully automatically, however this can give unexpected results. For this reason, the Dygimes framework allows the designer to have more control over the rendering of the UI by allowing them to specify which CIO is used to render an AIO [5]. Mapping rules can specify mappings for one AIO in one specific interface or they can specify mappings for a range of AIOs. This way, the designer can define a template, in the form of a set of mapping rules for a certain platform, that can be refined and adapted for specific user interfaces.

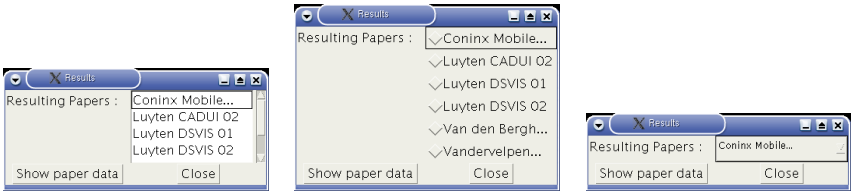
The mapping rules are intentionally kept simple because they are to be used at runtime on devices that can have very limited resources. The CIO that will be used to render a certain AIO depends on the type of the AIO and the name that identifies the AIO. Mapping rules can specify part of the name of an AIO, the complete name of an AIO or no name in order to define their applicability. We will illustrate this with the example of the paper database.

Listing 1.3. Two of the specified mapping rules

```
<mapping>
  <aio2cio>
    <aio>choice</aio>
    <cio>awt.CheckboxGroup</cio>
  </aio2cio>
</mapping>
<mapping>
  <aio2cio>
    <aio>choice</aio>
    <cio>awt.List</cio>
    <name>large</name>
  </aio2cio>
</mapping>
```

For this example, we used a template for the Java AWT platform. Two of the rules in this template are shown in listing 1.3. The first rule shows that the AIO "choice" is mapped onto a CheckboxGroup by default (no name is specified). When the number of items is large or varies over time, as is the case for the AIO that contains the result of the query, a List widget is a better choice. By giving the AIOs a name that contains "large", the second rule in listing 1.3 is used which gives the wanted result (figure 7(a)). If the designer prefers a Java AWT Choice for rendering the query result, this can be indicated by adding a rule that contains the full name of the AIO (Fig. 7(c)).

A feature of the templating system, which is not shown in this example, is that it allows to specify a "null" CIO for AIO's that should not or cannot be represented on a certain platform. This generates a lot of flexibility but can introduce problems as well: when the AIO that cannot be represented is crucial



(a) List (b) CheckboxGroup (c) Choice

Fig. 7. Three possible mappings for the query result on the AWT platform

in a certain task, it can render a whole part of the user interface useless. A way to deal with this situation in an effective way, without loosing the flexibility of the system, is being worked on. Currently, the designer is still being forced to deal with this situation explicitly, by providing an adapted task model for the specific device, as needs to be done in the approach taken by Calvary et al[3].

8 The Paper Database Example: resulting User Interfaces

Now that all the different parts are discussed, these techniques can be applied to a simple but illustrative example: the paper database (see Fig. 3). This task specification is annotated by high-level XML-based UI descriptions, depicted in Fig. 4.

Next, a set of (optional) spatial constraints can be defined to ensure consistent presentation of the UI on different devices (see Fig. 6). The annotation tool does its job by reading the XML-document which can be saved by the ConcurTaskTree tool, and inserting the UI building blocks into the XML-document. The result is a new XML document containing all the necessary information. The inserted UI building blocks can have spatial layout constraints defined.

These are the three things the designer should provide:

1. The task specification
2. The UI building blocks and their layout constraints
3. The relation between the UI building blocks and the different tasks in the task specification

This information is saved in an XML document. From this point the Dygimes runtime will handle everything automatically.

The XML document (describing the annotated task models and the constraints) produced by the tool can be fed directly to the runtime environment. First, it calculates the Enabled Task Sets. For the example it finds the following sets:

$$ETS_1 = \{LogIn\}$$
$$ETS_2 = \{Close, SelectAddPaper, SelectSearchPaper\}$$
$$ETS_3 = \{Close, SavePaper, Insertpapertitle, InsertAuthor, InsertProceedingsname\}$$
$$ETS_4 = \{Close, SelectFromResult, ShowPaperData\}$$
$$ETS_5 = \{Close, Entertitle, EnterAuthor, EnterProceedingsname\}$$
$$ETS_6 = \{ExecuteQuery, Close\}$$

When these sets are calculated, it is known which UI building blocks should be presented at the same time. For example, ETS_3 contains a UI for adding information about a paper into the database. The UI building blocks attached to the different tasks of ETS_3 are extracted and merged into a single XML-based UI description (see Fig. 8). The resulting description is rendered by the framework, according to the mapping rules and layout constraints. Depending on the device an appropriate UI is generated: Fig. 9 shows how Java Swing or a Java-enabled Mobile Phone can be used.

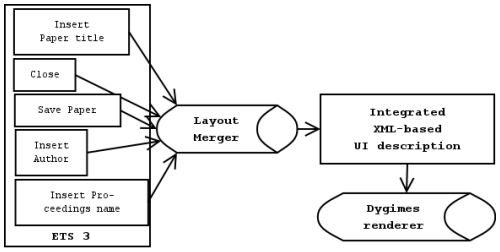


Fig. 8. Merging the UI building blocks of ETS_3

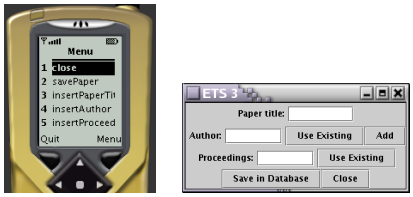


Fig. 9. ETS_3 automatically generated

Until now all the UIs are generated automatically out of the annotated task specification. Using the runtime environment, this can also be done dynamically (at runtime). Following this methodology, we get a set of User Interfaces which are perfectly fit to perform the task described by the task specification. This is one of the advantages of this approach.

Switching from an ETS to another requires a dialog model: an activity chain diagram is generated out of the task specification to indicate in which order the dialogs appear for the user. A detailed description of the dialog generation algorithm can be found in [10]. The activity chain describes the transitions of an ETS to another ETS. A State Transition Network for the dialog model is built by inspecting the temporal relations to solve this problem.

9 Conclusion and Future Work

We presented a framework for creating UIs for embedded systems and mobile computing devices. It incorporates several techniques from model-based UI de-

sign, XML-based UI descriptions, automatic layout management and location-transparent event handling. The main purpose is to ease the creation of consistent, reusable and easy migratable UIs. The UIs can automatically adapt to new devices, offering the same functionality, without being redesigned. If one wants a better adaptation to a particular device, the designer can choose to provide a set of mapping rules and/or a set of better spatial constraints to embellish the presentation of the UI for that device. Notice the actual UI does not need to be rebuilt from scratch here.

The Dygimes framework is already successfully used in the SEESCOA[6]-project. Most developers in this project had no prior experience in UI design, but were able to use the tool within a few hours for creating simple UIs. This practical experience allowed us to use real-life experience to add tools and other techniques necessary to create suitable UIs.

We are looking to extend the framework not only to support multiple devices, but also to support multiple and mixed modalities. The final goal is to make the tool suitable for the usage in pervasive and ubiquitous environments, where the UI is loosely-coupled to the devices and can migrate from one device to another device. An important aspect here is to take the human factors in account (e.g usability of the interface in certain situation).

The Dygimes process can be described using the reference framework for platicity of Calvary et al. [3]. We plan to do this, so the properties of our system can be compared with other systems. Using the framework to describe Dygimes will allow a better localization of important shortcomings. For now, we focused on building a framework to generate multi-device UIs at runtime provided an annotated task model is given as input.

10 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). The SEESCOA project IWT 980374 is directly funded by the IWT[7]. The authors would like to thank Jos Segers, Tim Clerckx, the SEESCOA partners and the reviewers of this paper for their contributions.

References

1. Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. *UIML: An Appliance-Independent XML User Interface Language.* World Wide Web, `http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html`, 1998.

[6] Software Engineering for Embedded Systems Using a Component Oriented Approach, `http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/SEESCOA/`

[7] Flemish subsidy organization

2. Alan Borning. ThingLab – A Constraint-Oriented Simulation Laboratory. Technical report, XEROX PARC, 1979. report SSL-79-3.
3. Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Nathalie Souchon, Laurent Bouillon, and Jean Vanderdonckt. Plasticity of user interfaces: A revised reference framework. In *Task Models and Diagrams for User Interface Design*, pages 127–134, Bucharest, Romania, July 18-19 2002. TAMODIA 2002.
4. François Bodart, Anne-Marie Hennebert, Jean-Marie Leheureux, and Jean Vanderdonckt. Towards a dynamic strategy for computer-aided visual placement. In *Workshop on Advanced visual interfaces*, pages 78–87. ACM press, 1994.
5. Jan Van den Bergh, Kris Luyten, and Karin Coninx. A Run-time System for Context-Aware Multi-Device User Interfaces. In *HCI International*, June 2003. Accepted for publication.
6. Alan Dix, Janet Finlay, Gregory Abowd, and Russel Beale. *Human-Computer Interaction (second edition)*. Prentice Hall, 1998.
7. Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta. Applying Model-Based Techniques to the Development of UIs for Mobile Computers. In *IUI 2001 International Conference on Intelligent User Interfaces*, pages 69–76, 2001.
8. Maloney J, Boming A, and Freeman-Benson BN. Constraint Technology for User Interface Construction in ThingLab II. In *OOPSLA*, 1989.
9. Simon Lok and Steven Feiner. A Survey of Automated Layout Techniques for Information Presentations. In *Proceedings of SmartGraphics 2001*, March 2001.
10. Kris Luyten, Tim Clerckx, Karin Coninx, and Jean Vanderdonckt. Derivation of a Dialog Model for a Task Model by Activity Chain Extraction. In *Interactive Systems: Design, Specification, and Verification*, 2003.
11. Giullio Mori, Fabio Paternò, and Carmen Santoro. Tool Support for Designing Nomadic Applications. In *Intelligent User Interfaces*, 2003.
12. Andreas Muelller, Peter Forbrig, and Clemens Cap. Model-Based User Interface Design Using Markup Concepts. In *Interactive Systems: Design, Specification, and Verification*, pages 30–39, 2001.
13. Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. Generating remote control interfaces for complex appliances. In *User Interface Software and Technology*, 2002.
14. Dan R. Olsen, Sean Jefferies, Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-modal interaction using XWeb. In *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 191–200, N.Y., November 5–8 2000. ACM Press.
15. Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.
16. Fabio Paternò and Carmen Santoro. One model, many interfaces. In Christophe Kolski and Jean Vanderdonckt, editors, *CADUI 2002*, volume 3, pages 143–154. Kluwer Academic, 2002.
17. Michael Sannella, John Maloney, Bjorn Freeman-Benson, and Alan Borning. Multi-way versus One-way Constraints in User Interfaces: Experience with the DeltaBlue Algorithm. *Software - Practice and Experience*, 23(5):529–566, 1993.
18. Jean Vanderdonckt and François Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In *ACM Conference on Human Aspects in Computing Systems InterCHI'93*, pages 424–429. Addison Wesley, 1993.
19. Chris Vandervelpen, Kris Luyten, and Karin Coninx. Location Transparant User Interaction for Heterogeneous Environments . In *HCI International*, June 2003. Accepted for publication.

