Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Semi-automatic user interface generation considering pointing granularity

Sevan Kavaldjian, David Raneburger, Jürgen Falb, Hermann Kaindl and Dominik Ertl
Institute of Computer Technology
Vienna University of Technology
Vienna, Austria
{kavaldjian, raneburger, falb, kaindl, ertl}@ict.tuwien.ac.at

***Abstract*—Development of GUIs (graphical user interfaces) for multiple devices is still a time-consuming and error-prone task. Each class of physical devices—and in addition each application-tailored set of physical devices—has different properties and thus needs a specifically tailored GUI. Current model-driven GUI generation approaches take only few properties into account, like screen resolution.**

Additional device properties, especially pointing granularity, allow generating GUIs suited for certain classes of devices like touch screens. This paper is based on a model-driven UI development approach for multiple devices based on a discourse model that provides an interaction design. Our approach generates UIs using an extended device specification and applying model-transformation rules taking them into account. In particular, we show how to semi-automatically generate finger-based touch screen UIs and compare them with usual UIs for use with a mouse that have also been generated semi-automatically.

***Index Terms*—Model-driven user interface generation, device specification**

I. Introduction

Manual creation of GUIs (graphical user interfaces) is hard and expensive, so we strive for semi-automatic generation. The formal specification of a physical device is insufficient for the automatic generation of GUIs, however, if there are application-specific device properties like pointing granularity and virtual input devices. For example, a touch screen panel can be used for fine-grained up to coarse-grained interaction. In addition, it can be combined with a mouse and a real or a virtual keyboard. For an automated generation of user interfaces, the possible properties and their values have to be refined. This results in an application-tailored device specification with application-specific properties.

We are able to semi-automatically generate GUIs in a way that takes pointing granularity into consideration. More precisely, we generate WIMP (window, icon, menu, pointer) UIs. Our model-transformation rules can be divided into different rule sets according to specific device properties. One rule set is suitable for fine granularity, another rule set for coarse granularity. Both rule sets have a common core, however. These different rules can result in different layout, widget size and even widget type selection. The generator program chooses the respective rule set according to the specified properties. Example applications for the rules that take coarse granularity into account are finger-based touch screen applications and applications for motor-impaired users.

The remainder of this paper is organized in the following manner. First, we review the state of the art in terms of related work. Then we sketch our discourse models taken as input for the semi-automatic generation of GUIs. After that, we discuss devices and their relevant properties. Based on all that, we present our semi-automatic generation and how it takes pointing granularity into account.

II. State of the Art

An advanced approach to generating multi-device UIs starts with hierarchical task modeling using ConcurTaskTrees (CTT) [1], where different temporal relations among tasks can be specified (e.g., enabling, disabling, concurrency, order independence, and suspend-resume). Out of such task models, a user interface can be created through several steps of model transformations. CTT also allows the derivation of presentation units consisting of a set of tasks and their transitions. A model-based multimodal user interface generation process for different devices is presented in [2], [3]. One logical description of an interface can be transformed into interfaces for different devices. A prototypical implementation of a GUI for a desktop application and a corresponding GUI used on a mobile phone are demonstrated.

When a GUI is generated for different devices, a mechanism can be applied that transforms a potentially large interface (highest degree of parallelism) into a more compact (serialized) form. An easy approach is just to add scroll bars to a GUI developed for a PC application which shall be used, e.g., on a PDA with reduced screen size. This potentially results in an annoyed user, as scrolling is often required when using the interface.

To circumvent this drawback, a technique presented in [4] has been developed which automatically transforms Web pages into hierarchically structured subpages. This approach considers the size of the screen, the size of page blocks, the number of blocks in each transformed page, the depth of the hierarchy that the subpages form and the semantic coherence between the blocks. "Size" is actually interpreted here in terms of number of pixels, i.e., screen resolution. Another approach to taking screen size into account for generating multi-target user interfaces can be found in [5], but it actually interprets size as screen resolution as well. It resizes based on given numbers

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

of pixels on the screen. Note, that such approaches are difficult to apply for touch screens with variable screen resolution, since the metric size of input widgets matters there.

Design guidelines for touch screens are described in [6], e.g., that the object separation should be at least $1/8''$. As another example, when the consequences of selection are destructive, i.e., a "remove" or "delete" functionality, confirmation helps to avoid inadvertent selection.

The minimum size for targets on a touch screen is mentioned in [7]. This study suggests a size of 9.6mm with 5% erroneous trials for discrete screen objects. Previous studies such as [8] investigated touch key design for target selection on a mobile phone, more precisely the size of objects and their location on the screen. However, this study focused on mobile phones, which have more constraints on the screen size than, e.g., touch screens for kiosk applications. A main result of this study was that larger touch key sizes lead to higher performance and more subjective satisfaction. The size of the targets on a touch screen is also part of an empirical study in [9]. It points out that large targets should be used whenever possible, having benefits like reduced contact time and fewer errors. However, a drawback of large targets are potentially less manipulable objects on the screen.

III. Discourse Model

The input for our semi-automatic user interface generation approach is a discourse model, see [10]–[13]. Such a discourse model serves as an interaction design on a high level of abstraction and based on concepts of human language theories. A small excerpt of a larger discourse model for interacting with a robot shopping cart is shown in Figure 1. We use this discourse model as a running example throughout the remainder of this paper. The part shown in Figure 1 models presenting the current state of the customer's shopping list that she has entered before and the possibility of removing items from the shopping list.

The main ingredients of our discourse models are communicative acts. A communicative act is represented as a rounded rectangle and models an utterance of one of the communication partners. In our example, the robot cart may ask a *closed question* or *inform* the customer, while the customer can provide an *answer* to the question. The association of a communicative act with a communication party is done by color. In our example, the green (or dark gray) communicative acts are uttered by the robot cart and the yellow (or light gray) ones are uttered by the customer.

Additionally, some communicative acts, like *Question* and *Answer* form a so-called adjacency pair—it is represented by a diamond in our discourse models—to define the turn-taking and thus the order of the utterances. The two communicative acts forming an adjacency pair must always belong to different communication parties. Some communicative acts, like *Informing*, do not require an explicit response and, therefore, form a degenerated adjacency pair consisting of only an opening communicative act (see the *Informing* in Figure 1).

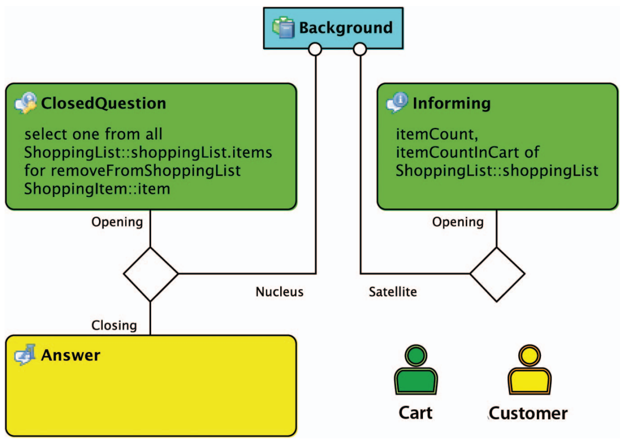


Fig. 1. Discourse model excerpt.

A communicative act also conveys information to achieve its purpose—e.g., asking a particular question. The communicative act's content is specified by a query-like expression language that refers to elements in the domain of discourse. For instance, the expression "`select one from all ShoppingList::shoppingList.items for removeFromShoppingList Shopping::item`" tells that the customer may select one item from the `items` of her `shoppingList` to remove it from the list; `shoppingList` is a variable of type `ShoppingList`, which is a reference to the *ShoppingList* class in the domain of discourse model.

Turn-taking sequences can be related in a discourse model with each other to build a tree structure. These relations may represent some subject-matter relationships as well as temporal relationships. In our example, the question–answer pair and the informing are related by a *Background* relation. This *Background* relation states that the Informing in the satellite branch contains background information to the "nuclear" utterances to support the customer in answering the closed question. This relation does not imply a temporal order per se. For instance, both pieces of information can be presented in parallel as done in the graphical user interface shown in Figure 5 below. On the other hand, if the cart uses speech for asking the customer, a natural order would be to ask the question first, then to provide the background information, and finally to wait for the answer of the customer. Thus, our discourse models specify classes of dialogues, with different possible orders of communicative act utterances.

IV. Devices and Their Properties

In order to parameterize our semi-automatic user interface generation approach for diverse devices, we need some formal specification of devices and their properties. Specifying physical devices, like desktop PCs and PDAs, and their physical properties, e.g., screen size, resolution and supported GUI toolkits, allows us to generate graphical user interfaces with an appropriate screen layout, for example. A physical device specification is often insufficient for semi-automatic user interface generation, however, since it does not specify how an application makes use of the physical device. For instance, if a touch screen is available, an application can use the device in different ways by imposing different interaction styles—pen-based interaction or finger-based interaction. These different ways of use shall be reflected in the user interface generation too, since finger-based interaction requires larger input widgets than pen-based interaction. Consequently, we introduce application-tailored device specifications in addition to physical device specifications.

Figure 2 illustrates both kinds of device specifications and their relation. The application-tailored device is derived from the physical device as a specialization and inherits the physical device properties from it. For instance, such properties are screen resolution and DPI, which together specify the metric screen size as well. The additional properties of the application-tailored device specify how the physical device is used by the application. For example, a physical device "touch screen" that can be either used with a pen or with fingers requires the additional property "pointing granularity" to be included in the application-tailored device specification. This property can have the value "coarse" for finger-based interaction, whereas for pen-based applications the "pointing granularity" will have the value "fine".

An application can further combine physical and virtual devices in a special way. For instance, a touch screen solution usually does not include a physical keyboard. Nevertheless, an application could require the presence of a virtual keyboard, which in turn imposes constraints (especially spatial layouting constraints) on the automatic user interface generation. Therefore, our specification of application-tailored devices includes also properties defining the inclusion of virtual devices. Their use for automated generation of user interfaces, however, is beyond the scope of this paper.

V. Semi-automatic UI Generation

Having such application-tailored device specifications available, user interface transformation rules and code generation can be restricted to a set or range of device property values. Our model-transformation rules are divided into different rule sets accordingly. Applying rules from these different rule sets can result in different layout, widget size and even widget type selection. The automated UI generation chooses the respective rule set according to the specified device properties. For example, rules for a coarse pointing granularity—due to finger-based use of the device—can lead to larger widgets or avoidance of complex widgets like drop-down boxes.

A. *Overview*

Our GUI generation process is divided into two steps. First a structural user interface model is generated, using the discourse model and a device specification as input. This structural user interface model specifies the widget hierarchy of the user interface. The structural user interface model is still independent of the GUI toolkit (in our case Java Swing) used to display the screens, but it depends already on the target device, e.g., a touch screen. Default values for metric widget sizes are chosen according to the specified metric screen size of the device. Device properties like screen resolution, etc., are taken into account either directly during the structural UI model generation or by the code generator. In the second step, a code generator translates the structural UI model into GUI toolkit specific source code.

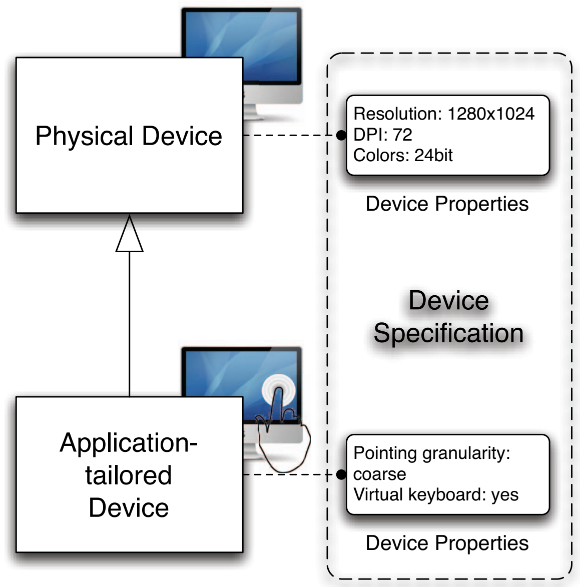


Fig. 2. Physical and application-tailored device specifications.

The *Discourse To Structural UI Transformer* gets the discourse and the domain-of-discourse models as input and transforms them into a structural UI model. This transformation takes device properties into account. The Discourse To Structural UI Transformer performs a model-to-model transformation based on rules. Such a transformation rule can state, for example, that each "Informing" communicative act found in the discourse model has to be transformed to a panel containing a label widget for each domain-of-discourse object referred to by the communicative act's propositional content.

The *UI Code Generator* gets the generated structural UI model as input and maps it to widgets available on the target platform's GUI toolkit. To complement the information not provided by the structural UI model, additional device properties and style information can be included deliberately.

The separation of the rendering process in two steps has the major advantage that the generated structural UI model is still platform independent and can be translated into several different GUI-toolkit languages in a second step. Another advantage is the possibility of influencing the screen design through modifying the generated structural UI model, before triggering the actual code generation. This leads to a more satisfying final user interface.

B. Discourse Model to UI Structure Transformation

Our model-driven approach transforms discourse models into structural UI models that are close to the final user interface but still GUI toolkit-independent (see, e.g., Figure 3). We first explain the transformation approach as well as the differences between transformation rules for coarse-grained pointing and transformation rules for fine-grained pointing. Subsequently, we present the structural UI model resulting from the transformations for coarse-grained pointing granularity.

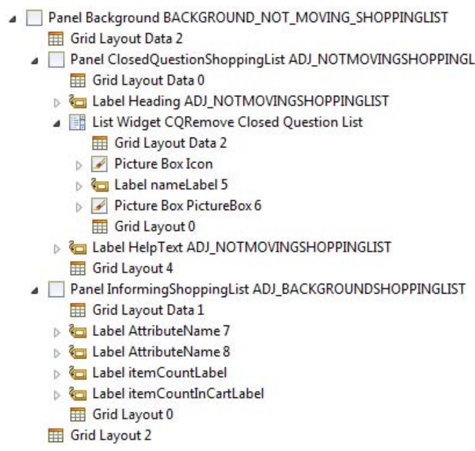
A self-developed transformation engine executes the transformation of a given discourse model. Transformation languages and engines like ATL[1] (ATLAS Transformation Language) or MOLA[2] (MOdel transformation LAnguage) also support the same transformation process, but these approaches lack conflict resolution strategies, requiring rules to be carefully designed in a way that only one rule matches a model element at any time. This is inappropriate for GUI rendering, since one wants to provide some general rendering rules and some more specific ones matching the same element with the specific ones taking precedence over the general ones.

Our model-driven transformation approach uses a process consisting of two interleaved transformation steps:

1) The first step applies rules according to the defined application-tailored device specification, that generate an overall UI structure based on patterns matched in the discourse model. These rules generate abstract widgets like labels for headings and placeholders for data of the propositional content. They also select the parts of the propositional content to be rendered and associate them with the placeholders.
2) The second step executes specific content transformation rules within the context of the rules of the first step. This allows the selection of abstract widgets depending on the content type, the content's referring communicative act type and the current context the communicative act is embedded in as defined by the enclosing rule.

We explain this process in more detail by means of our running example and in [14]. For transforming the discourse model excerpt in Figure 1, we need structural transformation rules for transforming the *Closed Question-Answer* adjacency pair and the *Informing* communicative act as well as the *Background* relation. Second, we need content transformation rules for transforming content types, like string, picture and number dependent on the communicative act they are embedded in. The transformation rules also contain heuristics to improve the generated structural UI model. For example, rules can transform content attributes differently based on their attribute name, e.g., a *name* attribute can be used as the heading for the rendered content. The structural rules below illustrate the difference between rules applied to the same communicative act for different device specifications.

[1]http://www.eclipse.org/m2m/atl/
[2]http://mola.mii.lu.lv/



```
Panel Background BACKGROUND_NOT_MOVING_SHOPPINGLIST
  Grid Layout Data 2
  Panel ClosedQuestionShoppingList ADJ_NOTMOVINGSHOPPINGLIST
    Grid Layout Data 0
    Label Heading ADJ_NOTMOVINGSHOPPINGLIST
    List Widget CQRemove Closed Question List
      Grid Layout Data 2
      Picture Box Icon
      Label nameLabel 5
      Picture Box PictureBox 6
      Grid Layout 0
    Label HelpText ADJ_NOTMOVINGSHOPPINGLIST
    Grid Layout 4
  Panel InformingShoppingList ADJ_BACKGROUNDSHOPPINGLIST
    Grid Layout Data 1
    Label AttributeName 7
    Label AttributeName 8
    Label itemCountLabel
    Label itemCountInCartLabel
    Grid Layout 0
  Grid Layout 2
```

Fig. 3. Structural UI model excerpt automatically generated for touch screen.

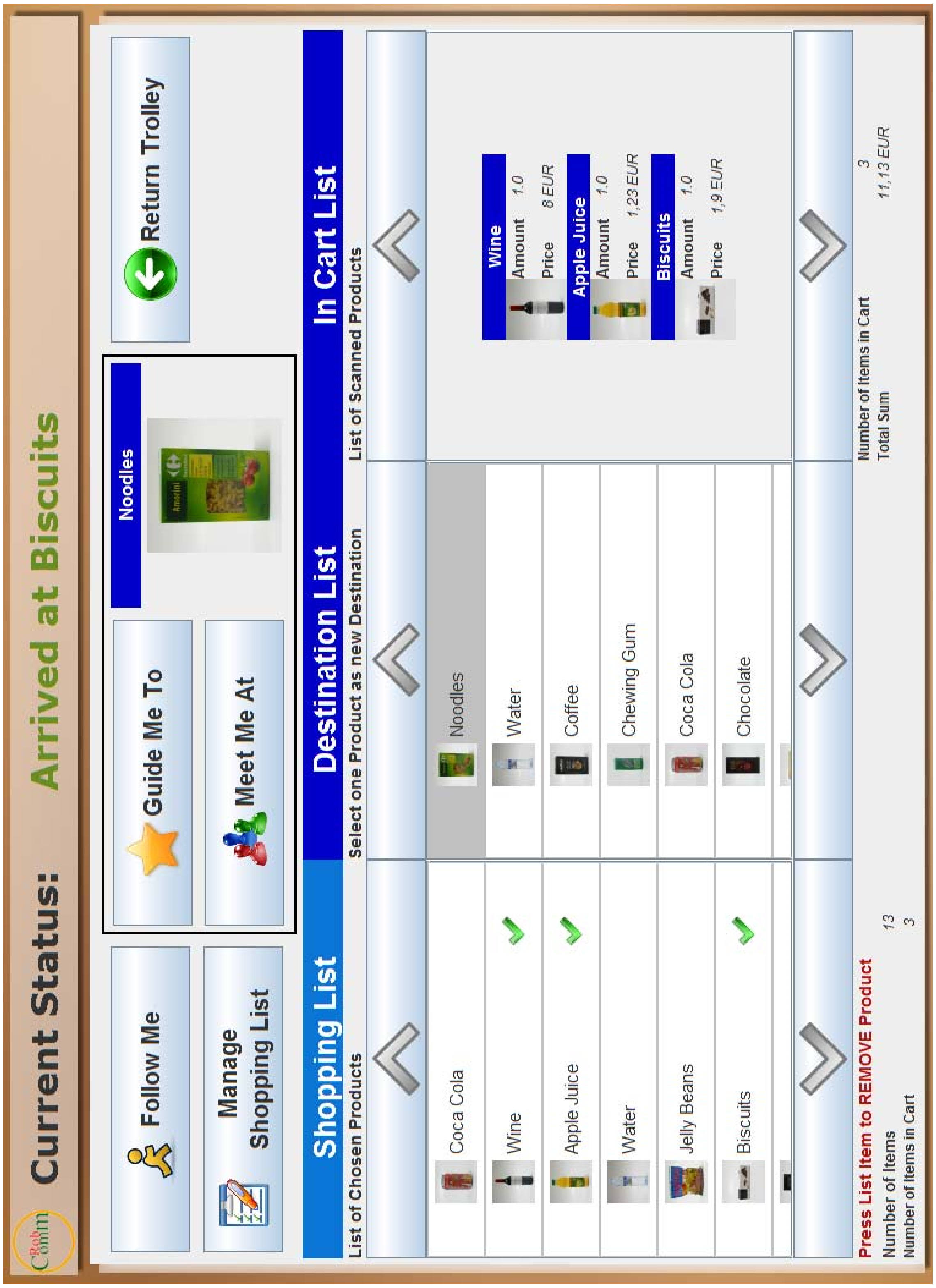
Closed question transformation rule for coarse granularity: This rule transforms each *Closed Question-Answer* adjacency pair to a panel with a label and a list with each list entry consisting of an output widget placeholder for the content object's identifier (e.g., name). The label represents the heading for the overall list and the label's text is derived from the name of the closed question communicative act. A property is set in the list widget element of the structural UI model that enables the generation of Scroll Buttons for the list. The result of this rule is the rendered "Shopping List" in the left part of Figure 4. Due to the coarse pointing granularity, the clickable area for each list entry is extended to the size of the whole list entry.[3]

Closed question transformation rule for fine granularity: This rule transforms each *Closed Question-Answer* adjacency pair to a panel with a label and a list with each list entry consisting of an output widget placeholder for the content object's identifier (e.g., name) and a button. The label represents the heading for the overall list and the label's text is derived from the name of the closed question communicative act. The result of applying this rule can be seen in the upper part of Figure 5. Due to the fine pointing granularity, the clickable area for each list entry is restricted to the size of the remove button. Additionally, a scrollbar is added to the right of the remove buttons.

Informing transformation rule: This rule matches an adjacency pair with an Informing communicative act (upper right part of Figure 1). The rule transforms the matched input to a *Panel* containing an *Output Widget* placeholder. The output widget placeholder has a property that holds an OCL[4] expression which selects all attributes of the referred content. As can be seen at the bottom left part of Figure 4 and the bottom of Figure 5, the Informing is rendered equally

[3]The size of this figure as given here is on purpose nearly the real size on the physical device. The minimum target size on the device is as suggested in the literature referenced above.

[4]OCL – Object Constraint Language, see http://www.omg.org/cgi-bin/doc?ptc/2003-10-14 for its specification



Current Status: Arrived at Biscuits

Noodles

Shopping List
List of Chosen Products

Coca Cola	
Wine	✓
Apple Juice	✓
Water	
Jelly Beans	
Biscuits	✓

Destination List
Select one Product as new Destination

- Noodles
- Water
- Coffee
- Chewing Gum
- Coca Cola
- Chocolate

In Cart List
List of Scanned Products

	Amount	Price
Wine	1.0	8 EUR
Apple Juice	1.0	1,23 EUR
Biscuits	1.0	1,9 EUR

Press List Item to REMOVE Product

Number of Items	13
Number of Items in Cart	3

Number of Items in Cart	3
Total Sum	11,13 EUR

Fig. 4. Final touch screen user interface.

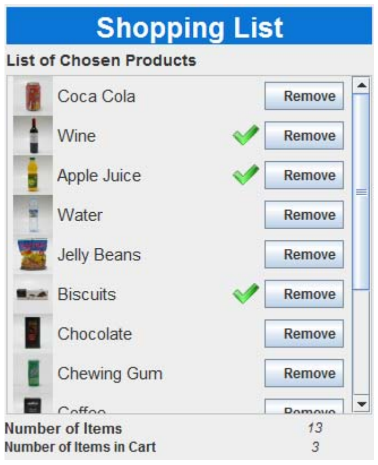


Fig. 5. Excerpt of final desktop user interface.

in both cases, independently of the pointing granularity. This implies that output widgets are not influenced by the pointing granularity. Therefore, this rule belongs to the set of common core rules.

When these rules are applied to our discourse model excerpt in Figure 1, we get the generated structural UI model for touch screens illustrated in Figure 3. The resulting structure of the *Closed Question-Answer* adjacency pair in Figure 3—the panel *ClosedQuestionShoppingList* and its children—corresponds to the structure described in the "Closed question transformation rule for coarse granularity" and is illustrated in the final UI as the Shopping List in Figure 4. The resulting structure of the Informing in Figure 3—the panel *InformingShoppingList* and its children—corresponds to the structure described in the "Informing transformation rule". It is shown in the final UI by two labels below the Shopping List in Figure 4. The label with the text "Press List item to REMOVE Product" results from the "Closed question transformation rule for coarse granularity". It is contained in the structural UI model in Figure 3 as the label "HelpText". The output widget placeholders are replaced by the appropriate second level rules.

The structural UI model generated for fine granularity is similar to the one for touch screens with coarse granularity. It additionally has a *Remove* button contained in the list widget element of the closed question panel. The resulting buttons in the final UI are shown on the right in Figure 5.

C. Screen Generation

The second step of the generation process is the actual screen generation. This process creates the target toolkit implementation of the final UI, that represents all windows and frames needed to communicate with a user.

The code generator's task is the translation of a structural user interface model to source code in a specified programming language. Since graphical user interfaces usually consist of a limited number of widget types combined in various ways, the resulting code can be seen as a combination of corresponding code fragments.

The most appropriate generation approach, regarding the input of the process, is the combination of meta-model and template-based generation. As large parts of our discourse modeling and user interface generation platform are based on the Eclipse Modeling Framework[5] and, therefore on Java, we chose Java Emitter Templates (JET) as the template language. We implemented the code generator as a template engine, using Java Swing[6] as target toolkit.

The templates give the system designer a fair amount of flexibility as they support the separation between functionality and content. Static data already available at generation time can be retrieved directly from the structural UI model. On the contrary, list widgets, for example, are filled with dynamic data available only during runtime of the system. This allows the modification of data without needing to generate the system anew.

The structural UI takes layouting into account, but hardly contains anything concerning the *look* of the application. All this data can be encapsulated in a style sheet, whose elements are then associated with the structural UI model elements. Moreover, the look aspect is something that does not influence the logic of a system at all and can, therefore, be treated separately.

Cascading Style Sheets (CSS) encapsulate all information concerning the *look* of an object, making it easy to adapt a system to a new look by simply exchanging the style sheet. Since CSS have been designed for HTML pages, we had to map CSS attributes to Java Swing attributes within the generator templates. Apart from this mapping, the code generator provides default values for attributes like font type and size, as well as for border thickness and color. These default values are needed in case the system designer does not specify all attributes needed by the generator. If a style sheet is provided, all specified attributes are applied to the corresponding widget, overriding the default values.

The code generator further supports the fall-back mechanism known from HTML. In a first step, the generator tries to extract all the values from the CSS style that is associated with the *structural UI model widget's identifier*. If no widget-specific style is found, the code generator tries to extract the values from the *style class* associated with the widget. In case no style class is defined for the widget in the structural UI model, the code generator checks for a corresponding CSS *element style*. If all attempts fail, the value is set to the default value that is predefined in the code generator.

The code generator extends the basic functionality offered by CSS. An extension that influences the *look* of an application is the representation of enumeration values through icons (e.g., tick marks in Figures 4 and 5). Furthermore, decorative images or sounds can be specified for different widgets.

We further provide several CSS templates that define the sizes of several widget types. In this way the style sheets

[5]http://www.eclipse.org/modeling/emf/
[6]http://java.sun.com/products/jfc/

capture not only characteristics regarding the *look*, but also the *feel* of an application, e.g., in case of a touch screen UI. The code generator selects the appropriate style sheet template according to the application-tailored device specification and extracts the needed information at generation time, specifying the button size or the size of each list widget entry in Figure 4. The style sheets also define default values regarding the size and font of every widget type. In this way, the minimum size of buttons or other interactive widgets like the shopping and destination list in Figure 4 are set. As these settings are specified for a widget type, they are valid for all widgets in the structural UI model unless they are overridden. Therefore, they can be seen as default values for the application-tailored device. They can be refined either directly, by editing values in the style sheet, or by attaching a style reference to the structural UI widget. In effect, the default values set by the template are substituted by the values defined in the corresponding style class. This allows the system designer to customize the standard values for selected widgets or widget classes.

The possibilities of style sheets however, do not cover all aspects concerning the *feel* of a user interface on different application-tailored devices as they do not allow modifying the widget hierarchy of the structural UI. This aspect is covered by the mapping rules applied during the first generation step. An example are the additionally generated scroll-buttons for each list in Figure 4. Their generation results from the application of different transformation rules than the ones applied for generating the desktop UI illustrated in Figure 5.

The style sheets are applied only during the second step of the generation, i.e., the structural UI model to Java Swing translation. Any change to characteristics that are captured by the style sheet, requires only the repetition of the second step instead of the whole generation process.

VI. Conclusion

In this paper, we extend the usual device properties taken into account for automated GUI generation. We include information on having a real or a virtual keyboard, as well as pointing granularity for a touch screen. In fact, these are device properties beyond those for physical devices. We call the extended specifications, therefore, application-tailored device specifications.

Our focus in this paper is on semi-automatic UI generation that takes pointing granularity into account. Fine pointing granularity leads to smaller input widgets on the screen, whereas coarse pointing granularity results in larger input widgets. Since we had to generate UIs for a finger-based touch screen, we developed specific model-transformation rules for course pointing granularity.

In order to show the flexibility of the generation approach, we also present model-transformation rules for fine pointing granularity. We used them for semi-automatically generating a usual GUI for use with a mouse as well. So, we have been able to generate both GUIs semi-automatically from the same high-level discourse specification, and we compare them in this paper.

This flexibility in semi-automatic GUI generation is unique at our best knowledge. In particular, it is based on application-tailored device specifications.

Acknowledgements

This research has been carried out in the CommRob project (http://www.commrob.eu) and is partially funded by the EU (contract number IST-045441 under the 6th framework programme).

References

[1] G. Mori, F. Paterno, and C. Santoro, "Design and development of multidevice user interfaces through multiple logical descriptions," *IEEE Transactions on Software Engineering*, vol. 30, no. 8, pp. 507–520, 8 2004.

[2] F. Paternò and C. Santoro, "One model, many interfaces," in *Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces (CADUI 2002)*, 2002, pp. 143–154.

[3] F. Paternò and F. Giammarino, "Authoring interfaces with combined use of graphics and voice for both stationary and mobile devices," in *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI 2006)*. New York, NY, USA: ACM, 2006, pp. 329–335.

[4] X. Xiao, Q. Luo, D. Hong, H. Fu, X. Xie, and W.-Y. Ma, "Browsing on small displays by transforming web pages into hierarchically structured subpages," *ACM Transactions on the Web*, vol. 3, no. 1, pp. 1–36, 2009.

[5] B. Collignon, J. Vanderdonckt, and G. Calvary, "Model-driven engineering of multi-target plastic user interfaces," in *Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems (ICAS 2008)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 7–14.

[6] W. O. Galitz, *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[7] P. Parhi, A. K. Karlson, and B. B. Bederson, "Target size study for one-handed thumb use on small touchscreen devices," in *Proceedings of the 8th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI 2006)*. New York, NY, USA: ACM, 2006, pp. 203–210.

[8] Y. S. Park, S. H. Han, J. Park, and Y. Cho, "Touch key design for target selection on a mobile phone," in *Proceedings of the 10th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI 2008)*. New York, NY, USA: ACM, 2008, pp. 423–426.

[9] G. T. Bender, "Touch Screen Performance as a Function of the Duration of Auditory Feedback and Target Size," Ph.D. dissertation, Wichita State University, 1999.

[10] J. Falb, H. Kaindl, H. Horacek, C. Bogdan, R. Popp, and E. Arnautovic, "A discourse model for interaction design based on theories of human communication," in *CHI '06 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM Press, 2006, pp. 754–759.

[11] C. Bogdan, J. Falb, H. Kaindl, S. Kavaldjian, R. Popp, H. Horacek, E. Arnautovic, and A. Szep, "Generating an abstract user interface from a discourse model inspired by human communication," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS-41)*. Piscataway, NJ, USA: IEEE Computer Society Press, January 2008.

[12] C. Bogdan, H. Kaindl, J. Falb, and R. Popp, "Modeling of interaction design by end users through discourse modeling," in *Proceedings of the 2008 ACM International Conference on Intelligent User Interfaces (IUI 2008)*, Maspalomas, Gran Canaria, Spain, 2008.

[13] R. Popp, J. Falb, E. Arnautovic, H. Kaindl, S. Kavaldjian, D. Ertl, H. Horacek, and C. Bogdan, "Automatic generation of the behavior of a user interface from a high-level discourse model," in *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences (HICSS-42)*. Piscataway, NJ, USA: IEEE Computer Society Press, 2009.

[14] S. Kavaldjian, J. Falb, and H. Kaindl, "Generating content presentation according to purpose," in *Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics (SMC2009)*, San Antonio, TX, USA, Oct. 2009.