UI-GEAR: User Interface Generation prEview capable to Adapt in Real-time

Jenny Ruiz[1], Estefanía Serral[2] and Monique Snoeck[2]
[1]*University of Holguín, XX Aniversary Avenue, Holguin, Cuba*
[2]*KU Leuven, Naamsesstraat 69, Leuven, Belgium*

Keywords: Abstract User Interface Model, Feature Model, Model-driven Engineering, Software Development Method, User Interface Development Environment, User Interface Generation Preview.

Abstract: User Interface (UI) preview enables UI developers to preview and see the current User Interface before being generated. Despite the many advantages that UI preview could offer, it is not provided by current UI development environments. This paper presents UI-GEAR, a UI generation preview capable to adapt in real-time. UI-GEAR is developed within the MERODE method, a model-driven engineering approach capable to generate a fully functional system prototype from its specification in models. UI-GEAR extends MERODE with a UI development environment that enables developers to play with generation options and to straightforwardly and in real-time visualize the consequences of their choices on the UI to be generated, thus providing them with immediate guidance and design flexibility. We have carried out an experiment with developers with novel experience on designing UIs that demonstrates the advantages of this approach.

1 INTRODUCTION

User Interface (UI) generation can be achieved through *Model-based UI Development* (MBUID) and *Model-Driven Engineering of UIs* (MDEUI), two similar and very well-known approaches that share the same principle (Aquino, Vanderdonckt, Panach, & Pastor, 2011): the target UI characteristics are captured in models, which are then used to generate the UI code (Calvary et al., 2003). The main difference is that MDEUI involves explicit *Model-to-Model* (M2M) and *Model-to-Code* (M2C) transformations that are themselves compliant with a particular metamodel (Schaefer, 2007), while in MBUID models can be used for different purposes but without explicitly defined transformations.

Both approaches provide many advantages over the manual coding of UIs or their coding through graphical programming environments. The use of abstract models allows a UI to be designed by using concepts that are closer to the UI domain and not bound to the underlying implementation technology. This makes UI development simpler, and the resulting UI error-free and easier to maintain, among other benefits. However, the UI generation process, whatever the method is, suffers from several shortcomings such as: lack of predictability, mismatch between selected design/generation options and the generated results, lack of knowledge on how to decide appropriate design options, and a high learning curve (Aquino et al., 2011), (Daniel et al., 2007). Also, UI generation is rarely integrated with the development of the underlying application.

This paper presents the MDEUI approach UI-GEAR, a UI generation pr<u>e</u>view capable to <u>a</u>dapt in <u>r</u>eal-time. UI-GEAR addresses the mentioned shortcomings by being integrated into an application development environment and enabling developers to preview the results of a UI generation before actually generating the UI. Generally speaking, preview enables users to see the final results of this process before actually completing it. For instance, *contents preview* is the preview applied to the process of contents submission: it enables user to see current data they have already entered and see the final results of thid data submission before actually submitting the data. Another example is *Print preview*, which enables users to see the pages they are about to print in a "What you see is what you get" (WYSIWYG) manner before actually printing them. In Human-Computer Interaction (HCI), *UI generation preview* enables UI developers and end users to see the currently designed UI and how it will look and feel before actually completing

Ruiz J., Serral E. and Snoeck M.
UI-GEAR: User Interface Generation prEview capable to Adapt in Real-time.
DOI: 10.5220/0006115402770284
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 277-284
ISBN: 978-989-758-210-3

the UI development.

The reminder of this paper is structured as follows: Section 2 examines the related work about preview as a general feature and as a particular application in UI generation. Section 3 describes UI-GEAR, the MDE process in which it is implemented and its supporting tool and provides an example to illustrate the potential benefits of UI-GEAR. Section 4 reports on a user experiment conducted to assess UI-GEAR and Section 5 concludes the paper.

2 RELATED WORK

2.1 Content´s Preview

Beyond general contents submission and printing, contents preview has been subject to some attention, always highlighting to what extend previewing the results may improve the final results.

Programing In the Model (Maleki, Woodbury, & Neustaedter, 2014) is a prototype Computer-Aided Design (CAD) system assisting in designing code. It promotes Lookahead as preview feature showing programmers new or modified model elements and lines of code highlighted in purple in the full code, consequently to their options. As a design exploration tool, Lookahead was not very successful. Instead of seeing the purple preview model as an alternate state, participants paid more attention to what objects in the model or lines of code in the syntax were purple. Preview was preferred to see what was being affected by the options they chose. In this study, preview is recognized more as a debugging and error prevention tool than a design exploration tool.

PreSense (Rekimoto et al., 2003) provides users with a preview for command execution, providing a significant benefit when it is not possible to undo a command. This preview helps users to see what will occur next. It is also helpful when the command assignment of the keypad dynamically changes.

SideViews (Terry and Mynatt, 2002) provides users with a preview for commands with parameters, that require direct user input (such as mouse strokes for a paint program), and for computationally-intensive commands. An example is executing a “Bold” command on a text selection: a preview is generated while the cursor is hovering the corresponding icon. If another icon is hovered, e.g., the “Underline” icon, the “Underline” command will be previewed. (Kristensson and Zhai, 2007) report that previewing command strokes with pen-based gestures could be effective to improve the command's predictability and the guidance for command-gesture correspondence.

These approaches show the importance and benefits of a preview functionality, however, none of them are created for previewing a UI. The approach closest to UI-GEAR is *PreSense* (Rekimoto et al., 2003), which shows a preview of what will happen if a command is executed. Similarly, UI-GEAR shows developers the effect of the chosen options on the final UI before actually generating it.

2.2 User Interface Generation Preview

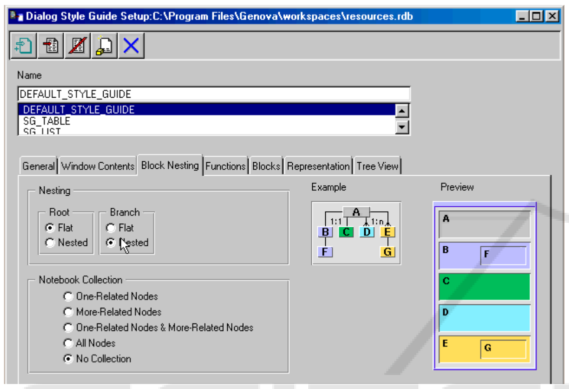
MBUID emerged in the early 90´s to support UI development by basing it on a set of abstractions (models). The most relevant Model-Driven Engineering UI generation approaches are as follows.

(Raneburger, 2010) introduces a semi-automatic approach based on communication models involving the designer during the generation process. It provides suggestions and automatic UI optimization based on former design decisions and heuristics. The transformation template approach proposed in (Aquino et al., 2010) makes the MDEUI process more explicit and flexible gathering some generation options in reusable templates. (Bacha, Oliveira, & Abed, 2011) present a MDEUI environment that automatically generates various UIs focused on content personalization. (Gaulke & Ziegler, 2015) present an approach that incorporates UI relevant metadata to an ontological domain model to be reused during the UI generation process.

MANTRA (Botterweck, 2007) automatically generates UIs for different computing platforms. Graceful degradation (Florins et al., 2006) emi-automatically generates many UIs for more constrained platforms (e.g., tablet, smartphones) starting from a UI for a less constrained platform (typically, a desktop UI) based on generation options. (Raneburger et al., 2012) propose an automated layout approach for model-driven window / icon / menu / pointing device UI generation. This approach allows specifying layout parameters in device-independent transformation rules. (Alonso-Ríos et al., 2014) present an environment that automatically generates various UIs for different smartphones exhibiting different capabilities.

FlowiXML (Guerrero et al., 2008) automatically generates various UIs for a workflow information system based on UI patterns derived from workflow patterns. JustUI (Molina et al., 2002) automatically generates a UI for HTML, Java, and C++ starting from the same conceptual models by applying so-called conceptual UI patterns.

Although some of the previous approaches offer different design choices, no UI generation preview of any kind is supported, being a common general drawback of current MBUID and MDEUI. The only approach that provides preview options is Genova (Genera AS & Lysaker, 2004) (see Figure 1), a commercial tool for the automatic generation of interactive applications, including their UI, based on a UML class diagram and a style guide. While defining the style guide, the developer is prompted with various generation options (e.g. windows contents, block nesting) that are reflected in a UI preview. However, this preview remains abstract and indicative: it is not applied to the actual UI being generated and it visualizes only one option at a time.


Dialog Style Guide Setup:C:\Program Files\Genova\workspaces\resources.rdb

Name: DEFAULT_STYLE_GUIDE

DEFAULT_STYLE_GUIDE
SG_TABLE
SG_LIST

General | Window Contents | Block Nesting | Functions | Blocks | Representation | Tree View

Nesting
Root: [✓ Flat] Nested
Branch: Flat [✓ Nested]

Notebook Collection
One-Related Nodes
More-Related Nodes
One-Related Nodes & More-Related Nodes
All Nodes
[✓ No Collection]

Figure 1: UI Preview in Genova software.

3 UI-GEAR

UI-GEAR has been developed within the context of MERODE, a method for enterprise information system engineering (Snoeck, 2014). MERODE allows an enterprise system to be specified using an object-oriented conceptual domain model that is platform independent and sufficiently complete to automatically generate the system's code from it. The model is composed of a Class Diagram to capture the domain classes, an Object-Event Table (OET) to capture interaction aspects, and Finite State Machines (FSMs) to capture enterprise object behaviour. The domain model is linked to a business process model capturing the user´s tasks and workflows. The supporting JMermaid tool (Sedrakyan and Snoeck, 2013) allows modelling the different views of a software system, managing consistency between them automatically and is able to automatically generate a full functional prototype including feedback features that help developers obtaining the prototype. The generated prototype provides a default and non-designable UI. In order to provide UI design flexibility and UI modelling guidance, MERODE has been extended with the UI-GEAR component.

3.1 UI-GEAR Component

UI-GEAR augments the MERODE models with a presentation model consisting of three different views: 1) the *General aspects*, which collects the name of the application and other information to be shown in the title, 2) the *Window aspects*, which captures preferences about how widgets will be shown, and 3) the *Input aspects* to capture the preferences related to how the user will input information into the application, like how the components for attribute input will be generated or the way the to-be-selected associated objects will be shown. The feature model (Benavides et al., 2010) in Figure 2 clarifies the different design options that constitute the *Window* and *Input aspects* of the presentation model. In the UI-GEAR tool, each view of the presentation model is presented in a different tab, as advised in the usability design pattern presented in (Van Welie & Trætteberg, 2000) (see Figure 2).

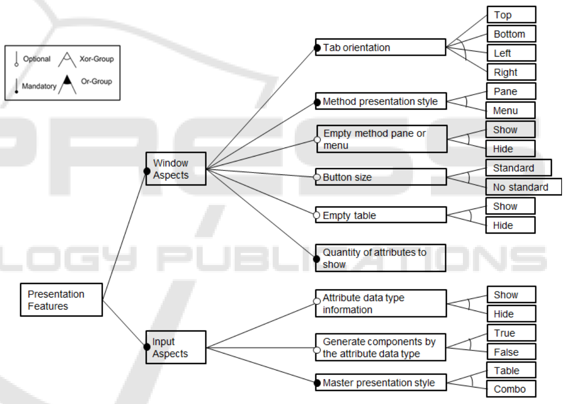
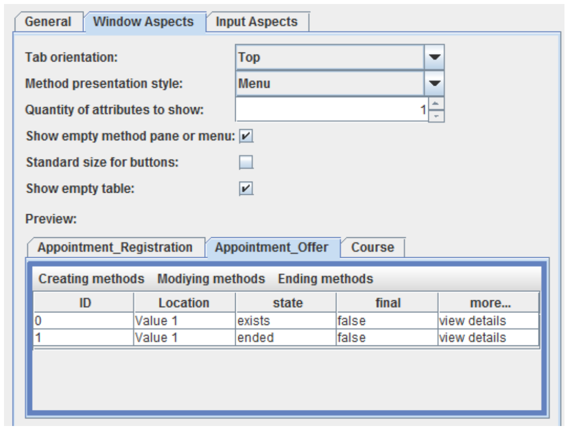


Figure 2: UI-GEAR Feature model.

At the bottom of the *Window* and *Input aspects*, UI-GEAR offers a preview of the to-be-generated UI. This preview automatically and instantaneously adapts to changes in the selected options, hence explicitly visualizing how the generated UI will look like. This allows a developer to trace changes from a model to their effects by testing several "what-if" scenarios. In addition, developers do not need to explicitly specify the values for all the design options of the presentation model as UI-GEAR also offers default options that can be directly used. An important difference and added value compared to Genova, is that UI-GEAR´s preview has the same layout as the to-be-generated UI, therefore also allows combining several generation options at once, enabling developers to assess the result of not just one option at a time, but of the overall generation process. Figure 3 shows the presentation model dialog with the three tabs, the *Windows aspects* tab and its preview being visible.



Tab orientation: Top
Method presentation style: Menu
Quantity of attributes to show: 1
Show empty method pane or menu: ✓
Standard size for buttons: ☐
Show empty table: ✓
Preview:

ID	Location	state	final	more...
0	Value 1	exists	false	view details
1	Value 1	ended	false	view details

Figure 3: Presentation Model and UI Preview.

To clarify the working of UI-GEAR, we illustrate how the different design option of the *Window aspects* determine the design of the UI. The prototype's user interface consists of one tab per application class. The user has to define the *tab orientation* for the generated prototype: at the top, bottom, left, or right of the main window. Each application class' tab always shows the id of the object, its state and whether it is final or not. The *quantity of attributes* determines how many of the regular attributes should be shown. If the class has less attributes than this quantity, then all its attributes will be shown. The *method presentation style* for an application class's window can be a pane with buttons or a menu. The methods are classified as *Creating*, *Modifying* and ·*Ending* methods. If the method presentation style is pane, a pane is generated for each kind of method, and inside the pane a button is generated for each method. Otherwise, the methods are in three menus following the same classification (See Figure 3). *Show empty method pane or menu* determines what to do if some type of method is not present (e.g., there are no *Ending* methods). The buttons for the methods can have *standard size* or not. Finally, the UI designer should indicate whether or not an *empty table* should be shown or not for classes without instances.

In the *Input aspects* tab the UI developer chooses whether or not to show *attribute data type information* for each attribute and whether or not to generate the *component* for the attributes input *according to the data type* or not. Finally, if the class has associations with other classes, the *master presentation style* must be chosen to define if this association is shown as a table or a combo.

The concrete selection of values for the *Window aspects* shown in Figure 3 is: tab orientation: Top, method presentation style: Menu, quantity of attributes to show: 1, empty method pane or menu: show, button size: not standard, empty table: show.

To realize the preview feature, UI-GEAR is implemented according to the *Java reflection principle*: an internal representation of the feature model is maintained that is parsed in real-time and interpreted so as to dynamically generate the corresponding preview based on the chosen options: each time a new option is decided, the internal representation is instantaneously updated and so is the UI example. This gives the possibility to validate and verify user requirements, and reduces the time and effort required to implement the UI.

3.2 Generation Process Overview

Once the preview shows an interface design that satisfies the developer, the code generation of JMermaid can be executed to automatically generate the final UI according to the selected features in the presentation model. JMermaid´s process generates the full application form the models, shown in Figure 4. Based on the conceptual domain model and the presentation model, a *Model to Model* (M2M) transformation generates an abstract user interface model, which is the expression of a UI in terms of interaction units without making any reference to the implementation.

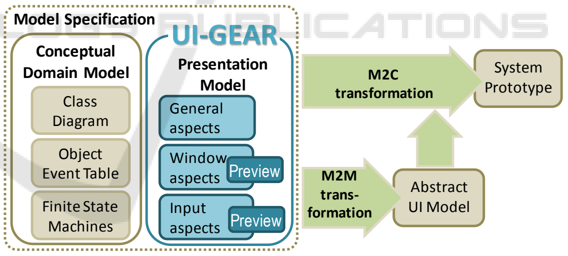


Figure 4: Prototype generation process in JMermaid.

Then, *Model to Code* (M2C) transformations generate the system prototype from the domain model and the abstract UI model. The final UI is generated from the abstract UI model, but also from the three domain model views, which are necessary to generate the persistence and event handler layers.

The *Model to Code* (M2C) transformations are built using Java and Apache's Velocity Templates Engine (http://velocity.apache.org). The transformations can work with a minimal model containing at least one class from which additional default elements are automatically generated by JMermaid (Sedrakyan et al., 2013).

In order to create the UI java code, six templates to display the system information as indicated in the presentation model are defined. These templates are dynamically populated based on the domain model and on the options selected in the presentation model and then used to generate the code for the graphical components of the application. A fragment of the template used to generate the java code of the interface shown in the preview of Figure 3 is shown below. Note that this template is already populated with the values indicated in the presentation model. Lines 1-6 ensure that methods can be generated as a menu or as a pane with buttons (default) and lines 7-15 ensure that for each associated class, objects can be generated as tables or combos. After the modelling activity and the code generation step, developers can still change the models to improve the quality of the generated prototype.

```
1   #if ($methodType == "Menu")
2   import ui.tabs.lists.ObjectList
3   MenuWindow;
4   #else
5   import ui.tabs.lists.ObjectListWindow;
6   #end
7   #foreach ($abstractDataIUM in
8   $abstractCompound.abstractDataIUMs)
9   #if ($masterType == "Table")
10  import ui.tabs.tables.${abstract
11  DataIUM.abstractDataIULabel}_Table;
12  #else
13  import ui.tabs.tables.${abstract
14  DataIUM.abstractDataIULabel}_Combo;
15  #end
```

3.3 Example

Figure 5 shows the class diagram of an appointment system at the university. Teaching assistants can publish appointment offers for a specific course. Students can create a registration to one of the offers.

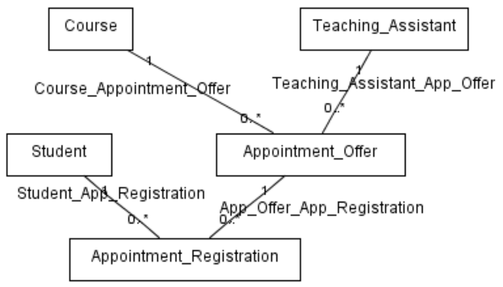
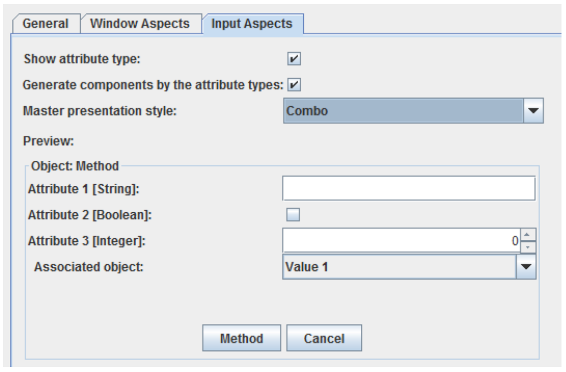


Figure 5: Student's appointment class diagram.

Next to the class diagram, the OET and FSMs capture non-default behaviour such as the states in the lifecycle of an appointment (available, closed, …). The presentation model collects information about the user's preferences. The developer can play with different options and preview the result.

Figure 6 shows an example of the chosen options for the *Input aspects* of the presentation model and the corresponding preview. The attribute's data types are shown, the components are generated according to this data type, and the associated objects are presented in a combo box.



Show attribute type: ✓
Generate components by the attribute types: ✓
Master presentation style: Combo
Preview:
Object: Method
Attribute 1 [String]:
Attribute 2 [Boolean]:
Attribute 3 [Integer]: 0
Associated object: Value 1

Figure 6: Input aspects of the Presentation model.

After this step, the developer can generate the prototype simply by choosing the location to save the generated code. First, the abstract UI model is generated, then, the code of the functional prototype. These two steps are transparent to the developer, who only sees the final result: the prototype. After testing the prototype, the developer can still come back to the UI options and regenerate another prototype: if the selected options of the feature model change, the UI changes accordingly. As for any MDE approach, the final result is subject to the problem of *round-trip engineering*: if any manual change is brought the generated UI, it therefore falls outside the scope of the transformation process from the abstract UI to the final UI, thus introducing some inconsistency between the initial model and the generated prototype that has been modified.

4 USER EXPERIMENT

We performed an experiment to evaluate UI-GEAR from the perspective of perceived usability by unexperienced developers. Several questionnaires have been used and reported in the literature for assessing the perceived usability of interactive systems, such as QUIS (Elkoutbi et al., 1999), SUS (Brooke, 1996) , and CSUQ (Lewis, 1993). We used the CSUQ (Computer System Usability Questionnaire), which was developed at IBM (Lewis, 1993). The questionnaire has been considered a reliable measure of overall satisfaction with an interface (McArdle and Bertolotto, 2012): it has been empirically proved that the IBM CSUQ questionnaire benefits from a 0.94 correlation with usability of the assessed system. Completing the CSUQ allows participants to provide an overall evaluation of the system they used. This questionnaire is composed of 19 items. The items are 7-point graphic scales, anchored at the end points with the terms "Strongly disagree" for 1 and "Strongly agree" for 7.

4.1 Method

Participants. We conducted a user experiment involving 12 participants who were recruited from the University of Holguin, Cuba. The participants have a background in informatics engineer / computer science. The average age of the sample was 29.42 years and the standard deviation was 3.68. The participants were software developers and university professors with novel experience on designing UIs. No participant has prior knowledge or exposure to UI-GEAR.

Task and procedure. In the first phase of the test, JMermaid, and in particular UI-GEAR were presented to the participants with an explanation of its use. Each participant was asked to carry out a set of tasks in JMermaid. Using an already developed conceptual domain model (shown in Figure 5) as starting point, they played with the different options of the presentation model to performed the following tasks: 1) create a presentation model using UI-GEAR and 2) generate the prototype using UI-GEAR. For the first task the participants needed to fill the prototype´s name and the information about the person who created it; create a prototype with tabs to the left, pane with buttons, three attributes in each table, components according to the type of the attribute, and check in the preview if the UI will be generated as expected (if not, make the necessary modifications to the previous values). Once the desired characteristics were obtained, (task 1) they generated a working prototype according to the values they gave for each part of the presentation model (task 2). After completing the tasks, the users were asked to fill the questionnaire for user-interaction satisfaction. During the sessions users were not allowed to ask questions to the evaluator.

4.2 Results and Discussion

The results from the CSUQ evaluations are presented in Table 1 in terms of mean with the standard deviation and the mode for each question of the CSUQ.

Table 1: CSUQ items and scores; range 1 (lowest) - 7 (highest).

CSUQ items	**Mean (std. dev.)**	**Mode**
1. Overall, I am satisfied with how easy it is to use this system	5.83 (0.72)	6
2. It was simple to use this system	5.92 (0.90)	5
3. I can effectively complete my work using this system	6.17 (0.83)	7
4. I am able to complete my work quickly using this system	6.33 (0.65)	6
5. I am able to efficiently complete my work using this system	5.92 (0.79)	6
6. I feel comfortable using this system	6.00 (0.85)	7
7. It was easy to learn to use this system	6.17 (0.72)	6
8. I believe I became productive quickly using this system	6.08 (0.51)	6
9. The system gives error messages that clearly tell me how to fix problems	6.33(0.89)	7
10. Whenever I make a mistake using the system, I recover easily and quickly	5.92 (0.79)	6
11. The information (such as online help, on-screen messages, and other documentation) provided with this system is clear	5.83 (1.03)	7
12. It is easy to find the information I needed	5.58 (0.79)	6
13. The information provided for the system is easy to understand	5.75 (0.62)	6
14. The information is effective in helping me complete the tasks and scenarios	6.00 (0.85)	5
15. The organization of information on the system screens is clear	6.08 (0.79)	6
16. The interface of this system is pleasant	5.75 (0.97)	6
17. I like using the interface of this system	5.75 (0.97)	5
18. This system has all the functions and capabilities I expect it to have	6.25 (0.62)	6
19. Overall, I am satisfied with this system	6.00 (0.85)	6

A first global observation is that the scores per item rank well above 5 on 7, indicating a very positive evaluation. We also observe that the highest mean values were obtained for the items 4, 9, and 18, while the lowest mean value was obtained for item 12. The mode represents what the majority of the participants score in the test. The mode of only three items was 5, while for all the other items the mode was 6 or 7.

We observed that item 12 about finding the information needed has the lowest score, even though the participants score this item well overall. The score is explained by the fact that some participants (with less experience) had doubts about which tab of the presentation model they should use to perform a subtask.
The cumulated histogram in Figure 7 summarizes the responses to the 19 CSUQ questions. The distribution of all the questions revealed that nobody disagrees with any question, and that for Q4, Q8 and Q18, the responses even were only agree and strongly agree.

Apart from the global item 19, the 18 items form three sub-scales, each of which measures a different component of (perceived) usability: System Usefulness (SYSUSE, items 1-8), Information Quality (INFOQUAL, items 9-15), and Interface Quality (INTERQUAL, items 16-18).

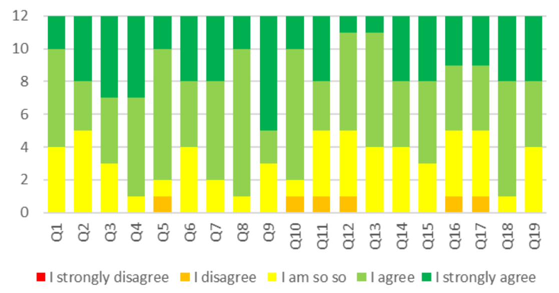


Figure 7: Distribution of participants' responses.

Table 2 shows the highest, lowest, mean and mode values for the mentioned sub-scales. (Perceived) System Usefulness and Interface Quality had the highest (6.05) and lowest (5.92) mean scores respectively. One can immediately note that these values are very close to each other. The values indicate a high level of satisfaction among the subjects regarding their perception of the usefulness of the tool.

Table 2: Values of CSUQ for the prototype.

	SYSUSE	INFOQUAL	INTER-QUAL	OVERALL
Highest	6.80	6.76	6.79	6.85
Lowest	5.31	5.10	5.04	5.15
Mean	6.05	5.93	5.92	6.00
Mode	6.00	6.00	6.00	6.00

The perceived usefulness is high: the users believe the system will enhance their performance and that the approach facilitates a presentation model to be created by showing its preview, which is also supported by the positive scores for the first 8 questions.

The information and interface quality have been also well appreciated, but sometimes there was a lack of details on where the information could be found. Question 19 about satisfaction suggests that UI-GEAR is positively perceived overall and provides the functionalities the developers expected.

5 CONCLUSIONS

This paper has presented UI-GEAR, a User Interface generation preview capable to adapt in real-time. UI-GEAR is an MDE environment that enables UI-developers playing with UI generation options and straightforwardly visualizing the consequences, thus providing them with guidance and flexibility in the design of the UI. For this purpose, a feature model structuring the generation options is parsed to show a UI preview at real-time.

UI-GEAR provides several important benefits for UI development. By showing the developer the results of the UI generation before actually obtaining it, errors, such as usability problems, can be detected and fixed early instead of waiting for the results of the generation. UI-GEAR also allows developers to customize the user interface generation on-the-fly, hereby eliminating the uncertainty regarding the final UI result. This enables the developer to focus on major UI aspects that need review.

UI-GEAR's preview facilitates alignment with developers' expectations and helps to improve developers' understanding of UI development. By presenting a preview of the UI before actually generating it, developers can discover differences between the actual user interface and what they hoped to obtain according to the chosen generation options, thus improving their understanding of the effects of generation options and reducing the learning curve.

The performed user experiment has demonstrated that developers find UI-GEAR very satisfactory in all areas for which the CSUQ accounts: usefulness, information quality, and interface quality. However, to assure generalization validity and assessment of the impact of the preview UI development, we plan to do further controlled experiments in the future.

In addition, another shortcoming of our approach is that only functional aspects of the UI are modelled: UI-GEAR is not focused on aesthetic appeal. For the moment, our approach only addresses the development of enterprise information systems in one language and one platform of use. However, since this approach relies on MDE, the generation of the system to other languages and platforms can be easily extended in future versions of the tool.

Finally, all features of a feature model are assumed to remain independent of each other: choosing a value for a feature does not affect the choice of other features. This assumption is not always valid in the real world: certain design options may impact one or many other features, thus making the interpretation of the feature model, and therefore the UI generation preview, more difficult. The feature model could be expanded for this purpose with constraints between feature values, but this would make the rendering engine more complex. This can be alleviated with JMermaid's current consistency checking mechanism to ensure that the options selected by the UI developer do not create inconsistencies.

ACKNOWLEDGEMENTS

The authors would like to thank Jean Vanderdonckt for his excellent feedback and the opportunity of visiting UCL. This research has been sponsored by the VLIR-UOS network program.

REFERENCES

Alonso-Ríos, D., Raneburger, D., Popp, R., Kaindl, H., & Falb, J. (2014). A User Study on Tailoring GUIs for Smartphones. In SAC´2014 (pp. 186–192). Conference Proceedings, Gyeongju: ACM Press, New York.

Aquino, N., Vanderdonckt, J., Panach, J. I., & Pastor, O. (2011). Conceptual modelling of interaction. In Handbook of Conceptual Modeling: Theory, Practice and Research Challenges (pp. 335–358). Book Section, Springer, Berlin.

Aquino, N., Vanderdonckt, J., & Pastor, O. (2010). Transformation templates: adding flexibility to model-driven engineering of user interfaces. In SAC´2010 (pp. 1195–1202). Conference Proceedings, Sierre: ACM Press, New York.

Bacha, F., Oliveira, K., & Abed, M. (2011). A model driven architecture approach for user interface generation focused on content personalization. In RCIS´2011 (pp. 1–6). Conference Proceedings, IEEE.

Benavides, B., Segura, S., & Cortés, A. R. (2010). Automated Analysis of Feature Models 20 Years Later: A Literature Review. Information Systems 35, 6, 615–636. Journal Article.

Botterweck, G. (2007). A model-driven approach to the engineering of multiple user interfaces. In Models in Software Engineering (pp. 106–115). Book Section, Springer.

Brooke, J. (1996). SUS: A Quick and Dirty Usability Scale. In Usability Evaluation in Industry. Book Section, London: Taylor & Francis.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. Interacting with Computers, 15(3), 289–308. Journal Article.

Daniel, F., Matera, M., Yu, J., Benatallah, B., Saint-Paul, R., & Casati, F. (2007). Understanding ui integration: A survey of problems, technologies, and opportunities. Internet Computing, IEEE, 11(3), 59–66. Journal Article.

Elkoutbi, M., Khriss, I., & Keller, R. K. (1999). Generating user interface prototypes from scenarios. In Requirements Engineering, 1999. Proceedings. IEEE International Symposium on (pp. 150–158). Conference Proceedings, IEEE.

Florins, M., Montero, F., Vanderdonckt, J., & Michotte, B. (2006). Splitting Rules for Graceful Degradation of User Interfaces. In AVI´2006 (pp. 59–66). Conference Proceedings, ACM Press.

Gaulke, W., & Ziegler, J. (2015). Using profiled ontologies to leverage model driven user interface generation. In SIGCHI´2015 (pp. 254–259). Conference Proceedings, ACM.

Genera AS, & Lysaker. (2004). GENOVA V8.0 User Guide (Report).

Guerrero, J., Vanderdonckt, J., & Gonzalez, J. (2008). FlowiXML: a Step towards Designing Workflow Management Systems. Journal of Web Engineering, 4(2), 163–182. Journal Article.

Kristensson, P. O., & Zhai, S. (2007). Command strokes with and without preview: using pen gestures on keyboard for command selection. In CHI'2007 (pp. 1137–1146). Conference Proceedings, San Jose: ACM Press, New York.

Lewis, J. R. (1993). IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use (Report) (Vol. 12). Boca Raton, Florida.

Llavador, M., & Canós, J. H. (2007). A Framework for the Generation of Transformation Templates. In ECDL'2007. Lecture Notes in Computer Science (Vol. 4675, pp. 501–504). Conference Proceedings, Springer.

Maleki, M. M., Woodbury, R. F., & Neustaedter, C. (2014). Liveness, Localization and Lookahead: Interaction Elements for Parametric Design. In DIS'2014 (pp. 805–814). Conference Proceedings, Vancouver: ACM Press, New York.

McArdle, G., & Bertolotto, M. (2012). Assessing the application of three-dimensional collaborative technologies within an e-learning environment. Interactive Learning Environments, 20(1), 57–75. Journal Article.

Molina, P. J., Meliá, S., & Pastor, O. (2002). Just-UI : A User Interface Specification Model. In CADUI'2002 (pp. 63–74). Conference Proceedings, Dordrecht: Kluwer Acad. Pub.

Raneburger, D. (2010). Interactive Model Driven Graphical User Interface Generation. In EICS '10. Conference Proceedings, 321-324.

Raneburger, D., Popp, R., & Vanderdonckt, J. (2012). An Automated Layout Approach for Model-Driven WIMP-UI Generation. In EICS '12. Conference Proceedings.

Rekimoto, J., Ishizawa, T., Schwesig, C., & Oba, H. (2003). PreSense: Interaction Techniques for Finger Sensing Input Devices. In UIST'2003 (pp. 203–212). Conference Proceedings, Vancouver: ACM Press.

Schaefer, R. (2007). A Survey on Transformation Tools for Model Based User Interface Development. In Proceedings of the HCI´International (Vol. 4550, pp. 1178–1187). Conference Proceedings, Beijing: Springer.

Sedrakyan, G., & Snoeck, M. (2013). A PIM-to-Code requirements engineering framework. In Modelsward´2013 (pp. 163–169). Conference Proceedings.

Snoeck, M. (2014). Enterprise Information Systems Engineering: The MERODE Approach. Book, Springer.

Terry, M., & Mynatt, E. D. (2002). Side Views: Persistent, On-demand Previews for Open-Ended Tasks. In UIST'2002 (pp. 71–80). Conference Proceedings, ACM Press, New York.

Van Welie, M., & Trætteberg, H. (2000). Interaction patterns in user interfaces. In 7th. Pattern Languages of Programs Conference (pp. 13–16). CONF.