A Fully Implemented Didactic Tool for the Teaching of Interactive Software Systems

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

Keywords: Abstract User Interface Model, Presentation Model, Feature Model, Model-Driven Engineering, Software Development Method, User Interface Design, User Interface Generation.

Abstract: User Interface (UI) design and software engineering complement each other to develop useful and usable interactive software systems. However, the body of knowledge for the development of an application and for the design of its UI are not always well integrated. The problem starts in the education of both subjects, which are normally taught independently of each other. Although an integrative teaching approach can significantly contribute to the development of better interactive software systems, there is a lack of concrete and proven approaches for such way of teaching. This paper presents a fully functional didactic tool for filling this gap. This tool provides the learner with feedback about how to develop an application and how to design a proper UI for it. Applying Model Driven Engineering principles, the tool automatically generates a working prototype of the interactive software system from its specification models, allowing the learner to try out the final application and validate the requirements. An experiment with novice developers demonstrates the advantages of this didactic tool.

1 INTRODUCTION

Over the last years, the development of useful, usable interactive software systems has become a key aspect (Akiki, Bandara and Yu, 2015; Hentati et al., 2016). The fields of User Interface (UI) design and software engineering complement each other to reach that goal. However, application development and UI design are not always well integrated. There is a gap between the two communities, as each focuses primarily on its own field (Seffah, Gulliksen and Desmarais, 2005; da Cruz and Faria, 2009). As pointed by (Meixner, Paternò and Vanderdonckt, 2011), there is a lack of harmonization between UI and application design, with both communities largely neglecting the relation to other software views.

The problem starts already in education: UI design and application development are normally taught in an isolated way, while the link between both should instead be made very explicit: the integration between UI design and application development can significantly contribute to the development of even better systems (Meixner, Paternò and Vanderdonckt, 2011). Specifically, there is a need of integrated teaching support to foster the understanding of the relationship between functional aspects and the UI.

At the same time, UI design has been considered as a difficult process (Nguyen and Rahman, 2016; Sboui and Ayed, 2016). Ease of use is one critical factor to be taken into consideration for a tool to be used by novice designers (Dehinbo, 2011). Different tools have been developed to ease the teaching of either software engineering or the teaching of UI design. However, very few of them provide (some level of) integration of both, and these tools are difficult to use and not tailored to learners.

This paper presents a fully-implemented and integrated simulation tool for the teaching of UI design and application development at once. With this tool, the learner can describe the interactive software system requirements using conceptual models. With a single click, the tool automatically generates a working prototype of the described system in an integrated environment, where the learner can test the system against its requirements. In addition, the tool provides two kinds of feedback during the design of the system: 1) feedback facilitating tracing the application's behaviour back to its origin in the underlying conceptual models; and 2) feedback related to the UI design,

Ruiz, J., Serral, E. and Snoeck, M.
A Fully Implemented Didactic Tool for the Teaching of Interactive Software Systems.
DOI: 10.5220/0006579600950105
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 95-105
ISBN: 978-989-758-283-7

giving the learner clues according to design principles for the UI functional aspects and helping understanding the relationship between functional aspects and the UI.

An experimental evaluation demonstrates this tool's effectiveness, its ease of use, and its level of integration towards the development of interactive software systems.

The remainder of this paper is structured as follows: Section 2 examines the related work on educational tools to teach UI design and application development. Section 3 describes the integrated learning tool for interactive software systems. Section 4 presents an evaluation of the developed tool. Section 5 discusses the limitation of the tool and Section 6 concludes the paper.

2 RELATED WORK

From the perspective of integration, a number of Model Driven Engineering (MDE) approaches support UI design while at the same time giving support for integration with the application. MASP (Feuerstack et al., 2008) has a service model which connects backend services to application tasks, but it is still the developer who manually needs to make the link with the application. LIZARD (Marin et al., 2015) has a data service model which is used to populate the UI controls. Like in MASP, the developer should provide manually the link with the rest of the application. Similarly, Dygimes (Coninx et al., 2003) allows defining the link to the application logic by means of operations invoking web services that are linked to the application. Also WAINE (Delgado et al., 2016) generates web applications where the components of the UI have to be defined at the highest abstract level, which is a difficult task.

Examples of Model Driven Architecture (MDA) approaches for supporting application development are OptimalJ (London and University of York, 2003) and AndroMDA (www.andromda.org), which are code generation frameworks that generate fully deployable applications. They generate the links between all the layers, including a default presentation layer, but without providing support to tailor the UI.

ArcStyler (www.interactive-objects.com) offers partial integrated support. It supports application development and has a mechanism for designing sequences in the UI. However, the developer needs to write the code for the integration of the different layers. OO-Method (Molina and Pastor, 2004; Pastor and Molina, 2007) provides complete, integrated support for the UI design and application development but also requires detailed models which makes it difficult to use by junior developers.

An important shortcoming of the previous approaches is that the core focus is not teaching support and therefore they are difficult to be used by learners. Some of the previous approaches offer design choices for the application and/or its UI but they do not help the novice learner by providing feedback. The approaches are not easy to use either. Ideally, technical hurdles should be avoided and it should be possible to generate the UI code and its integrated application code through a single click. Current approaches need different tools (e.g MASP and Dygimes have two and three tools respectively) or take models from external tools and require additional transformation steps (e.g. AndroMDA).

There is a limited number of approaches that focus on supporting the teaching of UI design. Barret proposes a hypertext module which presents interface design principles with examples of good and bad UIs (Barrett, 1993). The tutorial includes the explanation of using metaphors, input devices and evaluation issues. (Sutcliffe, Kurniawan and Shin, 2006) propose a multimedia design advisor tool. The tool gives recommendations about which media is appropriate (according to the information type) with examples.

There are also tools for teaching application development. For instance, the JMermaid tool, connected to the enterprise engineering MERODE method (Snoeck, 2014), allows generating a full functional application with a single click while embedding in the application the feedback explaining the application's behaviour by referring to the models it was generated from (Sedrakyan and Snoeck, 2013). The JMermaid tool has been successfully validated for teaching conceptual modelling for more than 5 years (Sedrakyan, Snoeck and Poelmans, 2014). However, this environment generates a default UI and does not offer support for teaching UI design.

While the presented initiatives have their merits, a proven approach to support the teaching of UI design and application development in an integrated way is lacking. Additionally, there seems to be no teaching support on how to design UIs based on practical approaches used in industry or on academic approaches for UI design. Only (Barrett, 1993) is explicitly based on design principles, but it does not show how the application of the principles affect the UI design.

3 INTEGRATED SUPPORT FOR INTERACTIVE SOFTWARE

To support the teaching of interactive software systems, this paper proposes an integrated tool that allows the co-design of an application and its UI.

In previous work we presented a first design of an extension to the JMermaid tool, provided by the MERODE method, to provide support for UI design. In line with the principles of design research (Recker, 2012), this first proposal was tested and then improved based on the observed shortcomings. This paper presents the extended and improved design, its implementation and its evaluation. First, the MERODE method and JMermaid tool are introduced. Then, the extensions to the tool, the generation process and how the teaching support is provided are explained.

3.1 MERODE Method

The MERODE method defines a conceptual domain model that is platform independent and sufficiently complete to automatically generate the application´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 (FSM) to capture enterprise object behaviour.

The MERODE method has its own proven conceptual modelling teaching environment, JMermaid, which allows the fast prototyping of a conceptual domain model (Sedrakyan and Snoeck, 2013). This tool has a MDE-based code generator that generates a fully functional prototype Java application out of the conceptual model (Sedrakyan and Snoeck, 2013). JMermaid has been successfully tested and validated for teaching conceptual modelling for more than 5 years (Sedrakyan, Snoeck and Poelmans, 2014).

The MERODE method does not provide support for UI design. JMermaid generates all the applications with a default UI: it always generates the same kind of representation format for the input and output services without giving options for designing the UI. There is no teaching support either for the UI design. And while the UI is automatically integrated with the application, there is no specific teaching support for the integration with the application development. By the fact that the possibility of changing the UI is lacking, understanding the link between the UI and the application development process is not actively supported. For a more complete analysis of limitations of JMermaid for supporting UI design the reader is referred to (Ruiz, Sedrakyan and Snoeck, 2015).

Without the possibility of visualising the behaviour of the application through a UI, checking the behaviour of the domain model would be really difficult for the learner: she would then have to simulate in her mind what would happen while interacting with the system. Without application logic, it would be difficult too, to test the functional aspects of the UI and the responses that the system would provide. It would also be difficult to find the missing functional aspects that the learner did not incorporate in the UI design.

The fact that the MERODE method starts with well-defined conceptual models gives the possibility of extending it with UI models to give integrated support. The UI and application models can be used to teach domain modelling, and at, the same time, the general idea behind UI design. The UI containers, widgets, etc. show the components that constitute an UI, how they are represented in the final UI and how these are related with the conceptual domain model and its behaviour. This integration helps the comprehension of the link between the UI and the application logic.

3.2 FENIkS

In order to provide an integrated approach for UI design and application development of interactive software systems, and to benefit from the built-in didactic support of JMermaid, we have developed an extension that we call *Feedback ENabled user Interface Simulation (FENIkS)*. The integration with application development requires basing the generation of the UI and the application code on the same set of integrated models.

The overall approach is shown in Figure 1.

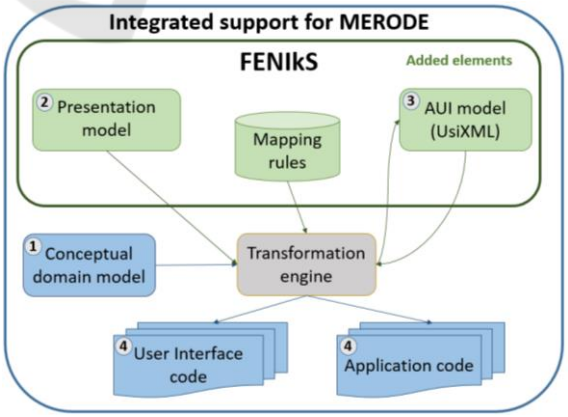

Figure 1: FENIkS generation architecture.

The input models required to generate a final application are: (1) a domain model capturing the application's functional logic, (2) a presentation model

capturing the characteristics of the interface components and user preferences. These two models are used to generate (3) an "Abstract User Interface" (AUI) which describes the UI in a technology-agnostic way. The AUI, the presentation and the domain model are then used to generate (4) the application code and the UI code. The transformations are automatically done by the tool and require no manual tailoring. The code generator for the whole integrated support of the MERODE method was built using the Java language and Velocity Templates Engine (http://velocity.apache.org).

The following subsections describe the presentation model and the generation of the AUI model, respectively.

3.2.1 Presentation Model

Previously, the MERODE tool generated a prototype where the user (only) received default output and input services. Default output services (or reports) are a list of instances of a single domain object type into one window (e.g. view the lists of an order) and viewing the details of one object with all its information (e.g. view details of one order) in another window. Default input service is triggering the execution of a business event (e.g. create an order). With FENIkS, extra output services can be created and configured to show specific information the user wants to see, for example, a list combining information from multiple domain objects (e.g. view a customer with all his orders).

The meta-model shown in Figure 2 shows how these additional reports are captured through the meta-object type Report and the associated meta-object types. The designer can select the objects to include in the reports, the attributes to be shown and the associations. For the associations to other domain objects it is also possible to select the attributes to be shown. The search can be configured in the same way.

Next to defining additional reports, the presentation model also captures preferences related to how elements of the UI should be configured. Rather than capturing these per individual report or input service, these are captured as general aspects that will be applied in a consistent way through the UI, for all the reports (default and non-default) and input services alike. This contributes to an important UI design principle of keeping a UI consistent.

The *Window aspects* capture the information about the static layout of the top level containers of the application. It reflects the visual features of the main application window and how information is displayed. Here it is possible to configure how the buttons for triggering services of the domain objects will be shown, where the options will be displayed, how the pagination will be, etc.

The *Input aspects* 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 (Ruiz, Serral and Snoeck, 2017). This is the place to select which widgets will be generated per object according (or not) to the attributes' data types.

Finally, the presentation model itself also stores attributes like the name of the application and other information to be shown in the application's title.

For all these aspects, FENIkS offers a set of generation options. This allows generating a family of prototypes with variations and commonalities in the way the information is presented and captured. The various UI generation options are captured as a feature model, as such model provides an adequate visual representation that is easy to manipulate (Benavides, Segura and Cortés, 2010).

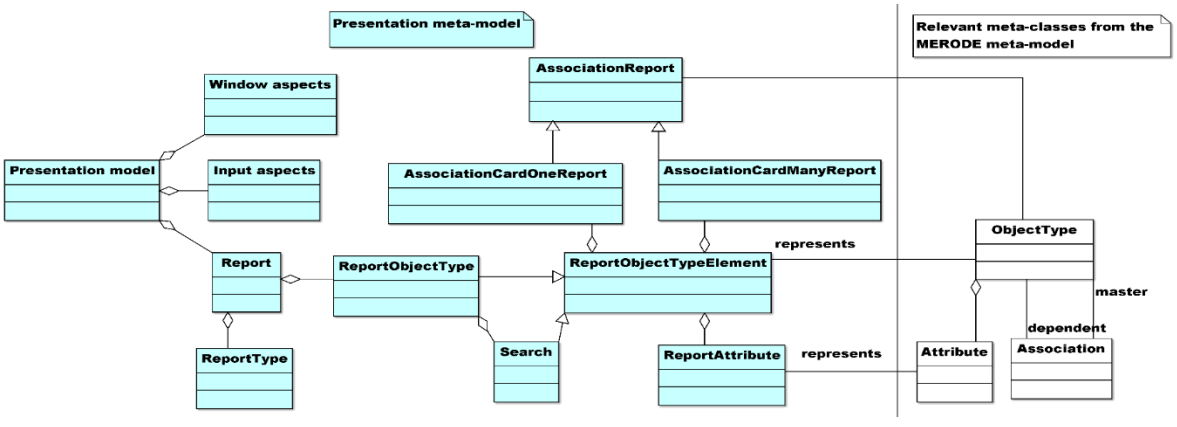

Figure 2: FENIkS presentation meta-model.

The feature model shows the relationships between a parent feature and its child features categorized as Optional or Mandatory depending on whether a child feature is optional or not, and 'Or' or 'Alternative (Xor)' depending on whether at least one or exactly one sub-feature must be selected.

Figure 3 and Figure 4 show the presentation features model of FENIkS. The node 'presentation model' is the root node and is mandatorily composed of the basic elements *Window aspects* and *Input aspects*, as also shown in the presentation meta-model.

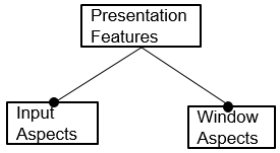

Figure 3: Feature Presentation model.

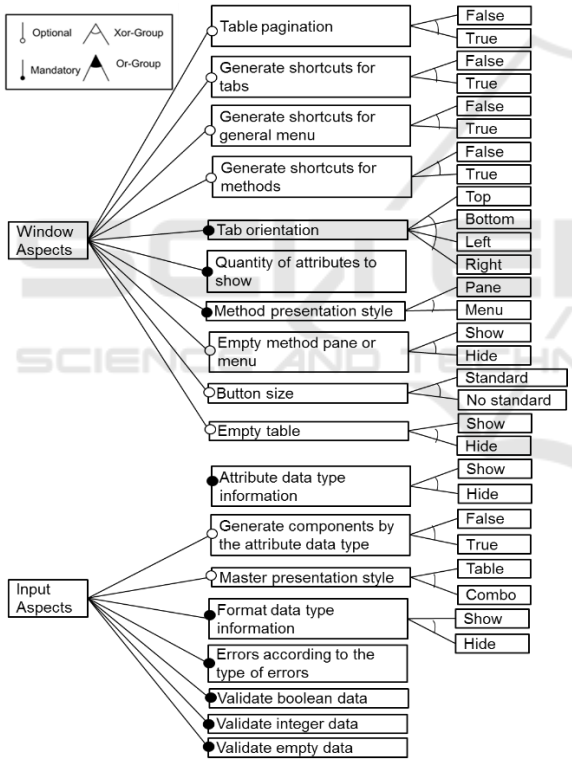

Figure 4: Feature Windows and Input aspects.

The majority of features of the *Window* and *Input aspects* are included for didactic purpose: they are used to generate the UI according to UI design principles. Examples of such features in the *Window aspects* are: Generate shortcuts for tabs and Method presentation style. Examples of such features in the *Input aspects* are: Attribute data type information and Validate Boolean data. See section 3.3.1 for a detailed explanation.

A number of additional features are included to give flexibility to the prototype generation process. Examples of such features are mainly in the *Window aspects*: Tab orientation, Empty method pane or menu, Table pagination, Quantity of attributes to show, Button size and Empty table. In the *Input aspects* there is only one feature to give flexibility, namely the Master presentation style.

3.2.2 Abstract User Interface Model

FENIkS allows the automatic generation of an AUI from the conceptual and the presentation models. The Abstract User Interface (AUI) is an expression of a UI in terms of interaction units without any reference to implementation and independent of any particular language.

AUIs are important for the development of applications for different contexts of use (Engel, Märtin and Forbrig, 2017) and also play an important role from the teaching perspective. The fact that the AUI represents the UI without taking into account any modality of interaction or platform helps in understanding the main principles behind UI generation. Thus, the generation of an AUI can be used to teach novice designers the general idea behind UI generation (i.e., the components that constitute an UI in abstract terms) while making the link between the UI and the underlying application logic.

The AUI meta-model of FENIkS is shown in Figure 5. It is based on the AUI meta-model of UsiXML (UsiXML documentation version 1.4), a User-Interface Modelling language proposed by (Limbourg et al., 2004). UsiXML stands for User Interface eXtensible Markup Language. The lower part of the figure (in light blue) shows the AUI meta-model, while the upper part (in white) presents the relevant concepts of the MERODE method meta-model and presentation meta-model connected to the AUI meta-model.

The AUI for the default output and input services will be populated by means of a model to model transformation from the conceptual domain model of the MERODE method.

The generation of the AUI for the user-specified output services takes as additional input the Report objects of the presentation model. In the AUI meta-model, the AbstractInteractionUnit is the basic unit for expressing the interaction in a recursive decomposition of an AUI in abstract terms. This decomposition can be related to one or many concepts like ObjectType, ReportObjectType, Attribute or ReportAttribute. An AbstractCompoundIU can be composed by one or many AbstractInteractionUnit.

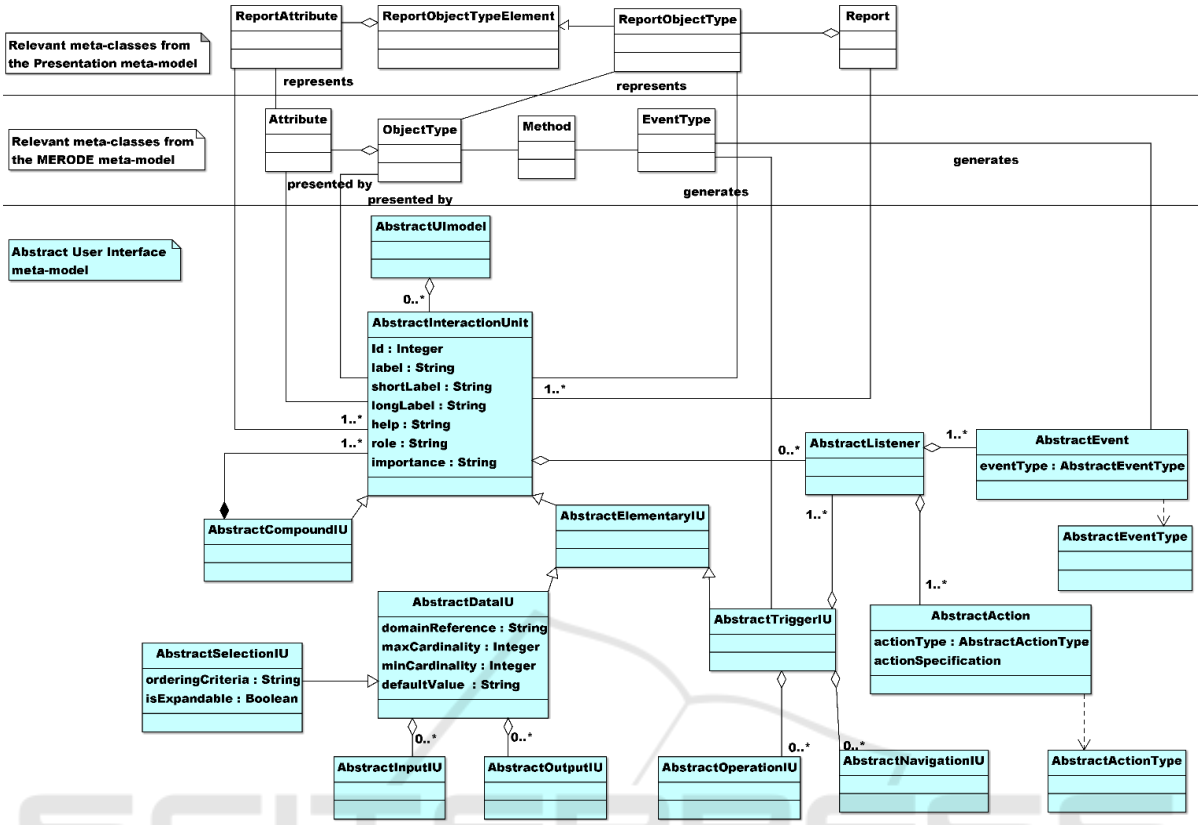

Figure 5: FENIkS AUI meta-model.

An AbstractElementaryIU can be an AbstractDataIU or AbstractTriggerIU. An AbstractDataIU consists of an elementary AUI that is responsible for data input and/or output that could be linked to a domain object via a domain object reference in order to ensure data binding with the associated domain model. The AbstractDataIU can be for input or output of data, through the meta-classes AbstractInputIU and AbstractOutputIU. A special case of the AbstractInputIU is the AbstractSelectionIU, which is a way to interact with the system by selecting an item from a list.

An AbstractTriggerIU allows navigating or operating with the UI. It is related to the AbstractListener, which describes the behaviour of the UI. An AbstractListener is composed of an AbstractEvent (which specifies the signal that triggers the action, e.g. onDataInput, onTriggerSelected) and an AbstractAction (which consists of updates or invocations on the domain model data, or of modifications of the abstract entities themselves, e.g. IUOpen, IUClose.

For the transformation to the AUI, FENIkS incorporates a set of mapping rules that allows determining the following:

- The abstract interaction units (the containers) starting from the conceptual domain model (using the object types) and the presentation model (using the report).
- The abstract data interaction units (the individual components) starting from the conceptual domain model (using the object types and attributes) and the presentation model (using the report object type and report attributes).
- The abstract trigger interaction units starting from the conceptual domain model (using the event types).
- The abstract listener for the abstract trigger interaction units.

The AUI is then transformed into the final UI code. This can already be done at a very early stage in the development process as long as the used models are correct.

3.3 Teaching Support

The didactic JMermaid tool extended with FENIkS provides support for teaching the development of interactive software system. The main extended features are as follows: 1) incorporation of UI design

principles, 2) runtime preview of the resulting UI according to the configured UI design, and 3) feedback for UI design and its integration with the development of the software application.

3.3.1 UI Design Principles

The teaching support of FENIkS is based on a set of UI design principles. UI design principles are high level concepts that allow guiding the software design (Mandel, 1997). They encompass the best practices in design, agreed upon by experts in the field.

Many design principles can be found in the literature. In order to select the design principles to be applied in FENIkS we analysed the ones proposed by important authors in the field; in particular those that propose empirical validated guidelines, such as (Norman, 1983; Nielsen, 1995; Stone et al., 2005; Johnson, 2007; Shneiderman, 2010). We only retained the ones that can be applied to the functional design of the UI. The selected design principles are: *Structure the UI, Allow users to use either the keyboard or mouse, Prevent errors, Good error messages* and *Provide visual cues.*

The UI design principles are shown to the learners as UI design options, related to the presentation model options described in section 3.2.1. The selected options are stored in the presentation model and further used for the generation of the UI.

Table 1: Feature model elements for Windows aspects.

Principle	Feature	Options
Structure the User Interface	Method presentation style	Pane
		Menu
Allow users to use either keyboard or mouse	Generate shortcuts for methods	True
		False
	Generate shortcuts for tabs	True
		False
	Generate shortcuts for general menu	True
		False

Table 2: Feature model elements for Input aspects.

Principle	Feature	Option
Prevent Errors	Validate empty data	True
		False
	Validate numbers	True
		False
	Validate Boolean	True
		False
	Generate components by the attribute type	True
		False
Good error messages	Generate errors according to the type of error	True
		False
Provide visual cues	Format data type Information	Show
		Hide
	Attribute data type Information	Show
		Hide

For better understanding, Table 1 and Table 2 show the relation between the presentation model elements (*Window* and *Input aspects)*, the UI design principles, the features in the Feature model and the options for each one of them related to the presentation model.

3.3.2 UI Generation Preview

To support a learner in understanding the presentation model more easily, FENIkS has included UI-GEAR: User Interface Generation prEview capable to Adapt in Real-time (Ruiz, Serral and Snoeck, 2017). UI-GEAR presents each view of the presentation model in a different tab. Figure 6 shows the presentation model dialog with its tabs, the *Windows aspects* and its preview being visible.

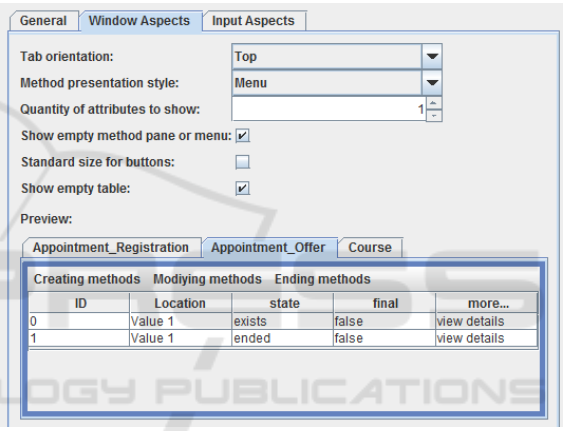

Figure 6: Presentation model and UI Preview.

At the bottom of the *Window* and *Input aspects*, UI-GEAR offers a preview showing how the UI will be generated in the prototype. The preview automatically adapts to any change in the selected options of the *Window* and *Input aspects,* allowing the learner to see how the UI will look like. The preview allows tracing changes from both conceptual domain and presentation models to their effects by testing several "what-if" scenarios. The presentation model has default options that can be used without explicitly be specified by the developer.

The UI-GEAR component for the presentation model has an internal representation of the feature model that is parsed and interpreted in real-time, enabling instantaneously update the preview. This gives the possibility to validate user requirements, reduces the time and effort required to implement the UI.

3.3.3 Automated Feedback

Feedback has been widely recognized and proven as an important aspect in teaching to ensure that students learn (Hattie and Timperley, 2007). Technology can support the provision of the frequent, constant and immediate feedback (Merrill, 2002) that is usually not possible to provide by teachers.

JMermaid has automated feedback features for developing the domain model of an application (Sedrakyan and Snoeck, 2013). The FENIkS extension incorporates feedback for the UI design. The feedback features allow explaining reasons of execution failures with graphical visualization that links the failure to the model where it is located. In a similar way as for domain model feedback, UI design feedback can assist the learners to validate the generated UI in a fast and easy way, while integrated with the rest of the application.

The introduction of functional design options, as previously explained, allows making the link with UI design principles. These design decisions are related to the preferences about how the components will be shown in the UI, and the preferences related to how the user will input information into the application, like how the components for attribute input will be generated or how the data will be validated.

The second kind of feedback is also in the UI Help. When developing the presentation model, the designer needs to take into account the UI design principles to select the correct options. This feedback shows to the learner which principles were well applied or not and why, according to the choices made. It also shows principles applied by default by the MDE engine while generating the prototype.

The teaching support in FENIkS for the UI design includes feedback features to explain 1) why the UI is generated in a specific way and how to change it, and 2) whether the design decisions are compliant with functional design principles or not and why.

The first kind of feedback is divided according to the structure of the presentation model. It is presented as a UI Help in the main window of the generated prototype. The designer selects *General*, *Window* or *Input aspects*, and which option he/she wants to see the explanation. Figure 7 shows an example of the first kind of feedback corresponding to the *Input aspects*.

The feedback is divided in three parts: what the stored values are in the presentation model, what the consequences are for the generated prototype and how it is possible to change it. From this place it is also possible to see all the values of the *Input aspects* and the presentation model.

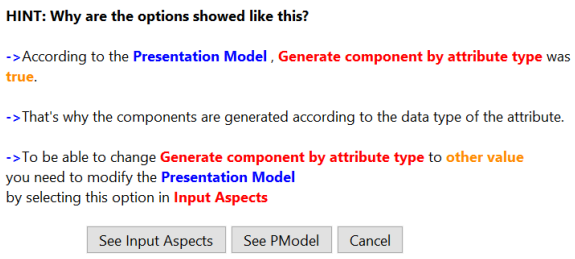

Figure 7: Design options explanatory feedback.

Checking the UI principles is also possible before generating the prototype. Figure 8 shows an example of the second kind of feedback.

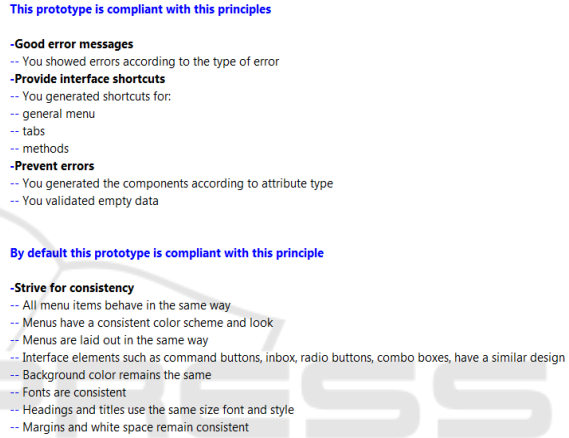

Figure 8: Checking the UI principles.

The automatic generation of the UI integrated with the application code enables the validation of the user requirements by simulation. This helps the learners to learn from and correct their mistakes. The generation of the final UI also allows comparing the impact of different design choices. The UI design feedback features explain to the learner the link between the design decisions, the applied design principles and the prototype integrated with the application development.

4 EVALUATION

To validate the developed didactic tool we evaluated: 1) its support as an integrated approach for the teaching of interactive software systems, and 2) its ease of use by learners.

4.1 Evaluation of the Integration

Our didactic tool assists learners in creating the domain model and the presentation model, as well as in

generating the AUI model and the system code. The advantages of the tool include:

1) It offers an integrated tool adapted to conceptual modelling goals and UI design goals through the extension FENIkS. This tool allows for the "co-design" of the application (with the conceptual domain model) and the UI (with the presentation model an further AUI model generation). The learner can easily switch between adapting the application or adapting the UI, while keeping the link between all the models.
2) The domain model and the presentation model allow the automatic generation of an AUI model, enabling further transformation to different contexts of use. The transformation process uses templates that can be changed in order to obtain the implementation of the interactive software system with different languages, platforms and ways of presentation.
3) The UI code is integrated with the application code. The generated prototype is fully functional and contains the link between UI and application logic.
4) Generation of the prototype can already be achieved from a minimal domain model consisting of only one object type. Default attributes and default options for the presentation model are present if not specified by the developer. There is no need to have a perfect or complete set of model before being able to test the UI and the application code.
5) FENIkS generates the UI taking into account the user´s preferences described in the presentation model. The generation of the UI according to the information of the presentation model allows iterative changes to the software solution, facilitating the comparison of each variant for a best match to user preferences.
6) Feedback features link the UI design options with the generated prototype, explaining the applicability of UI design principles and providing feedback about the domain model. The tool provides automatic feedback for both the application development and the UI design, by linking these to the conceptual domain and presentation models. Specifically, the feedback for the UI design can be checked before generation, which makes the tool easier to use. The possibility of generating the prototype without needing the complete models, checking partial versions of the prototype in a faster way, also contributes to ease of use.

4.2 Ease of Use

We evaluated MERODE tool extended with FENIkS from the perspective of perceived usability by performing an experiment with 12 novice developers. No participant has prior knowledge of the tool. We used the Computer System Usability Questionnaire (Lewis, 1993).

Each participant was asked to carry out a set of tasks in FENIkS. Using an already developed conceptual domain model as starting point, they played with the different design options to create a presentation model and to generate the prototype.

After completing the tasks, the users were asked to fill the CSUQ. During the sessions users were not allowed to ask questions to the evaluator.

The scores for all the items of the CSUQ ranked well above 5 on 7 (the highest possible value is 7), indicating a very positive evaluation. From all the items of the CSUQ, the mode of only three items was 5, while for all the other items the mode was 6 or 7. The highest mean values were obtained for items related to the ability to complete the work using the system, the clarity of the errors provided by the system that help to fix problems and that the system has all the functions and capabilities the developer expect.

The experiment demonstrated that 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. Developers found FENIkS very satisfactory in all areas for which the CSUQ accounts: usefulness, information quality, and interface quality. FENIkS is positively perceived overall and provides the functionalities the developers expected. For more details of the experimental evaluation the reader is referred to (Ruiz, Serral and Snoeck, 2017).

We also tested the suitability of FENIkS for novice UI designers by means of the questionnaires to measure the perceived usefulness (user acceptance). The experiment was made with 54 participants that took a UI design course as part of their 4th year of Informatics Engineering program at the University of Holguín. After using FENIkS, the participants filled the questionnaires.

The scores per item rank well above 5 on 6, indicating a positive evaluation. The mode of only four items was 5, while for all the other items the mode was 6 or 7. The learners agree that using FENIkS was a positive experience and that it improves their understanding of UI principles which were the items with a highest mean values.

The results of the questionnaire gave support that the proposed simulation method is suitable for novice UI designers.

5 LIMITATIONS

A first limitation of our approach is that only functional aspects of the UI are modelled: FENIkS is not focused on aesthetic appeal.

For the moment, the tool 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 interactive software system to other languages and platforms can be easily extended in future versions of the tool, using the current proposed AUI model. This will allow also comparing and giving feedback according to the results of the design in different final UIs.

Since the original MERODE tool had no support for the UI design, it is clear that the FENIkS extension improves UI design when designing interactive software systems. Nevertheless, the presentation metamodel could be further extended to improve flexibility. Other models (e.g., user model) could be incorporated to provide better support for users characteristics.

6 CONCLUSION

This paper has presented a MDE didactic tool for improving the teaching of interactive software systems. While designing the UI the learner receives feedback about how some UI design principles are applied through the options the learner selects. At the same time, the learner completes a conceptual domain model used for the generation of both the UI and the application code. For the conceptual modelling the learner also receives the feedback provided by JMermaid. FENIkS' automatic generation of the UI integrated with the application code allows validating user requirements against the prototype behaviour and the resulting UI. Thus, necessary changes in the models can be made in less time while maintaining the link between the UI and the application.

The developed tool improves the process of UI designing and application development by letting the learner tests the models incrementally. The feedback allows understanding how the UI design principles are applied and immediately shows their effects on the final UI.

Last, but not least, we discussed how FENIkS could be extended with more flexibility in the UI design and to support other context of use.

REFERENCES

Akiki, P. A., Bandara, A. K. and Yu, Y. (2015) 'Adaptive model-driven user interface development systems', *ACM Computing Surveys*. ACM, 47(1).

Barrett, M. L. (1993) 'A hypertext module for teaching user interface design', in *ACM SIGCSE Bulletin*. ACM, pp. 107–111.

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

Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J. and Creemers, B. (2003) 'Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems', in *Mobile HCI*. Springer, pp. 256–270.

da Cruz, A. M. R. and Faria, J. P. (2009) 'Automatic Generation of user Interface Models and Prototypes from Domain and Use Case Models', in *ICSOFT (1)*, pp. 169–176.

Dehinbo, J. (2011) 'Establishing and applying criteria for evaluating the ease of use of dynamic platforms for teaching web application development', *Information Systems Education Journal*, 9(5), p. 86.

Delgado, A., Estepa, A., Troyano, J. A. and Estepa, R. (2016) 'Reusing UI elements with Model-Based User Interface Development', *International Journal of Human-Computer Studies*. Elsevier, pp. 48–62.

Engel, J., Märtin, C. and Forbrig, P. (2017) 'Practical Aspects of Pattern-Supported Model-Driven User Interface Generation', in *International Conference on Human-Computer Interaction*. Springer, pp. 397–414.

Feuerstack, S., Blumendorf, M., Schwartze, V. and Albayrak, S. (2008) 'Model-based layout generation', in *AVI*. ACM, pp. 217–224.

Hattie, J. and Timperley, H. (2007) 'The power of feedback', *Review of educational research*. Sage Publications, 77(1), pp. 81–112.

Hentati, M., Ben Ammar, L., Trabelsi, A. and Mahfoudhi, A. (2016) 'A fuzzy-logic system for the user interface usability measurement', in IEEE/ACIS (ed.) *17th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/ Distributed Computing, SNPD*, pp. 133–138.

Johnson, J. (2007) *GUI bloopers 2.0: common user interface design don'ts and dos*. Morgan Kaufmann.

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

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L. and Florins, M. (2004) 'USIXML: A User Interface Description Language Supporting Multiple Levels of Independence', in *ICWE Workshops*, pp. 325–338.

London, K. C. and University of York (2003) *An Evaluation of Compuware OptimalJ Professional Edition as an MDA tool*. Available at: http://www.lcc.uma.es.

Mandel, T. (1997) *The elements of user interface design*. Wiley New York.

Marin, I., Ortin, F., Pedrosa, G. and Rodriguez, J. (2015) 'Generating native user interfaces for multiple devices by means of model transformation', *Frontiers of Information Technology & Electronic Engineering*. Springer, 16(12), pp. 995–1017.

Meixner, G., Paternò, F. and Vanderdonckt, J. (2011) 'Past, Present, and Future of Model-Based User Interface Development', *i-com*, 10(3), pp. 2–11.

Merrill, M. D. (2002) 'First principles of instruction', *Educational Technology Research & Development*, 50(3), pp. 43–59.

Molina, J. C. and Pastor, O. (2004) 'MDA, OO-Method y la tecnología OlivaNova Model Excecution', *I Taller sobre desarrollos dirigidos por modelos, MDA y aplicaciones. Málaga*.

Nguyen, K. D. and Rahman, M. A. (2016) 'Identifying Interface Design Patterns by Studying Intrinsic Designs', in *The Third International Conference on Computer Science, Computer Engineering, and Education Technologies (CSCEET2016)*. Poland, pp. 13–24.

Nielsen, J. (1995) *10 usability heuristics for user interface design*.

Norman, D. A. (1983) 'Design principles for human-computer interfaces', in *SIGCHI*. ACM, pp. 1–10.

Pastor, O. and Molina, J. C. (2007) *Model-driven architecture in practice*. Springer.

Recker, J. (2012) *Scientific research in information systems: a beginner's guide*. Springer Science & Business Media.

Ruiz, J., Sedrakyan, G. and Snoeck, M. (2015) 'Generating User Interface from Conceptual, Presentation and User models with JMermaid in a learning approach', in *Proceedings of the XVI Internation Conference on Human Computer Interaction*. Vilanova i la Geltrú, Spain: ACM. doi: http://dx.doi.org/10.1145/2829875.2829893.

Ruiz, J., Serral, E. and Snoeck, M. (2017) 'UI-GEAR: User Interface Generation prEview capable to Adapt in Real-time', in *Modelsward´2017*. Porto, pp. 277–284.

Sboui, T. and Ayed, M. Ben (2016) 'Generative Software Development Techniques of User Interface: Survey and Open Issues', *International Journal of Computer Science and Information Security*. LJS Publishing, 14(7), p. 824.

Sedrakyan, G. and Snoeck, M. (2013) 'Feedback-enabled MDA-prototyping effects on modeling knowledge', in *Enterprise, Business-Process and Information Systems Modeling*. Springer, pp. 411–425.

Sedrakyan, G., Snoeck, M. and Poelmans, S. (2014) 'Assessing the effectiveness of feedback enabled simulation in teaching conceptual modeling', *Computers & Education*, 78, pp. 367–382.

Seffah, A., Gulliksen, J. and Desmarais, M. C. (2005) *Human-Centered Software Engineering-Integrating Usability in the Software Development Lifecycle*. Springer.

Shneiderman, B. (2010) *Designing the user interface: strategies for effective human-computer interaction*. 5th edn. Addison-Wesley.

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

Stone, D., Jarrett, C., Woodroffe, M. and Minocha, S. (2005) *User interface design and evaluation*. Morgan Kaufmann.

Sutcliffe, A. G., Kurniawan, S. and Shin, J.-E. (2006) 'A method and advisor tool for multimedia user interface design', *International Journal of Human-Computer Studies*. Elsevier, 64(4), pp. 375–392.