Automatic Feedback Generation for Supporting User Interface Design

Jenny Ruiz[1][a] and Monique Snoeck[2][b]
[1]Faculty of Informatics and Mathematics, University of Holguin, XX Anniversary Ave., Holguin, Cuba
[2]Faculty of Economics and Business, KU Leuven, Naamsestraat 69, 3000 Leuven, Belgium

Keywords: Abstract User Interface Model, Presentation Model, Feature Model, Model-Driven Engineering, User Interface Design, User Interface Generation, Automated Feedback.

Abstract: Although the interest for User Interfaces (UI) has increased their study, their design is a difficult process to learn. Novel UI designers, therefore, need guidance through the learning process of UI design to obtain better results. Feedback is a key factor to improve knowledge and skills acquisition. However, providing individual feedback is a complex and time-consuming task and requires a fair amount of expertise. This paper describes a solution to this problem: Feedback ENriched user Interface Simulation (FENIkS). FENIkS is a model-driven engineering UI design simulation tool able to automatically provide instant feedback to the students about how they apply UI design principles. While designing the UI, the novice designer receives feedback on how design principles are applied through the options he/she selects. Then, when generating a working prototype from the models, feedback explaining the application of principles is incorporated in the prototype. An experimental evaluation was conducted, demonstrating FENIkS improves students' understanding of UI design principles. The perceived usability was also positively evaluated. This paper explains FENIkS' design: the meta-model, how design options, design principles and types of feedback are used to generate automated feedback on the observation of design principles in the tool and the generated prototype.

1 INTRODUCTION

The high use of software applications in everyday life has increased the importance of User Interfaces (UI), as the mean that allows the interaction between the end user and the application (Akiki et al., 2015). UI design is a difficult process (Beuvens & Vanderdonckt, 2012; Sboui & Ayed, 2016) due to its inherent complexity by its multidisciplinary nature, the need for designing UI for several contexts of use and the need for understanding a wide range of approaches and various levels of abstraction.

Novel UI designers, therefore, need guidance through the learning process of UI design. To improve their design skills, they require a large amount of practice and clues about their efforts. Feedback has been seen as a key factor to improve knowledge and skills acquisition (Shute, 2008). Providing individual feedback is a complex and time-consuming task and requires a fair amount of expertise. A solution could be providing students with a design for comparison. However, the fact that many design can be correct makes self-assessment difficult.

For UI design it is particularly challenging to address the complexity of the problem and giving feedback: when a student addresses a difficult or large exercise, giving personal feedback is even more time consuming. The many valid solutions for a single problem call for individual and immediate feedback. Technology can be used to provide more frequent and immediate feedback (Merrill, 2002; Van Eck, 2006). Few approaches has been developed to support the learning of UI design (Benitti & Sommariva, 2015; Lisowska Masson et al., 2017; Sutcliffe et al., 2006), although without providing automated feedback.

In previous work we proposed an approach to support learning achievements for the development of interactive software systems (Ruiz et al., 2019) with Feedback ENriched user Interface Simulation (FENIkS): FENIkS is a UI design simulation tool able

[a] https://orcid.org/0000-0002-1371-6353
[b] https://orcid.org/0000-0002-3824-3214

Ruiz, J. and Snoeck, M.
Automatic Feedback Generation for Supporting User Interface Design.
DOI: 10.5220/0010513400230033
In *Proceedings of the 16th International Conference on Software Technologies (ICSOFT 2021)*, pages 23-33
ISBN: 978-989-758-523-4

to automatically provide instant feedback to the students about how they apply UI design principles, while generating a working software prototype. This paper extends the previous work by describing the solution to the problem of providing feedback: it describes the conceptual design of the feedback and the technology to provide it. The remainder of this paper is structured as follows: Section 2 examines the related work on automated feedback and UI design teaching support. Section 3 describes FENIkS. Section 4 presents the evaluation and Section 5 concludes the paper.

2 RELATED WORK

We analyzed the approaches related to our work from several perspectives: automated feedback generation for learning support, teaching support for UI design, providing feedback to non-usability experts at design time or pattern-driven approaches.

Automated Feedback to Students. In (Butchart et al., 2009) the authors describe a software that provides automatic feedback on the students' progress, while they construct their maps of an argument. The approach proposed in (Mosteller et al., 2018) provides simulation feedback for domain specific modeling languages on the basis of meta-models and translational semantics using Petri nets. The framework presented in (Parvez & Blank, 2008) provides a feedback infrastructure learning style to help novices learn how to design a class in UML.

Supporting UI Design Teaching. A hypertext module called UID tutorial is proposed by (Barrett, 1993). This tutorial shows good and bad examples, i.e. where design principles are applied or violated. A similar approach is followed by (Sutcliffe et al., 2006) which uses examples to give recommendations about which media is appropriate. The teaching of usability engineering life cycle, prototyping and heuristic evaluation is supported by a game proposed by (Benitti & Sommariva, 2015). This game presents examples of design of web interfaces and the student needs to select which heuristics are applied. Usability issues are addressed in a learning management system: ILIAS (Lisowska Masson et al., 2017). This work presents a taxonomy of UI components within ILIAS and recommendations for how to use them.

Providing Feedback to Non-usability Experts at Design Time. In (Ormeño et al., 2014) the authors propose an approach to elicit usability requirements at early stages of the software development process providing feedback to non-experts. The approach shows interface design guidelines and usability guidelines to the analyst through questions to be asked to the end-user. Using the end-user´s answers, a set of usability requirements is obtained.

Pattern-driven Approaches. Patterns collect best practices and make expert knowledge explicit to novices. The authors of (Molina et al., 2002) present an approach that shows different examples of abstract UI patterns. This can be seen as a form of feedback that can help to make decisions with patterns by documenting problems, their corresponding best solutions and the impact in the automatic code generation process. Patterns are also used by OO-Method (Pastor & Molina, 2007) in a presentation model to capture user´s preferences, similarly to the approach proposed in this paper. OO-Method provides complete, integrated support for the UI design and application development. It has been extended with the transformation template approach proposed in (Aquino et al., 2010), that makes the model-driven engineering UI process more flexible gathering generation options in reusable templates.

The approach described in this paper differs from prior works in several ways. The approaches that support the generation of automated feedback are not tailored to supporting UI design. The approaches that support UI design by providing example-based help (i.e. worked out examples) do not provide feedback related to a real design, something that is possible in our proposed environment. The most significant difference is the present approach´s emphasis on testing the compliancy of UI design principles in an actual UI that is designed by the student by the specification of conceptual and presentation models.

3 FENIkS

FENIkS is an extended version of JMermaid, a tool for teaching conceptual modeling. This tool is based on the concepts of MERODE: a model driven engineering (MDE) method for enterprise information system engineering (Snoeck, 2014). MERODE allows the specification of an enterprise system from a conceptual domain model. The model is platform independent and sufficiently complete which allows the automatic generation of the system's code from it. The generated prototype is enriched with didactic feedback linking the application's behavior back to the choices made in the model, thereby supporting the learning of conceptual modeling (Sedrakyan, Snoeck, & De Weerdt, 2014; Sedrakyan & Snoeck, 2013a, 2013b). The effectiveness of this tool has been proven through several evaluations (Sedrakyan, Snoeck, & Poelmans, 2014; Sedrakyan & Snoeck, 2013a).

In order to support novel UI designers' learning of UI design principles, JMermaid was extended with FENIkS. FENIkS focuses on the learning of the functional aspects of Graphical User Interfaces for enterprise information systems. To that end, FENIkS incorporates two extra models in this extension: the Abstract User Interface (AUI) model to describe the UI in a technology-agnostic way and the presentation model to capture the characteristics of the UI layout and components and the user preferences to support the learning of UI design principles (Ruiz et al., 2017, 2018). These models and how they relate to existing models are described in subsection 3.1. Subsection 3.2 presents how the feedback provided by FENIkS was conceived. Subsection 3.3 presents details of the implementation and estimates the volume of work.

3.1 Models

FENIkS is as a model-based approach to UI design. Model-based approaches generate UIs (semi) automatically using models of different abstraction levels. FENIkS has different abstraction levels, meta-models, and a supporting tool for the transformations. This section presents the models of FENIkS.

3.1.1 Conceptual Domain Model

MERODE has a conceptual domain model that defines the classes of objects in an enterprise, while in UI design a domain model is used to show the description of the classes of objects manipulated by a user while interacting with a system. Those two definitions of a domain model were merged in FENIkS to improve the generated software.

MERODE´s conceptual model is composed of a class diagram, an object event table, and finite state machines that capture enterprise object behavior. The class diagram describes the domain classes including structure (attributes) and behavior (methods), and also associations between the classes. The object event table indicates which business events create, update or delete objects of a certain type. When an event affects objects of a certain type, this gives –accordingly- rise to create, modify or end methods in the corresponding class. This information is presented by means of a table associating object types and event types. The finite state machine is a diagram that specifies the life cycle of objects of a given class by means of states and transitions. There is a correspondence between the events triggering the transitions in the finite state machine and those that are represented in the object event table (Snoeck, 2014).

The supporting tool allows modeling the different views of the system. It manages the consistency between the three views: all the specifications that can be derived from one view to other are automatically generated by the tool. An example is the creation of an object in the class diagram that implies the automatic generation of creating and ending services and a default finite state chart.

3.1.2 Presentation Model

A presentation model is used to specify the UI by describing "the constructs that can appear on an end user´s display, their layout characteristics, and the visual dependencies among them" (Schlungbaum, 1996). The static part reflects the design of the UI as a composition of standard widgets like buttons, menu, etc. The dynamic part displays application dependent data that typically changes at run-time.

In many approaches, the presentation model is mainly used as abstract or concrete UI model. Others, like OO-Method (Pastor & Molina, 2007), use the presentation model to capture user preferences by means of interface patterns. The presentation model allows personalizing the presentation of the prototype based on user preferences. In line with the last definition, JMermaid was extended with a presentation model, resulting in FENIkS. The presentation model captures code generation options defining how the user will interact with the prototype and how the information will be shown.

JMermaid could only generate a default UI constituted of a window showing a list of instances of a single domain class, a window to view the details of one object and a default input window to trigger the execution of a business event. With FENIkS it is possible to define extra output services (or reports) to show specific information the user wants to see, for example, combining data from many domain objects.

The presentation meta-model of FENIkS defines windows and input aspects. Besides those aspects, additional output services are captured through the meta-object type Report and the associated meta-object types. A Report is composed of a selection of object types that needs to be shown and a selection of their attributes and associations (Ruiz et al., 2019).

The presentation meta-model is related to relevant parts of the MERODE meta-model. This is necessary for the definition of the dynamic aspects of the reports: rather than showing all objects, attributes and association, for each report it can be defined which objects, attributes and associations need to be presented to the user in the report. The presentation model needs to be connected to the class diagram to be able to retrieve the required information.

The meta-classes 'Window aspect' and 'Input aspect' capture the preferences related to how elements of the UI should be configured. 'Window aspect' captures the information of all reports, including default and additional reports. 'Input aspect' captures the preferences for input services. The preferences related to the static layout of the top level containers of the generated prototype and how the information is displayed are captured by the 'Window aspects'. Examples are how the pagination will be, how the methods will be shown, if there will be shortcuts for interacting with the system, etc. The preferences related to the way users will input the information into the generated prototype are captured by the 'Input aspects'. Examples are what kind of widgets are needed for inputting the information, how the validation of the inputs will be performed (or not) and what kind of error messages will be shown.

3.1.3 Abstract User Interface Model

Given the fact that software applications can be accessed by the users from a huge variety of contexts of use, AUI models are very important (Limbourg et al., 2004). The AUI model defines the UI independent from modality (visual, auditory, tactile, etc.), user interaction (keyboard, speech, etc.), the platform used for display, etc. (Trewin et al., 2004). With FENIkS, the designer does not define the AUI model: it is generated from the domain and presentation models.

The AUI represents the UI without taking into account any modality of interaction or platform. This is important to designers to help in understanding the main principles behind UI generation. An AUI for a default UI can be obtained by means of a model to model transformation from the conceptual model of MERODE. Figure 1 shows the process for obtaining the final UI for just one context of use.

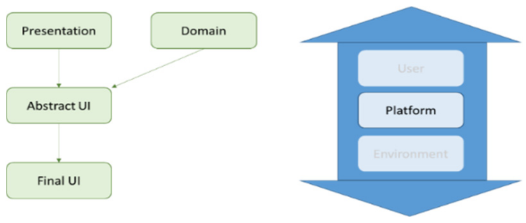


Figure 1: Models used in FENIkS for one context of use.

Because this research considers the platform as the only dimension of the context of use (to keep the scope of the research manageable), the other context aspects have been grayed out in Figure 1. The use of an AUI model makes possible future translation of the AUI to other final UIs for other contexts of use.

FENIkS' AUI meta-model is based on the AUI meta-model of the User Interface eXtensible Markup Language (UsiXML), a User-Interface Modelling language proposed by (Limbourg et al., 2004). To generate the AUI model, FENIkS uses concepts of the domain model (for the default output and input services) and of the presentation model (for the additional output services). In a previous paper (Ruiz et al., 2015) we showed the linking of the concepts of the AUI meta-model to the relevant concepts of the MERODE meta-model and the W3C AUI standard.

The presentation model captures the preferences for the UI generation through the meta-classes Windows and Input aspects. Capturing the preferences for showing or capturing information in a single place allows FENIkS applying the chosen options in a consistent way through the UI. The student mandatorily has to define the Window and Input aspects. For each of these aspects the student can set a number of options. Some of the options are at the abstract level of a UI and other that are at the concrete level. The abstract level features are used to generate the AUI model. The concrete level features are used to generate the final UI directly.

3.2 FENIkS Feedback

As explained before, the UI designer "designs" the UI by setting a number of options in the presentation model. The student receives feedback while setting the options in the presentation model and in the generated prototype. This section describes the design and implementation of this feedback.

3.2.1 UI Design Principles for Feedback

FENIkS supports the UI design process based on a set of UI design principles. Design principles encompass the best practices for designing usable UIs according to the experts on the field. Several authors have proposed design principles such as (Nielsen, 1994; Shneiderman & Plaisant, 2005).

From the more generic design principles, concrete guidelines have been proposed. To be able to incorporate design principles in an automatic way by means of MDE transformations it is necessary to select principles that can be translated into testable rules. The selection was executed by means of a systematic literature review described in (Ruiz et al., 2020b). We extracted 41 authors of UI design principles from a set of 475 papers. Focusing on the three most cited works of the 16 authors cited at least twice, we extracted 257 principles (including variations of the same principle). We unified the variants (principles similar by name, by concept, subsumed by more general principles), and, considering their scientific influence, we derived a selection of 22 principles that can be implemented in an MDE tool.

We note that some principles can be incorporated in an MDE tool with a manageable amount of effort while for other principles this would require a lot of effort, for demanding the implementation of difficult techniques. For example, incorporating the principle *Speak the User's Language* could require including natural language processing techniques. We studied several guidelines per principle and analyzed how many could be implemented. The more guidelines can be implemented, the easier we consider its implementation. Table 1 shows the list of principles and their level of implementation difficulty (Easy: E, Medium: M, Hard: H).

Table 1: Design principles and implementation difficulty.

Design principle	**E**	**M**	**H**
Total:	8	6	8
Prevent errors	x		
Provide good error messages	x		
Allow users to use the keyboard or mouse	x		
Provide visual cues	x		
Offer informative feedback	x		
Strive for consistency	x		
Visibility	x		
Structure the User's Interface	x		
Actions should be reversible		x	
Accommodate users with different skill levels		x	
Help users recognize and recover from errors		x	
Allow users to change focus		x	
Help and documentation		x	
Allow users to abort lengthy operations		x	
Minimize user´s memory load			x
Simple and natural dialog			x
Give the User Control			x
Speak the User's Language			x
Design dialogs to yield closure			x
Flexibility and efficiency of use			x
Allow users to customize the interface			x
Allow users to estimate how much time operations will take			x

For the proof of concept we implemented feedback for the easiest to implement principles:

- Prevent errors.
- Provide good error messages.
- Allow users to use the keyboard or mouse.
- Provide visual cues.
- Offer informative feedback.
- Strive for consistency.
- Visibility.
- Structure the UI.

The use of MDE makes that certain principles are automatically respected. Therefore, a number of principles were chosen to be automatically supported by default. Other principles were chosen to be taken care of by the UI designer by choosing the right UI design options. Some of the principles are partially taken care of by the UI designer through the options.

For the principles that need to be actively observed by the designer, s/he needs to select the options for the principles, according to the options part of the Windows and Input aspects. Each principle can have one or more options and each of them has positive or negative values. If the designer selects a positive value for an option, the UI will be compliant with the guidelines of the associated principle. If the designer selects a negative value for an option the principle will be violated. For a better understanding of how the design principles are observed through the options, Table 2 shows the principles, features and values associated to the presentation model.

Most of the features of the Window and Input aspects have been included in FENIkS for educational purposes: they are used to show the learner how to apply UI design principles and generate the UI accordingly. An example of such feature in the 'Window aspects' is 'Generate shortcuts for tabs'. An example of such feature in the 'Input aspects' is 'Validate Empty data'. Other features have been included to give flexibility to the prototype generation process. Examples of such features are mainly in the 'Window aspects': Table pagination, Empty table, etc.

3.2.2 Designing the Feedback

As explained in (Serral Asensio et al., 2019), several factors need to be taken into account when automating feedback. FENIkS' feedback features were elaborated based on this framework, including factors associated to the design of the feedback and for automatically creating and delivering the feedback.

At the general level, the most important factors for building the feedback are: 1) Content Design, to represent the relevant factors for automatically designing the feedback content; 2) Delivery, to describe the relevant factors for automatically delivering feedback; 3) Context, the contextual aspects to consider for automatic feedback; 4) Usage: to describe the possible usages of feedback by learners; 5) Impact: to express the factors that can be measured

Table 2: Design principles and associated features.

Design principle	Feature	Positive value	Negative value
Prevent errors	Validate boolean data	True (Compliant)	False (Not compliant)
	Validate integer data	True (Compliant)	False (Not compliant)
	Validate empty data	True (Compliant)	False (Not compliant)
	Generate components by the attribute data type	True (Components are generated according to attribute data type)	False (All the components are generated as input text boxes)
Provide good error messages	Errors according to the type of error	True (Messages generated according to the type of error)	False (Generic message generated without specifying the type of error)
Allow users to use the keyboard or mouse	Generate shortcuts for methods	True (Shortcuts for the methods are generated)	False (It is not possible to access to the methods through the keyboard)
	Generate shortcuts for tabs	True (Shortcuts for the tabs are generated)	False (It is not possible to Access to the tabs through the keyboard)
	Generate shortcuts for general menu	True (Shortcuts for the general menu are generated)	False (No access to the general menu through the keyboard)
Provide visual cues	Format data type information	True (The format of the data type is shown next to the attribute)	False (Only the name of the attribute is shown)
	Attribute data type information	True (Data type information of the attribute shown next to its name)	False (Only the name of the attribute is shown)

or observed to determine the impact of automated feedback and 6) Technique: the techniques, algorithms, etc. used or implemented for automating feedback (Serral Asensio et al., 2019). Each of these factors can be further broken down in several aspects.

Table 3 shows the most important factors of the feedback provided by FENIkS. The goal is to improve the learning of UI design principles: its feedback is oriented to help the student´s to reach this goal.

The purpose of the feedback in FENIkS is corrective and explanatory: it provides knowledge about the correct response related to the application of the UI design principles in the presentation model. The feedback is formative: it informs the compliancy of the UI design principles with guidelines to improve the answers, or reinforce the correct answers. It is possible to revise one´s design while performing the learning task, allowing to change the answers and receive new feedback. This feedback is provided at task-level and addresses how well tasks (the application of design principles) are understood, performed or applied. The feedback focuses on faults in the interpretation of the UI design principles.

For a principle with feedback generation options FENIkS explains if all the guideline constraints have been satisfied or not and why. The principle is considered well applied in the generated prototype if all the options associated to a design principle have the positive value. If at least one guideline constraint is not satisfied, the principle is considered violated.

For the factor delivery, the chosen format is textual, with messages embedded in FENIkS. It is also visual when showing the generated UI and while interacting with the generated prototype. With regard to the timing the student can check it anytime. The level of learner control is taken by the student who decides when and where to see the feedback.

Table 3: Applying the framework to FENIkS.

Factor	Aspect	FENIkS feedback
Content	Purpose	Corrective, explanatory, formative
	Level	Task-level
	Nature	Possitive and negative
	Domain	User Interface design principles
Delivery	Format	Textual and visual
	Timing	Anytime on demand to the student
	Learner control	Taken by the student
Context	Recipient	Individual learner
	Device	Desktop
	Learning task	Simple and complex tasks
	Educational setting	University
Usage		Check if the learning is on track
Impact		Measured by student´s scores
Technique		Template-based MDE technique for the derivation of feedback from the defined constraints about the compliancy of UI design principles showing the specific details of the error or the correct solution.

For the context factor, the recipient of the feedback is an individual learner. The used device is a desktop computer. The feedback is for simple and complex learning tasks. The educational setting is at university level where the learning takes place. For the usage factor, there are different possible usages for the feedback: it can be used for motivation purposes; for verifying learning progress, etc. In FENIkS, the feedback is oriented to help the learner to check if the learning is on track and therefore, improve it. For impact factor, we chose to measure it by means of student´s scores when evaluating their learning on the domain of UI design principles.

With a template-based model driven development technique (see Subsection 3.3) FENIkS builds the feedback according to two types of feedback to explain: 1) whether the design choices are compliant with UI design principles or not and why; and 2) why the UI is generated in a specific way tracing the application´s appearance back to its origin in the presentation model and how to change it (Ruiz et al., 2019).

Figure 2 shows the FENIkS feedback meta-model. The FeedbackModel meta-class is composed of two meta-classes according to the two types of feedback FENIkS provides: for the compliancy of the UI design principles (didactic purposes features) and for the options to give flexibility (flexibility features).

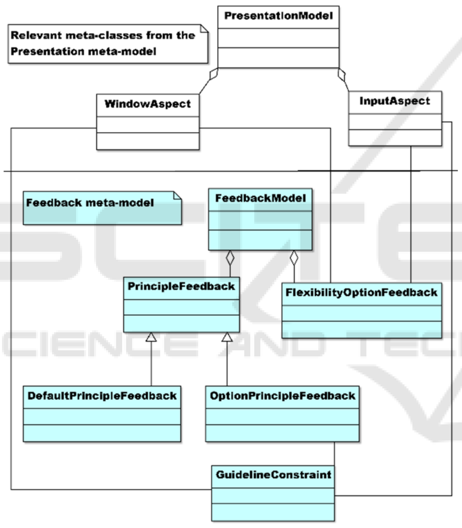


Figure 2: Feedback meta-model.

The PrincipleFeedback meta-class represents the provided feedback about the principles that are automatically applied. The OptionPrincipleFeedback meta-class expresses the feedback that is provided according to how the UI design principles have been correctly applied or not. To deliver this feedback it is necessary to check the constraints associated to the options of each principle (captured by Window or Input aspects). The upper part of the figure includes the relevant meta-classes of the presentation meta-model, and shows how the GuidelineConstraints link InputApects to OptionPrincipleFeedback. The features related to the Window and Input aspects which give the designer flexibility for the UI design are used to build the FlexibilityOptionFeedback.

3.2.3 Example of FENIkS´ Use

This section illustrates FENIkS´ use by means of an example. Figure 3 shows the class diagram of a student´s grade system. Teachers teach classes of certain subject. Students are enrolled in those subjects and obtain grades for each of them. Several grades can be obtained as the student is allowed at least two attempts, the best of which counts as final grade.

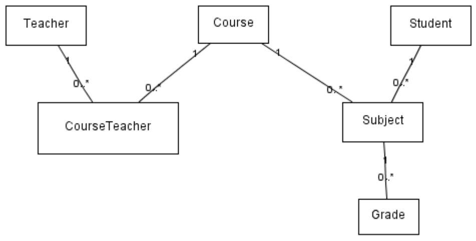
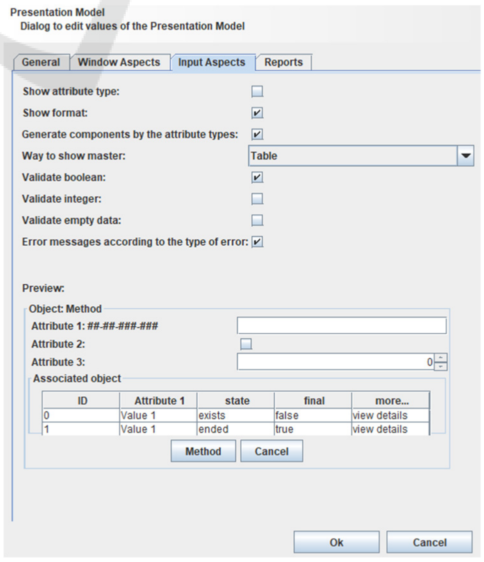


Figure 3: Student’s grade system class diagram.

The UI designer elaborates the presentation model. Figure 4 shows the tab corresponding to the 'Input aspect' of the presentation model.

After the generation of the prototype, the UI designer can check the UI feedback incorporated in the prototype itself. The prototype includes a UI Help option in the main menu where he/she can check the different kinds of feedback FENIkS provides. In the first part of the UI Help the designer can see all the preferences captured by the Window and Input aspects and the associated feedback given for the flexibility options. This kind of feedback allows selecting the options and seeing why the UI is generated in certain way.


Presentation Model
Dialog to edit values of the Presentation Model

Show attribute type: ☐
Show format: ✓
Generate components by the attribute types: ✓
Way to show master: Table
Validate boolean: ✓
Validate integer: ☐
Validate empty data: ☐
Error messages according to the type of error: ✓

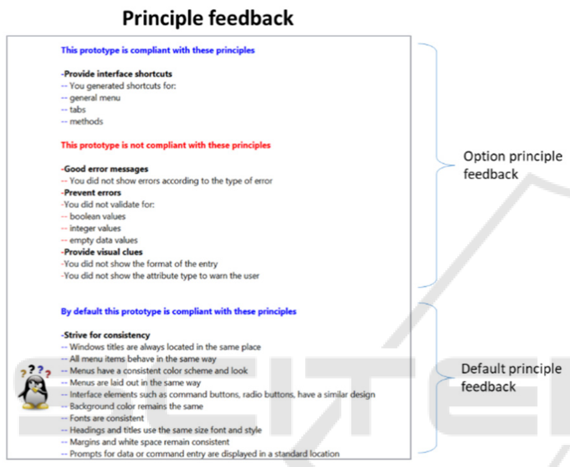
Preview:
Object: Method
Attribute 1: ##-##-###-###
Attribute 2: ☐
Attribute 3: 0
Associated object

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

Figure 4: Input aspects of the presentation model.

The designer sees a preview of how the prototype will be generated according to the options. Then, before the generation, the designer can check the feedback on whether the UI design principles are satisfied by the options.

The UI Help includes a second type of feedback about compliance of chosen options with the UI principles. An example is shown in Figure 5. This feedback is split in two parts to show: 1) the principles the generated prototype is compliant with and the principles the generated prototype violates and 2) the principles the generated prototype implements by default. The rationale for (non-) compliance is always given.


Principle feedback

This prototype is compliant with these principles

-Provide interface shortcuts
-- You generated shortcuts for:
-- general menu
-- tabs
-- methods

This prototype is not compliant with these principles

-Good error messages
-- You did not show errors according to the type of error
-Prevent errors
-You did not validate for:
-- boolean values
-- integer values
-- empty data values
-Provide visual clues
-You did not show the format of the entry
-You did not show the attribute type to warn the user

By default this prototype is compliant with these principles

-Strive for consistency
-- Windows titles are always located in the same place
-- All menu items behave in the same way
-- Menus have a consistent color scheme and look
-- Menus are laid out in the same way
-- Interface elements such as command buttons, radio buttons, have a similar design
-- Background color remains the same
-- Fonts are consistent
-- Headings and titles use the same size font and style
-- Margins and white space remain consistent
-- Prompts for data or command entry are displayed in a standard location

Figure 5: Different types of feedback generated by FENIkS.

3.3 Implementation

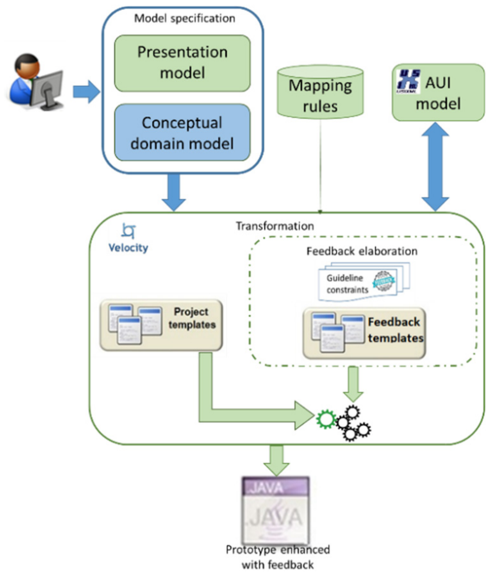


Figure 6: FENIkS transformation process.

As shown in Figure 6, the transformation engine constitutes the heart of FENIkS. Once the conceptual domain and the presentation models are elaborated first transformation to the AUI model and then transformation to the UI and application code are executed. A default presentation model can be used.

Mapping rules define how to transform the domain model into the AUI-model (see (Ruiz et al., 2019) for the set of mapping rules). The second step combines the coding templates, with the information contained in the conceptual, presentation and AUI models, and generates the Java code of the prototype. The code generator has been built using Java and the Velocity Templates Engine. The use of a template-based transformation allows going from model to code.

The coding templates come in two kinds: project templates for generating the application code and feedback templates to generate the feedback. The feedback messages are generated and incorporated in the prototype using two feedback templates. For the feedback associated to the compliancy of the UI design principles, the generated feedback shows the specific details of the error (why the principle is considered violated), or the correct solution (why the principle is considered well applied). The other type of feedback shows the chosen preferences for the presentation model, what the consequences are for the generated prototype and how the preferences can be changed, either for the Window or Input aspects.

The FENIkS extension has been implemented over a period of two years of non-fulltime work. Table 4 sheds some light on the volume of code. The implementation of FENIkS required incorporating new classes to JMermaid (6032 new lines of code), but also to modify existing classes. The modified lines of code are not shown in Table 4.

Table 4: Measures of the implementation.

Project	Lines of code	Comment lines	Number of classes
JMermaid	44085	509	572
With FENIkS	50117	568	607
Code generator	10618	1458	35
Templates	27781	353	26

As principles are generally expressed at a high level of abstraction, each original principle needed to be translated into a more concrete form as a guideline. However, even guidelines can be expressed at a level that is still not immediately fit for transformation rules. The selected principles and their associated guidelines were analyzed in order to be matched to options that can be used for code generation. In some cases, the principles were matched to guidelines that were applied by default by the generation engine.

Table 5 reports on the complexity of the principles. The complexity is calculated by counting the number of guidelines applied by means of design options and those applied by default by the tool. For each guideline with options a weight of 1 was given. For each guideline applied by default a weight of .5 was given. The total complexity of a principle is the sum of the effort of implementing the guidelines (guidelines with options + .5*(guidelines by default)).

Table 5: Complexity of implemented principles.

Principle	With option	By default	Complexity
Prevent errors	4	2	5
Good error messages	1	4	3
Allow users to use the keyboard or mouse	3	0	3
Provide visual cues	2	2	3
Structure the UI		1	1
Strive for consistency		6	3
Offer informative feedback		1	1
Visibility		2	1

4 EVALUATION

This section begins with the description of the performed experimental evaluation. Then, it discusses the limitations of the experimental assessment.

4.1 Results of the Evaluation

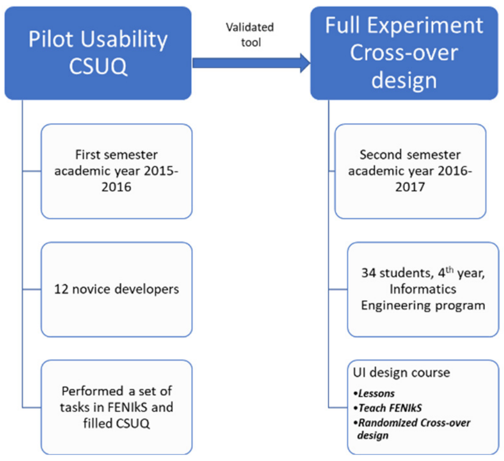

Pilot Usability CSUQ
Validated tool
Full Experiment Cross-over design
First semester academic year 2015-2016
12 novice developers
Performed a set of tasks in FENIkS and filled CSUQ
Second semester academic year 2016-2017
34 students, 4th year, Informatics Engineering program
UI design course
•Lessons
•Teach FENIkS
•Randomized Cross-over design

Figure 7: Sequence of experiments.

A pilot usability experiment was performed to assess FENIkS during the first semester of the academic year 2015-2016. After this pilot experiment a full experiment was conducted during the second semester of the academic year 2016-2017. Figure 7 shows the sequence of both experiments.

The pilot experiment allowed evaluating FENIkS with respect to the perceived usability with 12 novel developers, which had no prior knowledge about the tool. In this experiment the Computer System Usability Questionnaire was used (Lewis, 1993). The tool was perceived positively: the users believe the tool improves their work with respect to the design, facilitates the creation of the presentation model and has the expected functionalities. The quality of the information, the interface and the utility was well perceived too. See (Ruiz et al., 2017) for all the details.

To assess the benefits of the feedback generated by FENIkS about UI design principles, a quasi-experiment was executed with 34 students of the Informatics Engineering program, 4th year, at the University of Holguin. During a UI design course the students learned about UI design principles.

The experiment used a crossover design in which the dependent variable was the learning of UI design principles and the treatment consisted of creating a graphical UI with FENIkS (starting with an already developed conceptual model). The students received lectures about UI design principles and learned how to use FENIkS. Then, they completed Exercises A/B in the same day. The goal of both exercises was answering questions about specific choices in UI design, and whether these are in line with principles or not. Following the crossover design, students were randomly assigned to Group 1 or Group 2. Group 1 first made Exercise A without using FENIkS and then Exercise B with FENIkS, while Group 2 made first Exercise A with FENIkS and Exercise B without FENIkS (Ruiz et al., 2020a). Both tests were composed of equivalent sets of true/false questions about the UI design principles. To avoid the possibility of guessing the answers, the students had to motivate the answers.

Table 6 shows the results of a paired t-test to determine if the support given by FENIkS was effective. The results shows a significant improvement for the scores obtained by the students when using FENIkS support, with 95% confidence interval. The results provide evidence that FENIkS is effective in helping the students understanding design principles. See (Ruiz et al., 2020a) for the full experiment.

Table 6: Paired T-Test for Means of Scores.

$\bar{X}$ score without	$\bar{X}$ score with	$\bar{X}$ difference	p-value
24.06	26.85	-2.79	.000

4.2 Discussion

The experimental evaluation has some validity threats. Regarding to the internal validity we can say that the performed experiment did not include a control group. The rationale behind this decision was that there is a psychological risk in classroom studies where the students may worry about whether and how their participation or non-participation in the experiment will affect their grade: FENIkS is used in a course where students are graded. In this kind of experiment there is the risk of denying half of the group access to a tool that might improve their learning with is also impossible/unethical. Therefore, in line with the ethical considerations, in this research we conducted a quasi-experiment instead of a classic experiment. The problems were mitigated by using a crossover design with two groups. To avoid a maturation effect, the students did not receive feedback after completing the first test.

The validity of the results is limited to the course described in this research. However, the experiment's external validity improved by making the subject population similar to the target population; in this study, novice designers are the target population. A power analysis on our experimental design parameters found that the sample size was adequate to identify significant improvement on the learning of UI design principles, with a large effect size in the performed tests in general and a statistical power of 0.87 for each group of 17 students and 0.99 for the entire group made up of 34 students.

5 CONCLUSIONS

This paper presented an MDE tool for improving the teaching of UI design. The tool allows defining the conceptual and presentation models that are used for the generation of a full working prototype, with the UI code integrated with the application logic.

To assist novice UI designers, the tool incorporates a feedback technique that allows generating automatic feedback about UI design. While designing the UI the novice designer receives feedback about how the options he/she selects ensures compliance with UI design principles. The feedback automation technique is achieved through template-based code generation. It incorporates textual and visual feedback that helps understanding how the UI design principles have been applied and what the consequences are for the generated prototype.

An experimental evaluation was performed with students. The results of the experiment demonstrated that FENIkS improves their understanding of UI design principles, proving its effectiveness. The tool was evaluated for its perceived usability and positively perceived by novice developers.

The proposed technique could be extended to other areas than applying usability principles, such as requirements engineering or programming. A similar approach for other areas could be based on best practices and options reflecting the either good or bad application of those best practices. This requires formulating concrete and testable rules per practice.

The work presented in this paper can also be expanded to further develop the UI generation by:

- Extending the presentation model to improve flexibility by incorporating more design options and new UI design principles. This would also allow providing more learning support with feedback.

- Specifying a user model: The current version does not take into account a user model. The use of such a model would allow enhancing the support for learning UI design in a way that novice learners can check the consequences of different choices according to the user´s skills and characteristics.

- Allowing the designer to generate UIs for other contexts of use: For the moment, the tool addresses the development in one context of use. In order to adapt to the nowadays huge variety of user platforms and environments (contexts of use), the tool needs to be further extended. Since FENIkS relies on MDE and already incorporates an AUI model, future versions can adapt the generation of the interactive software system for other contexts of use and new feedback can be generated for that.

REFERENCES

Akiki, P. A., Bandara, A. K., & Yu, Y. (2015). Adaptive model-driven user interface development systems. *ACM Computing Surveys*, *47*(1). https://doi.org/https://doi.org/10.1145/2597999

Aquino, N., Vanderdonckt, J., & Pastor, O. (2010). Transformation templates: adding flexibility to model-driven engineering of user interfaces. *SAC´2010*, 1195–1202.

Barrett, M. L. (1993). A hypertext module for teaching user interface design. *ACM SIGCSE Bulletin*, *25*(1), 107–111.

Benitti, F. B. V., & Sommariva, L. (2015). Evaluation of a game used to teach usability to undergraduate students in computer science. *Journal of Usability Studies*, *11*(1), 21–39.

Beuvens, F., & Vanderdonckt, J. (2012). Designing graphical user interfaces integrating gestures. *Proceedings of the 30th ACM International Conference on Design of Communication*, 313–322.

Butchart, S., Forster, D., Gold, I., Bigelow, J., Korb, K., Oppy, G., & Serrenti, A. (2009). Improving critical thinking using web based argument mapping exercises with automated feedback. *Australasian Journal of Educational Technology*, *25*(2).

Lewis, J. R. (1993). *IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use* (Vol. 12, Issue 2).

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

Lisowska Masson, A., Lalanne, D., & Amstutz, T. (2017). A Usability Refactoring Process for Large-Scale Open Source Projects: The ILIAS Case Study. *2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 1135–1143.

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

Molina, P. J., Meliá, S., & Pastor, O. (2002). User interface conceptual patterns. *International Workshop on Design, Specification, and Verification of Interactive Systems*, 159–172.

Mosteller, D., Haustermann, M., Moldt, D., & Schmitz, D. (2018). Graphical Simulation Feedback in Petri Net-based Domain-Specific Languages within a Meta-Modeling Environment. *PNSE@ Petri Nets/ACSD*, 57–76.

Nielsen, J. (1994). *Heuristic evaluation. Usability inspection methods* (J. W. & Sons (ed.)).

Ormeño, Y. I., Panach, J. I., Condori-Fernández, N., & Pastor, Ó. (2014). A Proposal to Elicit Usability Requirements within a Model-Driven Development Environment. *International Journal of Information System Modeling and Design (IJISMD)*, *5*(4), 1–21.

Parvez, S. M., & Blank, G. D. (2008). Individualizing tutoring with learning style based feedback. *International Conference on Intelligent Tutoring Systems*, 291–301.

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

Ruiz, Jenny, Serral, Estefanía, & Snoeck, M. (2019). Learning UI Functional Design Principles through Simulation with Feedback. *Journal IEEE Transactions on Learning Technologies*.

Ruiz, J., Sedrakyan, G., & Snoeck, M. (2015). Generating User Interface from Conceptual, Presentation and User models with JMermaid in a learning approach. *Interaction' 2015*. https://doi.org/http://dx.doi.org/10.1145/2829875.2829893

Ruiz, J., Serral, E., & Snoeck, M. (2019). Technology Enhanced Support for Learning Interactive Software Systems. In S. B. Hammoudi S., Pires L. (Ed.), *Model-Driven Engineering and Software Development. MODELSWARD 2018. Communications in Computer and Information Science* (pp. 185–210). Springer, Cham. https://doi.org/https://doi.org/10.1007/978-3-030-11030-7_9

Ruiz, J., Serral, E., & Snoeck, M. (2020a). Learning UI Functional Design Principles through Simulation with Feedback. *IEEE Transactions on Learning Technologies*, *13*(4), 833–846.

Ruiz, J., Serral, E., & Snoeck, M. (2020b). Unifying functional User Interface design principles. *International Journal of Human-Computer Interaction (IJHCI)*.

Ruiz, J., Serral, E., & Snoeck, M. (2018). A Fully Implemented Didactic Tool for the Teaching of Interactive Software Systems. *Modelsward'2018*, 95–105.

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

Sboui, T., & Ayed, M. B. (2016). Generative Software Development Techniques of User Interface: Survey and Open Issues. *International Journal of Computer Science and Information Security*, *14*(7), 824.

Schlungbaum, E. (1996). *Model-based user interface software tools current state of declarative models*. Georgia Institute of Technology.

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

Sedrakyan, G., & Snoeck, M. (2013b). A PIM-to-Code requirements engineering framework. *Modelsward'2013*, 163–169.

Sedrakyan, G., Snoeck, M., & De Weerdt, J. (2014). Process mining analysis of conceptual modeling behavior of novices–empirical study using JMermaid modeling and experimental logging environment. *Computers in Human Behavior*, *41*, 486–503.

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

Serral Asensio, E., Ruiz, J., Elen, J., & Snoeck, M. (2019). Conceptualizing the Domain of Automated Feedback for Learners. *Proceedings of the XXII Iberoamerican Conference on Software Engineering, CIbSE 2019,*.

Shneiderman, B., & Plaisant, C. (2005). Designing the user interface 4 th edition. *Ed: Pearson Addison Wesley, USA*.

Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, *78*(1), 153–89.

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

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

Trewin, S., Zimmermann, G., & Vanderheiden, G. (2004). Abstract representations as a basis for usable user interfaces. *Interacting with Computers*, *16*(3), 477–506.

Van Eck, R. (2006). Digital game-based learning: It's not just the digital natives who are restless. . *Educause*, 16–30.