Научна конференция „Иновационни софтуерни инструменти и технологии с приложения в научни изследвания по математика, информатика и педагогика на обучението“, 23-24 ноември 2017 г., Пампорово
Scientific Conference „Innovative Software Tools and Technologies with Applications in Research in Mathematics, Informatics and Pedagogy of Education”, 23-24 November 2017, Pamporovo, Bulgaria

APPROACHES FOR GENERATING ADAPTIVE CONCRETE MODEL AND FINAL USER INTERFACE

Margarita Atanasova

Abstract. In this article we briefly present a software prototype for model-based adaptive user interface development. The motivation is to propose and implement a structured approach, which is intuitive and easy-to-use for developers, product managers, clients, sales representatives and UI designers. The goal is to speed up the development of adaptive UIs by providing an environment, which facilites the whole process: from business requirements analyses, through the UI modelling to the final UI generation. The focus in this paper is on programming approaches for automatic generation of concrete UI model and final UI. For platform-independent adaptive UI specification we use XML. We review different programming libraries that we have used in the development process. We also propose an approach for implementing corporate identity and adding properties for context-of-use specific information to the final product. The system architecture of the prototype environment is presented. It is described how this architecture allows adding different platform-specific languages and front-end frameworks used for the final UI.

Key words: adaptive user interfaces, programming approach, context of use, model-based development

1. Introduction

Model-driven development has become one of the leading approaches for rapid software development involving all stakeholders in the process [1], [2], [3]. The methodology uses visual models to describe data, business logic, user interfaces and more. Automated or semi-automated tools and processes eliminate the need for labour-intensive writing of huge amount of code [4], [5], [12], [14], [15], [16], [17], [18].

Our goal is to propose an online integrated environment (Iris Studio) that provides structured approach for describing and building adaptive user interfaces, maintaining the entire development cycle. Our focus is the environment to be intuitive and easy to use not only by programmers but also by product managers, stakeholders, sales representatives and designers. The aim is to support all steps: analysing client requirements, sketching the prototype, defining context of use for the UI elements and eventually generating the final code. This would allow programmers, designers, managers and clients to work together to clarify the requirements before the actual development, which will help define a proper budget for the project. Creating a quality prototype would speed up and facilitate contracting new customers as well as reduce the development cost.

A single user interface (UI) could be described using different models: goals model, tasks model, domain model, context model, functional core model, abstract UI model, concrete UI model and more. In order to be used easily in the real work environment the models should have a steep learning curve and should not complicate the development process. The Cameleon Reference Framework defines four levels in the model-based user interface development (MBUID) cycle: tasks and domain model, abstract, concrete and final UI models [6]. It is very well accepted in the MBUID community and it is proposed for standardization by the W3C Incubator Group [7]. Because of that we adopted the four levels of MBUID proposed by the Cameleon Reference Framework, modifying it in the context of adaptive user interfaces for business information systems.

In this article we present the programming approach we have used for transforming an abstract user interface model into a concrete user interface model and for generating a final user interface. In the process, we added a functionality for defining and applying contexts-of-use for the different UI elements. In the final step we generate a platform-independent UI specification using XML and platform-dependant specification using HTML and CSS.

2. Concrete User Interface

The concrete user interface (CUI) is a UI model that describes the appearance and interactivity of the user interface in a manner that can be perceived and manipulated by human [8]. The CUI model is built by Concrete Interaction Objects (CIO) and concrete relationships. A CIO could be a concrete container (window, box) or a concrete individual component (text area, radio button, checkbox, button, etc.). While the abstract user interface model is a formal platform-independent description of a user interface without details of the visual layout or behaviour of the system [9], the CUI represents the abstract user interface in terms of visual elements. Yet there is no standard notation for the CUI [10].

3. Transforming Task Model and Abstract User Interface into a Concrete User Interface

In this section we explain the programming approach and the transformations that we make to convert the task model and the abstract user interface into a concrete model in our integrated environment.

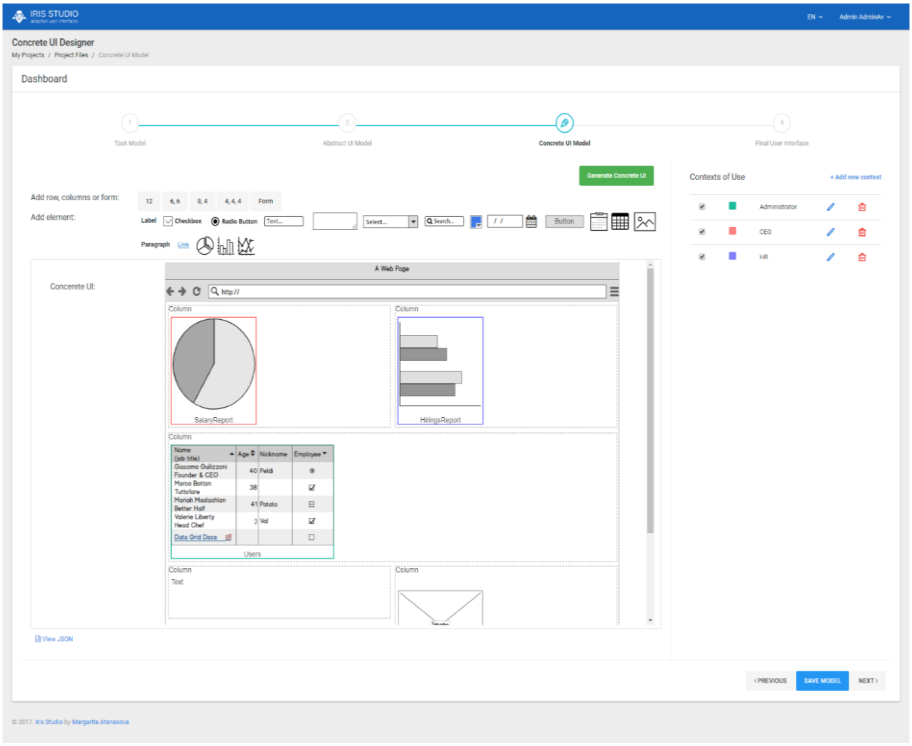
In the first step of the UI modelling process we define the task model using the ConcurTaskTrees notation [11]. We store the task model in a JSON format. To determine the structure of the concrete UI we start by analysing the topology of the task tree. We use the root task to define a new window (a single user interface). Next, we define the concrete containers and the concrete individual components. In this step the user of Iris Studio should define those concrete components which cannot be automatically defined.

Abstract Interaction object	Concrete Interaction Object
Input	text input field, text area, search field, radio button, select box, checkbox, data picker, color picker
Output	label, paragraph, table, image, calendar, pie chart, line chart, bar chart
Control	button
Navigation	link
Container	window, form, box

Table 1. Corresponding abstract and concrete interaction objects in Iris Studio

For example, an abstract object of input type could be a textbox, data field, color picker, etc. The user must select the one needed. For each abstract type we define possible concrete interaction objects (Table 1).

When the concrete UI is generated each concrete object appears in a new row. The user of Iris Studio can add additional rows and columns and rearrange the objects if needed. We use 12-column grid system so that the final UI can support different screen sizes and front-end frameworks. On Figure 1 the generated Concrete UI is presented. When generating the final UI, we take the values of the columns in the concrete UI layout and we use them to create a grid with appropriate classes corresponding to the defined ones in Bootstrap library (https://getbootstrap.com/).



Name (job title)	Age	Nickname	Employee
Giacomo Guilizzoni Founder & CEO	40	Peldi	◉
Marco Botton Tuttofare	38		☑
Mariah Maclachlan Better Half	41	Patata	⊟
Valerie Liberty Head Chef	:)	Val	☑
Data Grid Docs			☐

Figure 1. The CUI model in Iris Studio

The content of each cell and the order of the rows and columns are also reflected in the XML file, so no matter what technology is used, the information about the order of the components is preserved.

We have implemented two-way binding of the models so that when the user makes changes to one of the models this reflects the other models. For example: the user adds a button in the concrete UI model. This triggers an event which adds an abstract interactive object of type Control in the corresponding container in the abstract model and an Interaction task type in the task tree model. Any changes that are made on the text labels are also automatically transferred to the other models. Since each specific component has a single abstract object type and a task type, we automatically update the models using the relevant components (Table 2). This means that the editing is done in one place, but visible in all models.

Concrete Object →	Abstract Object →	CTT Task Type
Text...	Input facet	Interaction Task
Pie Chart	Output facet	System Task
Link	Control Facet	Interaction Task
Picture	Output facet	System Task

Table 2. Examples for reversed transformations in the models in Iris Studio

4. Final User Interface

The final user interface represents the user interface in terms of platform-specific implementation and programming code. It can be interpreted to a variety of front-end UI programming languages, such as Java GUI, HTML and CSS, AngularJS, etc. The adaptivity to the context of use is essential as it provides personalized and individual solutions [17], [18], [19]. This is quite important when it comes to creating educational software that automatically generates tests according to the skill level of the learner, software that makes text analyses or helps in learning new language [12], [13], [14], [15], [16], [17], [18].

In the Iris software prototype, we have developed an automatic generation of platform-independent description for an adaptive user interface in the form of an XML file. We consume it to create a platform-dependent description to generate a final user interface with HTML, CSS and JavaScript. For good visual layout of the end result we use the Bootstrap framework.

5. Transforming the Concrete UI Model into a Final UI

After the concrete UI is generated in the previous step, we have the relative positioning of the objects on the page as well as their concrete types.

We designed the architecture of the system and the database so that front-end technologies can be easily added. Table "mappings" in the database stores information about the technology-dependent implementations of the concrete objects. It consists of 4 columns: id, concrete object, bootstrap, xml. For the purpose of our study, we have added two technologies in this table – HTML and XML. For the values in these columns we use a Binary Large Object type because they might contain big chunks of executable code (Table 3).

Concrete Object	Bootstrap	XML

textarea	`<div class="form-group" data-context="###context###">`
	`<textarea class="form-control">`
	`###text###</textarea>`
	`</div>`
		`<textareaComponent data-context="###context###" data-placeholder="###text###">`
		`</textareaComponent>`

selectbox	`<div class="form-group" data-context="###context###">`
	`<select class="form-control">`
	`<option> ###text### </option>`
	`<option>2</option>`
	`<option>3</option>`
	`<option>4</option>`
	`</select></div>`
		`<selectComponent data-context="###context###">`
		`<option> ###text###</option>`
		`</selectComponent>`

Table 3. A sample of the mappings table from the Iris Studio database

We use words enclosed with “#” symbols to create reserved words in the mappings table: ###text###, ###content###, ###columns###, ###context###. We replace them with data, which we gather during the modelling process. Each task in the task model has a name, which is stored in a label. We replace the reserved word “###text###” with this label to create value or a placeholder in the final element. The transformation of the system task “Sales” is shown on Figure 2. In this case “Sales” label is transferred between all models. The final chart which is used to present this task uses the same name.

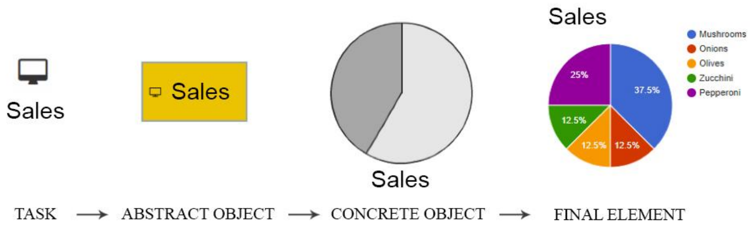


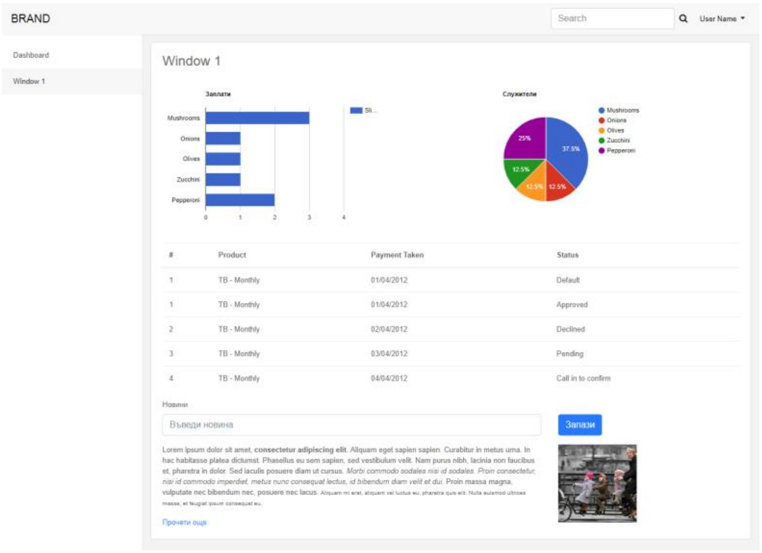
Figure 2. Transforming system task “Sales” to an HTML element.

The reserved word “###columns###” is used purely for visual purposes. We store information about the position of the object on the page using rows and columns in the concrete UI. This data replaces the word ###columns### in the programming code and help us build responsive layout for the final UI. This information can be consumed by any modern front-end framework to place the UI elements in a grid therefore supporting modern look and feel for all platforms.

For building the XML structure and the HTML DOM tree we created a recursive programming algorithm which replaces ###content### with a fragment of code. In this function we also replace ###text### and ###columns###. This function checks if the element has children. If so, the function calls itself. If not, we use string replace to fill in the ###content### of the element. That way, for example, we put a text field in a column, which is placed in a row.

When the function is executed it returns the body of the final page. The result is placed in a predefined page template. For the purposes of the study we have defined two master page templates – one HTML with Bootstrap and one XML. A master template defines how the page will look. It contains the global parameters and a dynamic content placeholder, where the final generated UI is displayed. That way we ensure that the final system will have a consistent look and feel. In the master page template, we define the head includes, all linked resources, the structure of the header, the top and left navigations, the footer and the main container.

For the final UI editor in Iris Studio we use an HTML iframe. This element is used to load an external HTML in the current document. We use an iframe for the real-time preview of the final UI because it is supported by all browsers. That way the user can make changes and implement corporate identity, immediately see the result and save it. However, the browsers cache the information in the iframe in different ways. When one of the models (task model, abstract and concrete) were changed, the iframe didn’t show any changes on the final UI. To solve this problem and to make the content in the iframe update we created a feature that takes the current date and time and adds them to the URL. By using different URL when reloading the page, the iframe was forced to reload the content instead of using cached files. The final UI generated from the concrete UI from Figure 1 is shown on Figure 3. The files of the final UI can be downloaded and used in an actual business software. Each element contains meta-information about the context of use. We use the reserved word ###context### in the code to point the place where the defined contexts should be placed. For this purpose, we use data-context attribute in each element. If the context name contains more than one word, we add “_” between the words. For example, a system task “Employees” might have contexts of use “HR specialist” and “admin”. When we create the code for this element it looks the following way: <element data-context=”hr_specialist admin”> … </element>. We separate the different contexts by using whitespace.



#	Product	Payment Taken	Status
1	TB - Monthly	01/04/2012	Default
1	TB - Monthly	01/04/2012	Approved
2	TB - Monthly	02/04/2012	Declined
3	TB - Monthly	03/04/2012	Pending
4	TB - Monthly	04/04/2012	Call in to confirm

Lorem ipsum dolor sit amet, **consectetur adipiscing elit**. Aliquam eget sapien sapien. Curabitur in metus urna. In hac habitasse platea dictumst. Phasellus eu sem sapien, sed vestibulum velit. Nam purus nibh, lacinia non faucibus et, pharetra in dolor. Sed iaculis posuere diam ut cursus. *Morbi commodo sodales nisi id sodales. Proin consectetur, nisi id commodo imperdiet, metus nunc consequat lectus, id bibendum diam velit et dui.* Proin massa magna, vulputate nec bibendum nec, posuere nec lacus. Aliquam mi erat, aliquam vel luctus eu, pharetra quis elit. Nulla euismod ultrices massa, et feugiat ipsum consequat eu.

Figure 3. Final UI generated in Iris Studio

The system’s corporate identity can also be defined using Iris Studio. The designer can upload a logo, add link to google fonts and set colors. All these parameters are saved in the database and applied to all files in the project so that the designer does not need to implement the corporate identity for every single page. The link to the Google Fonts and the other global corporate identity parameters are placed in the master page template for this project and can be downloaded together with the final files.

6. Charts

One of the most important functions in the business information systems is reporting. Usually it is presented in the form of tables and charts. In Iris Studio we have added Google Charts to support different types of charts in the final UI. Google Charts is an open-source library for drawing charts in an HTML canvas (https://developers.google.com/chart/). It is supported by all browsers (including older versions of IE) and platforms (iOS, Android). We have implemented 3 types of charts: pie, bar and line. In a similar way, all types of charts supported by the library can be added. For example, to add a chart type “Map”, the programmer should follow several easy steps: create a concrete object “map” with a graphical representation and add the code in the mappings table. An intuitive user interface for adding different chart types and plugins can be added to Iris Studio as a perspective to the project.

All the above provides the designer with the ability to quickly create dashboards, admin panels and reports and test them with real users shortly after. The step with the labour-intensive writing of code, searching and implementing libraries is eliminated.

7. Conclusion

Iris Studio was tested with 3 designers in a real work environment. They were all able to quickly produce a final user interface. After a short overview, they learned how to model a UI by themselves and created several pages. We will continue examining the strengths and weaknesses of Iris Studio as it is a prototype environment. The database and the architecture of the software gives us the possibility of adding different technologies for the final UI as well as different front-end and charting libraries. Using one UI model, the designer will be able to produce both desktop and web applications in short periods of time which will greatly lower the cost of any project.

References

[1] Bertti, E., D. Schwabe, MIRA: A Model-Driven Framework for Semantic Interfaces for Web Applications, *International Conference on Web Engineering*, Springer International Publishing Switzerland, 2016, 40–58.

[2] Akiki, P., A. Bandara and, Y. Yu, Engineering Adaptive Model-Driven User Interfaces, *IEEE Transactions on Software Engineering*, 42 (12), 2016, 1118–1147.

[3] mendix.com, last visit: Jan 2018.

[4] Cheresharov, S., Krushkov, H., Stoyanov, S., Popchev, I., Modules for Rapid Application Development of Web-Based Information Systems (RADWIS), *Cybernetics and Information Technologies*, 17(3), 2017, 109–127.

[5] Stoeva, M. and N. Pavlov, Remote Tool for Controlling User Interface of Mobile Applications, *MCIS 2017 Proceedings*, 34, 2017.

[6] Calvary, G., J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon and J. Vanderdonckt, A Unifying Reference Framework for Multi-Target User Interfaces, *Interacting with Computers, 15* (3), 289–308, 2003.

[7] W3C Incubator Group, *Model-Based UI XG Final Report*, 2010.

[8] Molina, A., W. Giraldo, J. Gallardo, M. Redondo, M. Ortega and G. García, CIAT-GUI: A MDE-compliant environment for developing Graphical User Interfaces of information systems, *Advances in Engineering Software*, *52*(Supplement C), 2012, 10–29.

[9] Constantine, L., Canonical Abstract Prototypes for Abstract Visual and Interaction Design. *Interactive Systems. Design, Specification, and Verification: 10th International Workshop*, Funchal, Madeira, 2003, 1–15.

[10] Pleuss, A., G. Botterweck and D. Dhungana, Integrating Automated Product Derivation and Individual User Interface Design. *Fourth International Workshop on Variability Modelling of Software-Intensive Systems.* Linz, Austria, 2010.

[11] Paternò, F., ConcurTaskTrees: An Engineered Notation for Task Models. In D. Diaper, & N. Stanton (Eds.), *The Handbook of Task Analysis for Human-*Computer Interaction, Mahwah, New Jersey, USA: Lawrence Erlbaum Associates, Inc, 483–503, 2003.

[12] Malinova, A. and O. Rahneva, Automatic generation of English language test questions using Mathematica, *CBU International Conference on Innovations in Science and Education*, March 23-25, 2016, Prague, Czech Republic, Vol. 4: CBU International Conference Proceedings, 2016, 906–909.

[13] Cheresharov, S., H. Krushkov and M. Krushkova, NLP Module for Bulgarian Text Processing, *CBU International Conference on Innovation in Science and Education*, Prague, Czech Republic, 2017, 1113–1117.

[14] Malinova A., O. Rahneva and A. Golev, Automatic Generation of English Language Test Questions on Parts of Speech, *International Journal of Pure and Applied Mathematics – IJPAM* , Vol. 111, No. 3, 2016, 525–534.

[15] Malinova, A., V. Ivanova and A. Rahnev, Computer algebra aided generation of English language tests, *The 11th Annual International Conference on Computer Science and Education in Computer Science CSECS 2015*, Boston, MA, USA, June 04–07 2015, Computer Science Education & Computer Science Research Journal: ISSN 1313-8624, Vol. 11, 2015, 66–74.

[16] Rahnev, A., A. Malinova and N. Pavlov, Parameterized examionation in DisPeL, *Proceedings of the International Conference "FROM DELC TO VELSPACE"*, Plovdiv, 26–28 March, 2014, 263–272, Third Millennium Media Publications, ISBN: 0-9545660-2-5 (in Bulgarian).

[17] Rahnev A., N. Pavlov and V. Kyurkchiev, Distributed Platform for e-Learning – DisPeL, *European International Journal of Science and Technology (EIJST)*, Vol. 3, No. 1, ISSN 2304-9693, 2014, 95–109.

[18] Pavlov N., A. Rahnev, K. Mirchev, T. Gardjeva and D. Krushkova, Responsive User Interface for People with ASD, *International Journal of Pure and Applied Mathematics – IJPAM*, Vol 111, No 1, 2016, ISSN 1311-8080 (printed version), ISSN 1314-3395 (on-line version).

[19] Pavlov, N., A. Rahnev, V. Kyurkchiev, A. Malinova and E. Angelova, Geographic map visualization in DisPeL, *Proceedings of the Scientific Conference "Innovative ICT in Business and Education: Future Trends, Applications and Implementation"*, Pamporovo, 24–25 November 2016, ISBN: 978-954-8852-72-2, 2016, 13–20, (in Bulgarian).

Faculty of Mathematics and Informatics
Plovdiv University
236 Bulgaria Blvd,
Plovdiv 4003, Bulgaria
E-mail: margi_hk@yahoo.co.uk

ПОДХОДИ ЗА ГЕНЕРИРАНЕ НА АДАПТИВЕН КОНКРЕТЕН МОДЕЛ И ФИНАЛЕН ПОТРЕБИТЕЛСКИ ИНТЕРФЕЙС

Маргарита Атанасова

Резюме. В тази статия накратко представяме прототип на среда за моделно-ориентирана разработка на адаптивни потребителски интерфейси. Мотивацията за това проучване е да предложим и имплементираме структуриран подход, който е интуитивен и лесен за приложение от разработчици, продуктови мениджъри, специалисти от отдел продажби, дизайнери и други. Целта е да се ускори процесът на разработка на адаптивни потребителски интерфейси чрез използването на софтуер, който подпомага целия цикъл на разработка: от анализирането на бизнес изискванията, през моделиране и скициране до генериране на финалния код. Фокусът на тази статия е да се представят програмни подходи за автоматично трансформиране на конкретен модел на потребителски интерфейс във финален код. За платформено-независимо описание на финалния потребителски интерфейс използваме XML. Разглеждаме още различни програмни библиотеки, които използваме по време на разработката. Описваме подход за имплементиране на корпоративна идентичност и добавяне на адаптивност на елементите чрез дефиниране на контексти на употреба. Представени са още системната архитектура и части от базата данни и как това предоставя възможност за добавяне на различни езици за програмиране и работни рамки, с които да се генерира финален потребителски интерфейс.

Ключови думи: адаптивни потребителски интерфейси, програмни подходи, контекст на употреба, моделно-ориентирана разработка