MODUS: Model-based user interfaces prototyping

Marina Machado
HASLab/INESC TEC &
Departamento de Informática /
Universidade do Minho
Braga, Portugal
pg25336@alunos.uminho.pt

Rui Couto
HASLab/INESC TEC &
Departamento de Informática /
Universidade do Minho
Braga, Portugal
rui.couto@di.uminho.pt

José Creisssac Campos
HASLab/INESC TEC &
Departamento de Informática /
Universidade do Minho
Braga, Portugal
jose.campos@di.uminho.pt

ABSTRACT

Model-based methodologies, supported by automatic generation, have been proposed as a solution to reduce software development costs. In the case of interactive computing systems specific challenges arise. On the one hand, a high level of automation requires the use of detailed models, which is contrary to the iterative development process, based on the progressive refinement of user interface mockups, typical of user centered development processes. On the other hand, layered software architectures imply a distinction between the models used in the business logic and in the user interface, raising consistency problems between the models at each level. This article proposes a tool supported approach to user interface generation directly from the architectural models of the business logic. In many situations, user interfaces provide similar features inside a specific domain. The identification of the application domain is thus a key factor in supporting the automation of the generation process.

ACM Classification Keywords

D.2.2. Software Engineering: Design Tools and Techniques; H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces

Author Keywords

Model-driven engineering; Model-based user interface development; Automated user interface generation.

INTRODUCTION

Model-based tools propose to facilitate and accelerate the development of User Interfaces (UIs) by providing a degree of automation to the development process. An increase in the degree of automation, however, is usually accompanied by a need for the inclusion of additional information by the user of the tool, making the development of the models a complex activity [12]. Additionally, tools that focus on automation tend to be criticized for poor integration in the software development process. On the one hand, model-based generation tends to have problems integrating with the creative process of designers, having a negative impact on the quality of the generated UI [13]. On the other hand, the models used tend to diverge from the business-layer models used in Model Driven Engineering (MDE) approaches, raising coordination and productivity problems [12].

This paper presents MODUS (MOdel-based Developed User Systems), a tool supported approach for the semi-automated generation of user interfaces for Web applications. MODUS presents the following contributions: a) it takes advantage of the separation between Web UIs' content definition and graphical layout to provide designers with the capability to refine the generated UI; b) it makes use of the business logic architecture of the system (as a class diagram) as a starting point for the generation process, thus promoting integration between business logic and UI development; c) it takes advantage of a domain definition for the application to limit the need for additional models in the generation process, enabling a substantially more automated interface generation (indeed, for the same application domain, it is found that user interfaces tend to be similar in navigation, structure and visual components).

In MODUS, the concept of *Evolutionary Prototyping* [7] is fundamental. Developers have the ability to edit and refine the generated interface at all stages of the development process until the final interface is obtained, thus supporting an iterative approach to development. The tool supports the selection of front-end frameworks and templates to be used, as well as the configuration of the elements in the UI. In this way, the developer have a greater control over the visual aspect of the final result.

BACKGROUND

Model-driven engineering vs Model-based user interface development

In the Model-Driven Engineering (MDE) paradigm [20, 17], high-level (abstract) models of the problem the system should provide a solution to, are successively refined into (more concrete) models at lower-levels of abstraction, with the ultimate goal of transforming them into an executable systems [20, 17]. This conversion process can be manual or automated. In both cases the goal is to ensure a coherent software development process, ensuring the quality and correction of the final result. Resorting to an automatic transformation of the models into the executable code, leads to a reduction in development time [19].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

EICS '17, June 26-29, 2017, Lisbon, Portugal

© 2017 ACM. ISBN 978-1-4503-5083-9/17/06…$15.00

DOI: http://dx.doi.org/10.1145/3102113.3102146

MDE is typically applied to the creation of the business logic and data layers of systems, but not so much the user interface [11]. Similar efforts to reduce the amount of time and effort expended in design and development of user interface, while guaranteeing their quality, lead to Model-based User Interface Development (MBUID) approaches [19, 5].

The Cameleon Reference Framework, widely accepted as the reference architecture for MBUID, specifies four main levels of modeling [3, 22, 23]:

1. **Domain Model and Tasks** - description of user tasks and domain concepts related to their realization.
2. **Abstract User Interface (AUI)** - description of the user interface in terms of Abstract Interaction Units or Abstract Interaction Objects and their relations. A UI is represented independently of any technology or mode of interaction.
3. **Concrete User Interface (CUI)** - description of the user interface in terms of Concrete Interaction Units or Concrete Interaction Objects, determining layout and interface navigation. A CUI, being dependent on the interaction mode, describes the application *Look & Feel*.
4. **Final User Interface (FUI)** - description of the user interface in terms of the source code, either in a programming language or mark-up. A FUI can be interpreted or executed after compiling the code.

Generation in MBUID starts with high-level declarative models. However, very detailed models of the various aspects of the interface are often needed. This fact limits the acceptance of the methodology, since the elaboration of the interface implies increased modeling costs; either in the number of models or in the effort of elaboration of each one of them [19, 5].

Additionally, the type of models used in MBUID (task models, UI models) is quite different from those of MDE, meaning that the two approaches are not as integrated as would be ideal. Approaches such as [6] addressed the problem by exploring how to adapt the modeling languages used in MDE to express the models needed for MBUID. While this solves the need to learn different modeling languages for each layer's development process, it is still necessary to develop different models for each of the layers.

We address this issue, proposing an approach based on a semi-automated UI generation process that starts from the (MDE) architectural model of the system. Together with the application domain's specification, this model is the basis for the partial or complete generation of the UI, which can then be refined with the support of suitable tools.

MBUID tools

The first MBUID tools were typically based on a universal declarative model, focusing on the idea of a fully automatic generation process (e.g. [21, 16]). The created UIs tended to be simple desktop applications, based on *Create, Read, Update and Delete* (CRUD) operations, always following the same visual model. Over time, the UI model became a composition of declarative models, aiming to improve the quality of the generated interfaces, at the expense of specifying additional

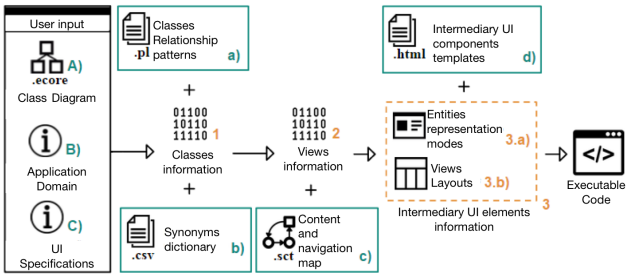

Figure 1. MODUS overview

models. In these tools the level of automation tended to be low. An increase in the degree of automation implied the inclusion of new models with additional information about the interface. Currently, work on MBUID tools is multifaceted. A major concern are the challenges brought by the emergence of new devices and platforms, from UIs that adapt to the computational platform in use [1, 15] to the generation of families of UIs [14]. The need to better support the development process has also been acknowledged [8, 9].

In general, a tension can be observed between the degree of automation, the modeling effort required, and the customizability of the generated solutions. More automated tools (e.g. Integranova M.E.S.[1]) either require an higher modeling effort, or are based on restricted models of interaction (typically CRUD operations), generating rigid UIs that lack in appeal to the user. More flexible tools (e.g. OutSystems[2]) are less automated, needing more user intervention, which increases development costs.

The approach proposed in this paper intends to reconcile these two aspects, seeking to contribute to a solution that, from a minimal set of models, is able to produce user interfaces which might be tailored to specific needs. The approach is largely orthogonal to concerns about adapting to the context, focusing at this stage on providing an efficient solution for development, starting from architectural models of the business layer. For web application UIs, the use of front end technologies such as JavaScript and CSS3 contributes to a significant increase in compatibility between different devices and platforms at run time [18]. The integration of these technologies into the generation process may be an alternative to specifying the context of use in the models.

THE MODUS APPROACH

Figure 1 presents an overview of the MODUS approach, from reading the models to generating a (web) UI. Each step of the process is numbered. *User* inputs (for a specific application and domain) are labeled with capitalized letters, while tool inputs (generic data across domains) in lower case letters. Arrows represent information flow.

The process starts with (c.f. Figure 1): A) the business logic architecture for the system as a class diagram (*ecore* [2] is the supported format); B) an identification of the target application

[1] http://www.integranova.com
[2] http://www.outsystems.com

```
1 entity(buyer).
2 entity(address).
3 relation(has,buyer,address,0,n).
```

Listing 1. Example of Prolog ontology facts.

```
1 ecommerce_address_pattern(A):-
2   entity(A),              % Address is an entity
3   relation(_R,U,A,_M,_N), % User has Address
4   has_at_least(O,A,1), % Order has at least 1
        Address
5   different(A,U),different(A,O),different(O,U).
```

Listing 2. Example of a Prolog relationship pattern

domain; C) optionally, additional information regarding the UI generation (e.g. the front-end and the framework to use). The user interface generation proceeds through three steps. In the first step (labeled 1, in Figure 1), the relevant entities in the business logic model are identified using the available architectural patterns and semantic knowledge database for the domain. In the second step, the concrete UI is defined. This makes use of a predefined abstract user interface model for the particular domain. In the third step, the final UI is specified, resorting to available templates. This includes defining the presentation modes for the entities, and views layouts.

Standard Classes inference

The identification of relevant entities in the architectural model makes use of the notion of *standard classes*, i.e. classes commonly found in a specific application domain. An example is the `Article` class (entity) in the *eCommerce* application domain. The identification of these classes supports the identification of which classes in the architecture represent relevant domain entities, and guides the development process. The process of associating a domain entity to a class in the architectural model consists of two steps. The first step corresponds to identifying, in the class diagram, the architectural pattern in which the *standard* class exists. The second step consists in comparing the class and the entity level, in order to retrieve a matching value.

For the first step, an inference process based in [4] is used. The class diagram is translated into a *Prolog* ontology. Listing 1 presents an example where a buyer can have several addresses. Lines 1 and 2 declare the existence of entities `buyer` and `address`. Line 3 declares the existence of a relationship (`has`) between the two entities. The `0` and `n` parameters represent the lower and upper bounds of the multiplicity (of `address`) in the relationship. Queries are then used to verify the existence of patterns. A knowledge base defines the relevant patterns for the domain. A pattern corresponds to a set of relationships which exist between a *standard class* (the candidate entity) and other classes in the model. Listing 2, defines the pattern for the `Address` *standard class*, in the *eCommerce* domain. This pattern defines that `A` (`Address`) must be an entity of the ontology, a class `U` (the `User`) must exist that has addresses, and some other class `O` (the `Order`) must have, at least, one address. `A`, `U` and `O` must be distinct entities.

Patterns can overlap, which means that for any given *standard classes* in a pattern more than one potential match in the architectural model might be identified. A mechanism is required for selecting the most appropriate candidate. This is achieved in the second step of the process.

In the second step, for each *standard class* there is a dictionary entry, which aggregates its synonyms, alongside their probability to match that class's name. The name of an entity in the model is searched for in the dictionary. The matching percentage varies according to the name similarity, and probability defined in the dictionary. The entity with the biggest probability is then associated to the *standard class* in question. A similar analysis is performed for classes attributes.

Concrete UI – Content and Navigation Map

MODUS generates Web UIs, hence constituted by a set of interlinked web pages. Each page will have a layout, which defines its visual structure, and a set of entities, which define its contents. Representation modes define how an entity should be presented.

MODUS resorts to an auxiliary model to represent the UI to be generated, the contents and navigation map. This consists of a diagram (based on UML's Statecharts) that defines all of the information required to create the UI. It defines the pages and page fragments (*partials*) of the UI, as well as the navigation between the different elements. Due to the need to represent not only navigation, but also UI contents in the model, the elements of the content and navigation map are interpreted as follows (cf. Figure 2):

State Represents a full page, a partial (a page fragment), a page section or an entity's representation. The specific role of a state is defined by annotations in its name and its location in the diagram. If a state is inside another state, then it is an element of the page represented by that state (e.g. `#left-sidebar`). Partials are identified, by preceding their names with `'_'` (e.g. `_index_list_product`). Specific page elements or entity identifiers are delimited by `'<' '>'` (e.g. `<product>`). Remaining states identify pages (e.g. `homepage`). States contain attributes to represent auxiliary information.

Final state Represents an exit transition.

Initial state Represents the landing page of the application.

Transition Represents either a transition, a substitution or a composition, depending on the target element. A transition to a page represents navigation to that page. A transition to a partial (see label 7) represents replacing the current page's contents with the partial's contents. A transition to an entities representation represents content composition.

Condition Denotes the set of identifiers related with a transition (see label 8).

The example in Figure 2 describes a UI with an initial `homepage`, which has a section `#left-sidebar`, which contains the partial `_index_sidebar_category`. The partial is composed by the repetition of `_show_sidebar_category`, which in turn includes the `sidebar` representation of the

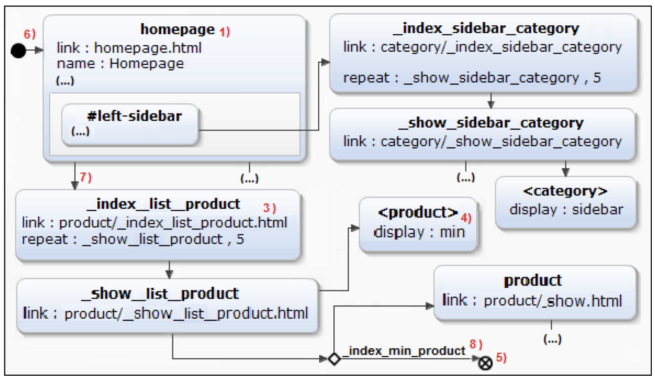

Figure 2. Extract of a contents and navigation map

`Category` class. This combination, is commonly used to represent a list of a partial, or an entity according to a given mode. The `homepage` is composed of a list of `Product` displayed using the `min` representation, and each element redirects to the `Product` page, unless the state is `_index_min_product`.

Final UI components generation

For UI generation to be possible for a given technology, mappings from representations modes and layouts to that technology must be available. HTML templates are provided by the tool to support this mapping. The templates can be modified by the user at runtime, thus personalizing the components that will compose the FUI. UI components are created by parsing these templates and adapting them to the provided input.

This templates-based generation process, allows the generation algorithms to be generic. Adding, removing or modifying the templates does not affect the generation process itself. This is particularly relevant in the context of user interfaces, since it means that by changing or adding templates it becomes possible to adapt to new user interface styles or different devices' characteristics.

MODUS TOOL

The MODUS tool supports the described approach. The tool was implemented in Java as an Eclipse plugin [2]. The plugin uses several Eclipse features, as is the case of a WYSIWYG HTML editor and graphical editors for *ecore*. As a plugin, users can integrate MODUS in their work flow, avoiding the need to learn a completely new tool. Once the plugin is installed, the context menu for *ecore* files will provide the option to create a new user interface with MODUS (see Figure 3).

Starting the process

The first step of the approach is to provide a class diagram to the generation process. In practice, the diagram is uploaded in the IDE, and then the context menu used to start the plugin on that diagram. Before proceeding the tool first validates the file and highlights any errors it finds. MODUS then opens the main user interface (see Figure 4) where it is possible to specify the application domain, and the user interface specifications. At each step, help is provided.

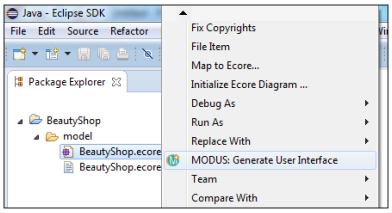

Figure 3. The context menu

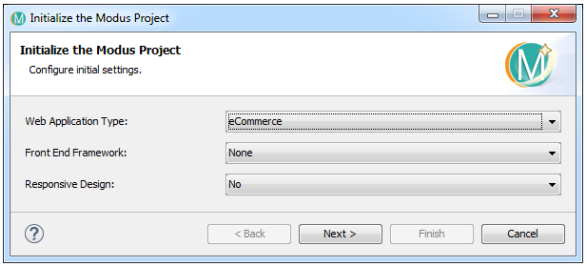

Figure 4. MODUS main window

Mapping Standard Classes to the model

Figure 5 presents the interface which shows the mapping between standard classes in the domain and classes from the model. The interface shows the result of the matching process, which can be modified by the user. The *attributes* buttons open similar windows where each entity's attributes are presented.

Content and Navigation Map

After the mapping is concluded, the content and navigation map for the domain is used (c.f. Figure 2). The user might edit the model, using an available Statecharts editor, or upload a new model.

Managing Entities representation Modes and Layouts

Figure 6 presents the interface for managing the entities representations. Entities are selected through the *drop down* at the top (*Select an Entity*). Once an entity is selected, on the left side (*Display Mode List*) the presentation modes available for that entity are presented. Selecting a mode, shows its details on the right side pane. Similarly to presentation modes layouts can also be edited.

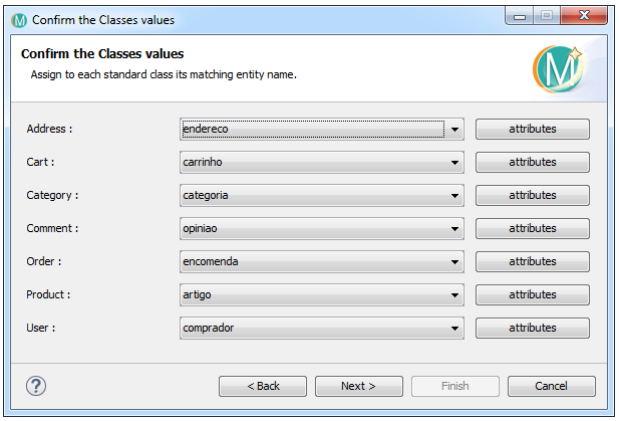

Figure 5. Managing standard classes' associations

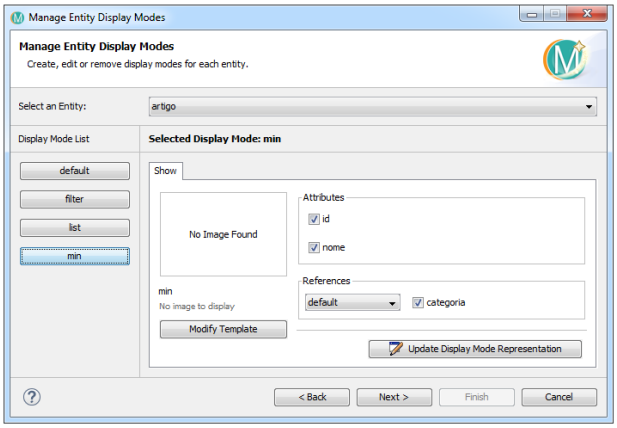

Figure 6. Managing entities display modes

Interface generation

Once the previous steps are concluded, the FUI can be generated. The final interface presents relevant entities in the business logic, according to the defined content and navigation map, using the chosen presentation modes and layout. Figure 7 presents a user interface generated by the MODUS plugin using the content and navigation map introduced in Figure 2. The list of `Product` can be seen towards the right in the middle of the page. It is worth noting that this interface depicts a completely automatic generated interface. The objective is to show the viability to create meaningful user interfaces by automated means. With the inclusion of front-end frameworks, such as Bootstrap[3] and Foundation[4], the interface can be further personalized.

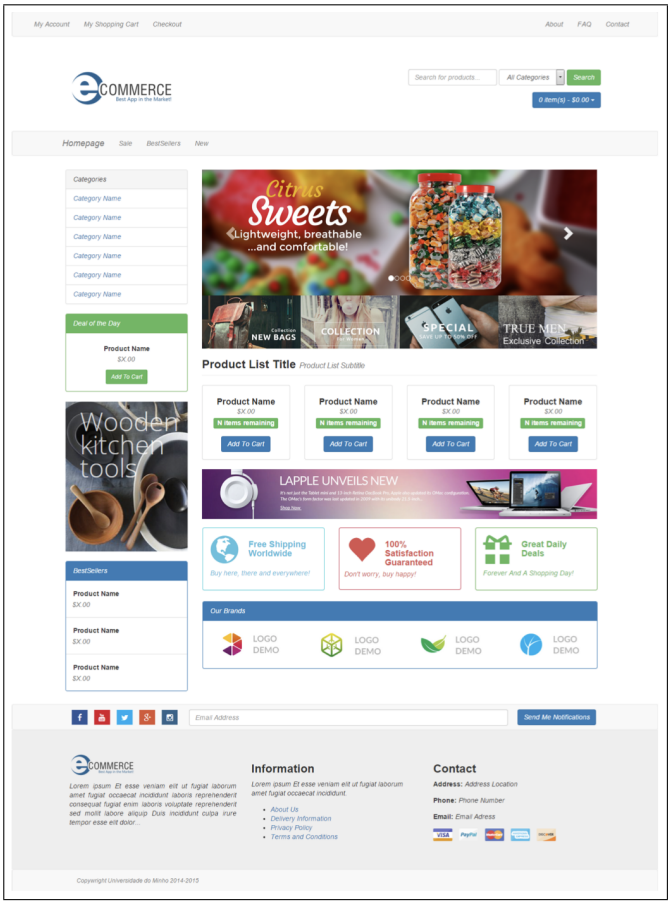

Figure 7. A MODUS generated user interface

CONCLUSIONS AND FUTURE WORK

This papers presented MODUS, a MDE and MBUID based approach to support the generation of user interfaces, with a high automation level. In MODUS, class diagrams and the application domain are the core elements. Class diagram define the entities that are to be presented in the UI. The application domain provides the information to perform several assumptions regarding the UI, avoiding the need to create full UI models. Entities presentation modes and page layouts implement the principle of separating content and style. This way, templates define aesthetics and structural details, supporting more complex and complete UIs.

In order to support the approach, the MODUS Eclipse plugin was developed. The plugin supports each step of the automatic generation process. Through the plugin it was possible to prove the viability of generating the complete user interface for an application, with meaningful content. The support for user customizations has shown to improve the quality of the resulting UI.

In its current instantiation the choice of modeling languages was influenced by, on the one hand, the technology being used to implement the plugin and, on the other hand, the fact that the approaches goal is to support software engineers. This is particularly relevant in the case of the content and navigation map. A Statecharts based approach was chosen both because there were editors available for the language, and because it was a language that software engineers will know. One possible criticism, however, is that this is not a standard use of the language, so some level of training will in any case be needed, and some confusion might arise from the fact that non conventional meaning is being assigned to the language constructs. While a more formal evaluation of the language currently used is still needed, we envisage that languages from the MBUID area (see e.g. [10]) could be adopted with advantage over the current solution.

[3] http://getbootstrap.com
[4] http://foundation.zurb.com

Acknowledgments

This work was financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961.

REFERENCES

1. S. Berti, F. Correani, G. Mori, F. Paternò, and C. Santoro. 2004. TERESA: A Transformation-based Environment for Designing and Developing Multi-device Interfaces. In

CHI '04 Extended Abstracts on Human Factors in Computing Systems (CHI EA '04). ACM, 793–794. DOI: http://dx.doi.org/10.1145/985921.985939

2. F. Budinsky. 2004. *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley.

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

4. R. Couto, A.N. Ribeiro, and J.C. Campos. 2012. MapIt: A Model Based Pattern Recovery Tool. In *Model-Based Methodologies for Pervasive and Embedded Software, 8th International Workshop, MOMPES 2012. Revised Papers*. 19–37. DOI: http://dx.doi.org/10.1007/978-3-642-38209-3_2

5. P.P. da Silva. 2001. User Interface Declarative Models and Development Environments: A Survey. In *7th Intl. Conf. on Design, Specification, and Verification of Interactive Systems (DSV-IS'00)*. Springer, 207–226.

6. P.P. da Silva and N.W. Paton. 2003. User interface modeling in UMLi. *IEEE Software* 20, 4 (July 2003), 62–69. DOI:http://dx.doi.org/10.1109/MS.2003.1207457

7. A.M. Davis. 1992. Operational prototyping: a new development approach. *IEEE Software* 9, 5 (Sept 1992), 70–78. DOI:http://dx.doi.org/10.1109/52.156899

8. Alfonso García Frey, Jean-Sébastien Sottet, and Alain Vagner. 2014. Towards a Multi-stakehoder Engineering Approach with Adaptive Modelling Environments. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '14)*. ACM, 33–38. DOI: http://dx.doi.org/10.1145/2607023.2610273

9. Werner Gaulke and Jürgen Ziegler. 2015. Using Profiled Ontologies to Leverage Model Driven User Interface Generation. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '15)*. ACM, 254–259. DOI: http://dx.doi.org/10.1145/2774225.2775070

10. J. Guerrero-Garcia, J.M. Gonzalez-Calleros, J. Vanderdonckt, and J. Munoz-Arteaga. 2009. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *2009 Latin American Web Congress*. 36–43. DOI: http://dx.doi.org/10.1109/LA-WEB.2009.40

11. B. Hailpern and P. Tarr. 2006. Model-driven Development: The Good, the Bad, and the Ugly. *IBM Syst. J.* 45, 3 (July 2006), 451–461. DOI: http://dx.doi.org/10.1147/sj.453.0451

12. G. Meixner, F. Paternò, and J. Vanderdonckt. 2011. Past, Present, and Future of Model-Based User Interface Development. *i-com* 10, 3 (2011), 2–11. DOI: http://dx.doi.org/10.1524/icom.2011.0026

13. P.J. Molina. 2004. A Review to Model-Based User Interface Development Technology. In *Proc. of the First International Workshop on Making model-based user interface design practical: usable and open methods and tools*.

14. Andreas Pleuss, Stefan Wollny, and Goetz Botterweck. 2013. Model-driven Development and Evolution of Customized User Interfaces. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '13)*. ACM, 13–22. DOI: http://dx.doi.org/10.1145/2494603.2480298

15. Roman Popp, David Raneburger, and Hermann Kaindl. 2013. Tool Support for Automated Multi-device GUI Generation from Discourse-based Communication Models. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '13)*. ACM, 145–150. DOI: http://dx.doi.org/10.1145/2494603.2480334

16. A.R. Puerta, H. Eriksson, J.H. Gennari, and M.A. Musen. 1994. Beyond Data Models for Automated User Interface Generation. In *People and Computers IX, Proceedings of HCI '94*. Cambridge University Press, 353–366.

17. J. Rech and C. Bunse. 2008. *Model-Driven Software Development: Integrating Quality Assurance*. Information Science Reference - Imprint of: IGI Publishing.

18. A.I. Sampaio and J.C. Campos. 2014. Towards a Framework for Adaptive Web Applications. In *HCI International 2014 - Posters' Extended Abstracts, Part I (Communications in Computer and Information Science)*, C. Stephanidis (Ed.), Vol. 434. Springer, 240–245. DOI: http://dx.doi.org/10.1007/978-3-319-07857-1_43

19. E. Schlungbaum. 1996. *Model-based User Interface Software Tools Current state of declarative models*. GVU Center Technical Reports GIT-GVU-96-30. Georgia Institute of Technology.

20. T. Stahl, M. Voelter, and K. Czarnecki. 2006. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

21. P. Szekely, P. Luo, and R. Neches. 1993. Beyond Interface Builders: Model-Based Interface Tools. ACM Press, 383–390.

22. J. Vanderdonckt. 2005. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Advanced Information Systems Engineering: Proc. 17th International Conference, CAiSE 2005*, O. Pastor and J. Falcão e Cunha (Eds.). Springer, 16–31. DOI: http://dx.doi.org/10.1007/11431855_2

23. J. Vanderdonckt and F. Bodart. 1993. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, 424–429. DOI: http://dx.doi.org/10.1145/169059.169340