Evaluating user interface generation approaches: model-based versus model-driven development

Jenny Ruiz de la Peña

jruizp@uho.edu.cu

University of Holguín, XX Anniversary Ave, 80100 Holguín, Cuba

Prof. dr. Estefanía Serral Asensio

estefania.serralasensio@kuleuven.be

Prof. dr. Monique Snoeck

Monique.Snoeck@kuleuven.be

Department of Decision Sciences and Information Management, KU Leuven, Leuven, Belgium

PREPRINT: This paper has been accepted for Software and Systems Modelling

Please cite as follows:

Ruiz, J., Serral, E. & Snoeck, M., Evaluating user interface generation approaches: model-based versus model-driven development, Softw Syst Model (2018). https://doi.org/10.1007/s10270-018-0698-x

Evaluating User Interface Generation Approaches: Model-based vs Model-Driven Development

Jenny Ruiz[1], Estefanía Serral[2], Monique Snoeck[2]

[1]*University of Holguín, XX Anniversary Ave, 80100 Holguín, Cuba*

+53 482672, jruizp@uho.edu.cu

[2]*KU Leuven, Naamsestraat 69, 3000 Leuven, Belgium*

+32 16 32 68 79, {estefania.serralasensio, monique.snoeck}@kuleuven.be

Abstract

Advances in software design possibilities have led to a growing interest in the study of User Interfaces (UIs). Many Model-Based User Interface Development Environments (MB-UIDEs) have been proposed to deal with the generation of the UIs (semi) automatically by using models with different levels of abstraction. Often, this generation is limited to the UI-part of an application. However, achieving true model-driven development (MDD) requires the co-development of application and user interface and hence needs to go a step further. This paper analyzes a large set of existing MB-UIDEs and evaluates, from a critical perspective, to what extent they can be considered full MDD environments and adequately addressing the co-design of UI and application. A robust assessment framework is defined and applied for this purpose. Following the guidelines proposed by Kitchenham & Charters in 2007, we performed a systematic literature review. A total of 82 papers were examined. Based on these papers, an assessment framework containing 10 criteria with specific metrics to evaluate MB-UIDEs was defined and 29 different environments were evaluated following these criteria. The evaluation shows that, although a strong progress has being achieved over the last years, the existing environments do not yet fully exploit the benefits and potentialities of MDD, nor do they adequately integrate UI design with application logic design and generation. Further research needs to be done to support the model-driven development of user interfaces and the co-design of the underlying application. The difficulty of use of the existing MB-UIDEs, the lack of UI design flexibility, and the lack of complete and integrated development support, are the main research gaps that need to be addressed.

Keywords

Model-based User Interface software tools, User Interface generation, Model-driven Development, Integration with Application Development.

1. Introduction

In the last decades, the Information Technology industry has witnessed a rapid growth of platforms, devices, interaction modalities, and environments. This has led to an increase of design possibilities and therefore to a growing interest in the study of User Interfaces (UIs) (Gomaa, Salah, & Rahman, 2005).

The development of the UI of an application represents around 50% of the total application code and development time (Myers & Rosson, 1992), (Jha, 2005), (Kennard & Leaney, 2010). Different kinds of software tools have been created to manage the complexity and reduce the time of UI development. (Myers, 1995) classified these UI tools according to how the UI layout and its dynamic behavior are specified: language-based tools, interactive graphical specification tools, and Model-Based User Interface Development Environments (MB-UIDEs) tools. In language-based tools, developers have to program in a specific language. With interactive graphical specification tools, developers can make an interactive design of the UI. Last, MB-UIDEs aim to generate user interfaces (semi) automatically using models with different levels of abstraction (abstract, concrete, final UI), and provide supporting tools to assist in the modeling task and/or the automatic generation of the UI. MB-UIDEs generate UIs from a set of declarative and high-level models that represent required UI characteristics as collected during the analysis and design phase. The use of high-level models allows an application to be designed by using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain. The use of abstract models facilitates therefore the participation of end-users in the early stages of the development process because models allow end-users to focus on the main concepts (the abstractions) without being confused by many low-level details (Paternò, 2003).

While the benefits of MB-UIDEs are clear, further benefits can be gained by targeting model-*driven* development (MDD), which goes a step further than model-*based* development. The term MDD is used to denote approaches that focus on the creation and exploitation of domain models as prime artefacts to document domain knowledge in an abstract way and use this knowledge (among others) for software creation. The foundational principles of MDD have been explained by many authors. Generally speaking, authors agree on the following basic building blocks: the use of models, meta-models, model-transformations, and platform definitions (Mellor, 2004), (Bézivin, 2004), (Hailpern & Tarr, 2006), (Schmidt, 2006), (Brambilla, Cabot, & Wimmer, 2012), (Kleppe 2003). Model-driven development is further supported by different artefacts: tools, standards, and languages. Some of these artefacts are 'general-purpose MDD tools', such as the tools AndroMDA[1], Acceleo[2], ArcStyler[3]. Other MDD artefacts are specific for UI development, such as the UI definition languages UsiXML, UIML, XIML, or standards such as the International Flow Modeling Language (IFML)[4] adopted by the Object

[1] http://www.andromda.org/
[2] https://eclipse.org/acceleo/downloads/
[3] http://www.arcstyler.com
[4] http://www.ifml.org

Management Group and the Extensible Hypertext Markup Language (XHTML)[5] developed by the WWW.

While model-*based* and model-*driven* UI development environments share a same principle (Aquino, Vanderdonckt, Panach, & Pastor, 2011), namely that one or many models are used to explicitly represent the target UI characteristics and to generate the UI code (Calvary et al., 2003), there is a main difference. In MDD, models are the prime artefacts that drive the development of software based on explicit model transformations. In MB-UIDE however, models are sometimes simply only used for purposes like analysis, early or advanced design, evaluation, etc. and if they are used for code generation, the transformations are not always made explicit. Another shortcoming of a number of MB-UIDEs, is that often the UI model is an isolated artifact used to generate a UI that is not linked with the application logic. This is problematic as, for example, a class diagram will determine what data is available to show in the UI, and how data can be navigated (through defining associations between classes). In MDD, the use of meta-models that define the semantics of the models used allows UI models to be integrated with the rest of the application models and as such generating a UI that is directly integrated with application logic.

The goal of this research is therefore to analyze prevalent MB-UIDEs and answer the following question: to what extent can these MB-UIDEs be considered as 'real' MDD environments and to what extent are they able to integrate UI design and generation directly with the design and generation of the rest of the software application?

To answer this research question, we followed the methodology depicted in Figure 1. We first performed a systematic literature review (SLR) of MB-UIDEs (see Section 2) in order to identify existing surveys on MB-UIDEs and assess to what extent the question can be answered by extracting information from these surveys.

Different surveys were found, but neither on their own, and neither a combination of them provided the required information to answer our research question (see section 2). We proceeded therefore to the elaboration of a new more complete survey. We built an assessment framework to evaluate MB-UIDEs: a) we determined the criteria to be used for the evaluation and b) we defined a metric for each criterion of the framework (see Section 3). Based on the results of the initial and subsequent searches, we also determined the set of MB-UIDEs to be assessed (see Section 4) and we then evaluated these according to the proposed framework (see Section 5). Finally, the analysis of the assessment results led to the identification of important research gaps (see section 6).

[5] https://www.w3.org/TR/xhtml1/

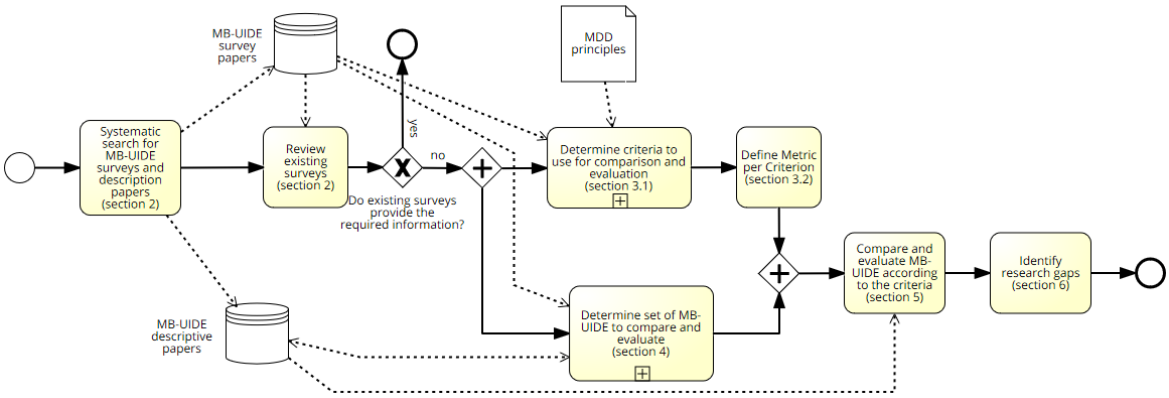

Figure 1 Methodology followed for the comparison and evaluation of MB-UIDEs

It is important to note that this study focuses on model-based approaches that are specifically geared to UI development. Therefore, even though *general purpose* tools, languages, and standards may be used as a component of a MB-UIDE, they are out of scope of this evaluation. For surveys of general-purpose model-driven engineering tools, the reader is referred to (Karanam, 2015) and (Cabot & Teniente, 2006). For surveys of UI definition languages the reader is referred to (Guerrero-García, Gonzalez-Calleros, Vanderdonckt, & Muñoz-Arteaga, 2009; Guerrero-García, González-Calleros, Vanderdonckt, & Muñoz-Arteaga, 2011; Souchon & Vanderdonckt, 2003).

2. Systematic Literature Review on MB-UIDEs

With the purpose of finding the relevant research done on MB-UIDEs, we performed a SLR following the guidelines proposed by (Kitchenham & Charters, 2007). These guidelines propose to perform the SLR in three phases: planning, conducting, and reporting.

Planning the SLR

The main research questions that need to be answered through this search is: Which reviews about UI generation approaches have been performed?

Rather than searching specifically for literature reviews, we searched in general for papers about model-driven development of user interfaces. We performed our search with Scopus and Web of Science databases, which are the largest databases of peer-reviewed literature. Therefore, the search string in Web of Science was: **Theme:** ("model-based user interface" OR "model-driven user interface") *AND* **theme:** (development or design or generation).
The exact search string in Scopus was: (TITLE-ABS-KEY ("model-based user interface" OR "model-driven user interface") AND TITLE-ABS-KEY (design OR development or generation)) AND (EXCLUDE(SUBJAREA,"MEDI OR EXCLUDE SUBJAREA OR EXCLUDE SUBJAREA ") OR EXCLUDE(SUBJAREA," DENT OR EXCLUDE SUBJAREA OR EXCLUDE SUBJAREA ") OR EXCLUDE(SUBJAREA," EART OR EXCLUDE SUBJAREA OR EXCLUDE SUBJAREA ") OR EXCLUDE(SUBJAREA," VETE OR EXCLUDE SUBJAREA ") OR EXCLUDE(SUBJAREA,"MEDI OR EXCLUDE SUBJAREA ") OR EXCLUDE(SUBJAREA," CENG") OR

EXCLUDE(SUBJAREA,"EART") OR EXCLUDE(SUBJAREA,"AGRI") OR EXCLUDE(SUBJAREA,"ARTS") OR EXCLUDE(SUBJAREA,"BIOC") OR EXCLUDE(SUBJAREA,"SOCI") OR EXCLUDE(SUBJAREA,"BUSI") OR EXCLUDE(SUBJAREA,"DECI") OR EXCLUDE (SUBJAREA,"ECON OR EXCLUDE SUBJAREA ") OR EXCLUDE (SUBJAREA," PHAR") OR EXCLUDE(SUBJAREA,"MATE") OR EXCLUDE(SUBJAREA,"ENVI") OR EXCLUDE(SUBJAREA,"ENGI") OR EXCLUDE(SUBJAREA,"MULT OR EXCLUDE SUBJAREA ")).

In order to assess the quality of the query, we checked that the studies we already knew to be relevant for our search (such as (Da Silva, 2001), (Engel, Herdin, & Märtin, 2014)), appeared in the results in order to ensure that this search process was able to find these papers.

Reporting the SLR:

The query resulted in a collection of 241 papers (201 from Scopus and 124 from Web of Science) dating from 1994 to 2016. Most of the papers were in both databases. In anticipation of the fact that existing surveys might be incomplete, and that "regular" papers could have interesting information in their "related work" sections, the inclusion and exclusion criteria should not only allow the identification of existing evaluations of MB-UIDEs but also allow selecting papers for the later extraction of the set of MB-UIDEs to be further analyzed. The inclusion and exclusion criteria were therefore as follows:

- Inclusion criteria:
  - Papers with explicit criteria-based comparison of MB-UIDEs
  - Papers that describe (and possibly also analyze) MB-UIDEs specifically for the design, development or generation of UI.
- Exclusion criteria: The papers that were excluded mostly had a too narrow scope, e.g. only describe a particular aspect of the development. The full list of rejection criteria and number of papers rejected according to those criteria is provided in Appendix 1.

Based on these criteria, we retained 55 papers and discarded 186 papers. Next, 27 additional papers were found through snowballing the references of all 55 papers directly obtained from the SLR, finally leading to a total of 82 papers.

Within the complete set of papers, we searched for surveys on MB-UIDE. We found the following 7 surveys summarized in : 4 as direct result of the query ((Akiki, Bandara, & Yu, 2015), (Engel et al., 2014), (Griffiths et al., 2001) and (Da Silva, 2001)); and 3 as a result of the snowballing step ((Gomaa et al., 2005), (Schlungbaum, 1996) and (Vi Tran, Vanderdonckt, Kolp, & Wautelet, 2010)). Those papers describe the most prominent MB-UIDE approaches.

Table 1 Selected papers about Model Based User Interface Development Environments

Reference	Summary
(Schlungbaum, 1996)	Characterizes the criteria that an environment should have to be classified as model-based, gives a definition for declarative models, and analyses their use by model-based approaches.
(Griffiths et al., 2001)	Presents a survey of MB-UIDEs and the main features to be classified as interface development environment: support for the automatic generation of interfaces, use of declarative methods and models for specifying interfaces and a methodology to support the development of the interface.
(Da Silva, 2001)	Gives a survey of MB-UIDEs and their use of declarative models. It describes the User Interface process design in a MB-UIDE, and presents the models, their semantics and notations, and the tools that are used by each MB-UIDE.
(Gomaa et al., 2005)	Presents a survey of MB-UIDEs with a short description of each approach. This survey also shows which models are used by the MB-UIDEs and describes some limitations of the approaches.
(Vi Tran et al., 2010)	Presents an analysis of the models used by the MB-UIDEs as well as a description of their tools and the languages used to describe the generated UI.
(Engel et al., 2014)	Presents a brief description of some MB-UIDEs and an analysis of the models they use. This survey also shows the coverage of the Cameleon Reference Framework (CRF) abstraction levels and the notations used by each of the approaches.
(Akiki et al., 2015)	Focuses on adaptive UIs and presents an overview and an analysis of the strengths and shortcomings of model-driven UI architectures, techniques, and tools.

Table 2 shows which MB-UIDEs were analyzed in which survey. As can be seen in this table, older evaluations are obviously missing more recent approaches. But even (Engel et al., 2014) and (Akiki et al., 2015), which are the most recent evaluations, are nevertheless incomplete in terms of approaches covered: they do not cover older evaluations and -as will be explained in section 4- in the total set of papers we found another 11 (recent) approaches that are missing from their analysis as well.

Table 2 MB-UIDEs and authors who analyzed them

Approach	(Schlungbaum, 1996)	(Griffiths et al., 2001)	(Da Silva, 2001)	(Gomaa et al., 2005)	(Vi Tran et al., 2010)	(Engel et al., 2014)	(Akiki et al., 2015)
ITS		X	X	X		X	
HUMANOID		X	X	X			
UIDE	X	X	X	X			
AME	X	X	X	X		X	
ADEPT	X	X	X	X			
GENIUS	X	X	X	X	X		
TRIDENT	X	X	X	X	X		
JANUS	X	X	X	X	X		
MECANO	X	X	X	X	X	X	
TADEUS	X	X	X	X	X		
MASTERMIND	X	X	X	X			
FUSE	X	X	X	X	X		
TEALLACH		X	X	X	X		
UI-TERESA				X		X	
SUPPLE						X	X
GOLIATH					X		
MARIA						X	X
DRIVE				X			
Dygimes				X			
CTTE							X
Cedar							X
Damask							X
GrafiXML							X
GUMMY							X
Ideal XML							X
Leonardi							X
MASP							X
SketchiXML							X
UsiComp							X

Looking at criteria used for evaluation, the different papers use different sets of criteria, which makes it difficult to obtain a global picture, even by combining the results from two or more papers. Some of the older works use criteria that are interesting from the perspective of evaluating the model-driven character of the approach, but recent works do not use these criteria given the different focus of the survey. For example, (Akiki et al., 2015) does not use *Models used*, *Language used for generated UI*, or *Generation of application code* (see section 3 for more details). This survey rather focuses on adaptive UIs, while our research targets a more general goal.

Finally, metrics for the criteria are often missing. For example, (Akiki et al., 2015) explains the criteria but does not propose metrics for the criteria. The evaluation indicates a level of satisfaction (full, half or empty circle) but without explaining when a criterion is completely, or only partially fulfilled. Also this makes it hard to derive a clear picture from the existing surveys.

From this overview it is clear that none of the existing criteria-based comparisons, nor a combination of them, provides an answer to the question to what extent existing MB-UIDEs are model-driven or not and to what extent they address the integration with application logic development.
A new comparison is therefore needed, which will be presented in the rest of this paper. In summary, compared to the existing surveys, the evaluation in section 5 will present the full collection of approaches (see section 4), including 11 recent approaches that have not been analyzed before, and the comparison will be based on a uniform set of criteria with a clear metric for each criterion (see section 3).

3. Framework to assess MB-UIDEs

In this section we build and motivate an evaluation framework to assess MB-UIDEs. First, in subsection 3.1 we identify the main criteria to be included in the framework. Next, subsection 3.2 details how criteria satisfaction will be measured.

3.1 Framework Criteria

As we are targeting the assessment of the extent to which a MB-UIDE can be considered a full MDD environment, the specific characteristics of MDD are an important source to determine the relevant evaluation criteria. However, starting from the general MDD principles holds the risk to end up with too general criteria, not specifically geared towards UI design and development. We therefore decided to collect first the set of criteria proposed in previous research (see step 1 in Figure 2). This yielded 20 criteria that were then critically evaluated for overlap and subsumption, and for their relevance in a MDD context (see step 2 in Figure 2). Nine criteria were retained, and 11 criteria were removed. Finally, in the third step, we performed a completeness check against the major MDD principles to check the coverage of the list of retained criteria. In this third step, 1 new criterion was added, resulting in a total set of 10 criteria in the framework.

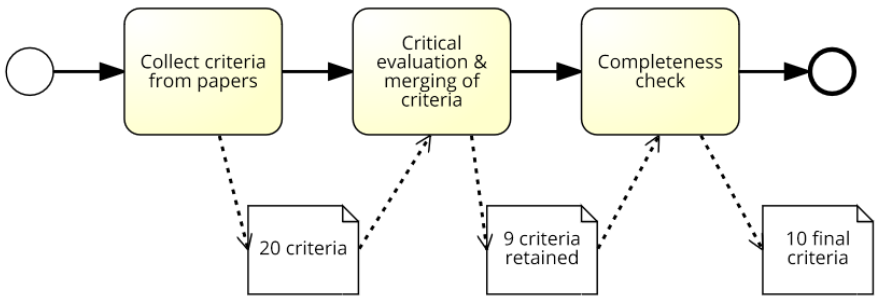

Figure 2 Methodology to determine criteria to use for MB-UIDE evaluation

We collected the criteria by analysing the papers obtained from the SLR that give an explicit criteria-based comparison of MB-UIDEs. Authors of previous surveys propose criteria for evaluating MB-UIDEs from the perspective of the richness of the models used

by the tools (Da Silva, 2001), the coverage of different levels of abstraction by the approaches (Engel et al., 2014), the extent and quality of support the tools offer to the UI developers for the creation and manipulation of models, and the generation of UI code (Griffiths et al., 2001), (Akiki et al., 2015). Only the criteria that satisfy the following constraints were included:

- having been referenced as an important feature for MB-UIDEs by relevant authors in the field,
- being relevant for the purpose of determining to what extent the MB-UIDEs adhere to major principles of MDD.

Below we present the 20 criteria proposed in these **surveys**. Some of the criteria have been renamed or slightly modified to better represent what is actually evaluated. The first three of the retained criteria are of a descriptive nature (meaning that the rating can only be done by means of a factual description), whereas for the remaining six criteria the support can be rated on an ordinal scale. The metrics for those criteria will be presented in subsection 3.2.

Descriptive criteria retained for the comparison of the approaches:

1. **Models used:** proposed by (Schlungbaum, 1996), (Da Silva, 2001), (Griffiths et al., 2001), (Gomaa et al., 2005), (Vi Tran et al., 2010), and (Engel et al., 2014). This criterion describes which models are used by the different model-based approaches. As the importance of this aspect is to know which models are used by the approaches in a descriptive way, we evaluate the approaches in a purely descriptive way along this criterion, without rating it against a predefined norm.
2. **Language used for generated UI:** proposed by (Da Silva, 2001), (Vi Tran et al., 2010), (Engel et al., 2014). This criterion describes for which languages the approach generates the UI. It will be divided in language used for expressing a UI design (= input for the code generation) and target language used for the generated UI code (= output of the code generation). This criterion is useful to select the most appropriate approach for a specific case.
3. **Tool support:** proposed by (Da Silva, 2001), (Griffiths et al., 2001), (Vi Tran et al., 2010), (Engel et al., 2014), (Akiki et al., 2015). This criterion evaluates the tools used by the approaches for the creation, manipulation and transformation of models, and the generation of UIs.

Quantitative criteria retained for the evaluation of the approaches:

4. **Threshold and ceiling:** proposed by (Akiki et al., 2015). This criterion describes to what extent an approach is easy to use and how advanced the outcome of the tool can be. An ideal tool should be easy to learn and use. We propose to keep this criterion for the framework and name it **Ease of use.**
5. **Generation of application code:** proposed by (Da Silva, 2001), (Griffiths et al., 2001), (Vi Tran et al., 2010). This criterion evaluates to what extent the approach is able to generate application code. As it is not only necessary to create and manipulate the models, but also to obtain code that can be used in an application we keep this criterion for the framework.
6. **Completeness:** proposed by (Engel et al., 2014), (Engel et al., 2014). This criterion is related to the types of UIs (e.g. WIMP, tangible, etc.) that can be produced using an approach. A system with higher completeness can be used for developing a wider variety of software applications, a need that UI designers have today. We

propose to keep this criterion but reformulate it as **Code generation extensibility**. The completeness describes the current possibilities the approach has. However, our proposed reformulated criterion not only addresses the current possibilities but also those that the approach can have in a near future if extended. The evolution from model-based to model-driven development implies the use of explicitly defined model-to-model and model-to-code transformations that are compliant with an explicitly defined meta-model of the approach. The use of explicit transformations enables the possibility to generate code for different target platforms. This criterion will allow evaluating to what extent MD-UIDEs have evolved to MDD.

7. **User feedback on the adapted UI:** proposed by (Akiki et al., 2015). This criterion evaluates whether the approach keeps the end-users in the loop of the adaptation process. Due to the variety of types of users that we witness today, and the necessity of creating UIs able to accommodate for all these types of users, we keep this criterion for the framework. We proposed to rename it as **User adaptiveness,** to clarify that it evaluates to what extent an environment is taking into account the user characteristics and preferences.
8. **Coverage of the UI development levels as proposed by the CRF:** proposed by (Da Silva, 2001), (Gomaa et al., 2005), (Engel et al., 2014), (Akiki et al., 2015). Explains if the approach provides the UI from the most abstract way to the most concrete one. As previously explained, today there is a variety of contexts of use and hence a need to develop UIs that can be used in all these contexts. Because the CRF offers a solution for this issue because of its ability of capturing the UI at different levels of abstraction, we keep this criterion for the framework.
9. **Control over the UI:** proposed by (Akiki et al., 2015). This criterion relates to the level of details that the designer can manipulate and the fact that it is important to allow the designer providing different versions of the UI. We keep this criterion for the framework but we propose to rename it as **UI design flexibility.**

A number of criteria used in previous research was not retained for the framework for being subsumed by one of the retained criteria or not being relevant for the analysis of the adherence of the environments to MDD. Below we list those criteria and motivate for each of them why we did not retain it:

10. **Empowering new design participants:** proposed by (Akiki et al., 2015). This criterion evaluates the possibility of incorporating people to the design process like non-developers such as end-users, IT personnel, etc. This criterion will not be retained in the framework. As previously explained we propose to use the criterion **User adaptiveness** to incorporate the ability of the approach to incorporate in the generated UI the user preferences and characteristics, rather than involving them to the development process.
11. **Possibility of domain model for integration with Computer Aided Software Engineering (CASE) tool:** proposed by (Schlungbaum, 1996). This criterion evaluates whether the domain model can be integrated with existing CASE tools. As the approaches should have their own tools to create and manipulate models, (not only the domain model), and we already have the criterion **Tool support**, we propose not to keep this criterion for the framework.

12. **Inter model relationship:** proposed by (Da Silva, 2001), (Engel et al., 2014)[6]. This criterion evaluates which constructs from different models are related to each other. Although we believe that once an approach is selected this is an important criterion to understand how the approach is defined, the level of details of this criterion is out of scope for the comparison of approaches.
13. **Extensibility:** proposed by (Akiki et al., 2015). This criterion evaluates to what extent a UI adaptation approach includes a variety of adaptation types such as feature reduction, navigation, help, etc. At the same time, it assesses the capability of the approach to add new adaptive behavior at runtime. This criterion was proposed specifically for analyzing the way adaptive behavior is developed, and therefore it is not aligned to the purpose of evaluating the adherence to MDD principles. However, part of the criterion that is relevant is included in the criterion **User adaptiveness** (number 7 above).
14. **Visual and code-based adaptive behavior:** proposed by (Akiki et al., 2015). Describes the visual and code-based representation for the developers to implement adaptive behavior. As this criterion (like the previous one) was proposed specifically for analyzing the way adaptive behavior is developed, we propose to not use it in the framework.
15. **Modeling, generation and synchronization:** proposed by (Akiki et al., 2015). This criterion evaluates the ability of the tools to create and manipulate the models with different levels of abstraction in an easy way. This criterion is subsumed in **Coverage of the levels of abstraction**, the **Ease of use** and the **Tool support** criteria already considered in the framework.
16. **Preserving designer input on the UI:** proposed by (Akiki et al., 2015). This criterion evaluates if the approach is able to maintain the UI designer decisions without overriding the input made by the designer. Fulfilling the criterion **UI design flexibility** should satisfy the ability of taking into account the designer decisions and therefore this criterion is not retained.
17. **Reducing solution viscosity:** proposed by (Akiki et al., 2015). This criterion assesses the reduction of the effort required to iterate on the possible solutions so as to be able to make rapid design changes and have users testing the adaptive behavior. This criterion also evaluates if the designer can accomplish more by expressing less and how close the means for expressing design choices are related to the problem being solved. We propose to not use this criterion in the framework as it evaluates the choices for realizing adaptive behavior.
18. **Functionality:** proposed by (Engel et al., 2014). This criterion assesses if the approach provides support for UI modeling and UI generation. We propose not to retain this criterion in the framework as we already have criteria that assess the **Tool support** and the **Generation of application code** (not only UI code) of the approach.
19. **Availability:** proposed by (Engel et al., 2014). This criterion describes whether the tools of the approach can be found. In our evaluation we already indicate the tool/s

[6] Proposed as Model notation criterion by these authors

provided for each approach. We consider the readily availability of the tools not a robust criterion because it can change in the future.

20. **Application examples:** proposed by (Engel et al., 2014). This criterion just mentions where the approach has been used. We proposed to discard this criterion for the comparison of approaches.

When modelling the different aspects of an application-to-be, different modelling languages can be used for different aspects. Nevertheless, MDD assumes an encompassing meta-model or model bridges such as to be able to integrate different models into an encompassing view. This is required in order to generate integrated code that covers all aspects of the application.

For this reason, we propose to add the following criterion: **Integration in the application development cycle.** This criterion is aligned with the principle of MDD approaches of driving the complete development process with models as opposed to having separate, non-integrated approaches for the development of UIs on the one hand, and the development of the applications functionality on the other hand. This criterion thus evaluates to what extent MD-UIDEs consider the UI as part of the whole application and not as an isolated aspect.

3.2 Metrics to Measure the MB-UIDE Assessment Framework

This section presents the metrics, i.e. the assessment scales, for each of the framework criteria. First we will explain the rating scale for the three criteria that allow making comparison between the different analyzed approach in a descriptive way: Models used, Language used for the generated UI and Tool support.

Models used: According to (Gomaa et al., 2005), the following kinds of declarative models can be distinguished in the existing MB-UIDE approaches:

- *Task model*. Describes the tasks to be accomplished by the users when using the application. It can include sub-tasks, their goals, and how the goals can be achieved.
- *Application or domain model*. Specifies a description of the objects that can be manipulated, accessed or visualized by the users. This information is independent of the way objects are displayed, or how the operations are invoked.
- *User model.* Describes the abilities, characteristics and knowledge of users of the applications. It also models the User Interface preferences of individual users or user groups.
- *Presentation model.* Describes the static representation of the User Interface. This model exists at two different levels of abstraction: abstract and concrete. The first one is an abstract view of an interface, independent from platform, language or interaction model. The second one is a concrete interface that can be presented to the user.
- *Dialogue model*. Holds the conversational (human-computer dialog) aspect of the UI.

While analyzing the surveyed approaches we found that there are other models used by only some approaches. Those models are the context, interaction, device, platform,

transformation, mapping, data service, service, layouting, event, view, data and validation and workflow models.
User, presentation and dialogue models are specific for the development of the UI. Application or domain models are obviously the main sources of integration of UI generation with application generation. Task models are specific for UI development as well, but at the same time they provide the possibility of integration with the concept of activity or task of business process models, thus facilitating the integration of UI design for process aware information systems.
The mentioned models used by the different approaches will be shown in a table where:

●: Indicates that the model is used
○: Indicates that the model is not used

Evaluation along the criteria **Language used for generated UI** and **Tool support** will be presented in a table showing for which languages the approaches can generate the UIs and with which tools.
The remaining seven criteria will be assessed on a scale with three levels. The following clarifies the rating scale for each of these criteria.

- **Ease of use**: According to (Myers, 1995), MB-UIDEs have not been well spread because their use implies that programmers must learn new languages to define and describe the models. This can be avoided by offering easy to use MB-UIDEs. Ease of use means that the environment should be clear and understandable, flexible to interact with, and easy to become skillful at. This can only be measured in an indirect way. Due to the complexity of the criterion, in order to properly evaluate it, we have subdivided it into the following sub criteria:
  - Needed tools and their interoperability. (Myers, 1993) explains that the tools to help with UI are extremely complex. So, the more tools are needed by an approach, the more difficult it is to use them:
    ●: One single tool for all the phases of the approach
    ◖: Different tools that are interoperable: different tools are needed but it is possible to import models from one tool to another.
    ○: It is necessary to use different tools that are not interoperable
  - Amount of documentation. If developing the UI is a difficult process another way to make it easier is by having documentation:
    ●: Documentation for all the covered phases and tools
    ◖: Incomplete documentation; e.g., documentation provided only for some phases or tools
    ○: No documentation available
  - Clarity of documentation. It is not enough to have sufficient documentation; it is also important that this documentation is clear:
    ●: Clear and understandable documentation
    ◖: Documentation requires some effort to understand
    ○: Unclear and hard to understand documentation
  - Global evaluation. This criterion aggregates the previous three sub criteria into one value:
    ●: Either all the sub criteria are evaluated as ●, or two of the three as ● and one as ◖

◖: All the other combinations of evaluation of the three sub criteria
○: At least two out of the three sub criteria are evaluated as ○

- **Generation of code** (Vi Tran et al., 2010): this criterion assesses application code generation. To produce an accurate application at low cost, developers expect that both the UI and the application code are automatically generated:
  ●: Fully functional application generated
  ◖: UI code only or with generic application functions generated
  ○: No code generated
- **Code generation extensibility**: This criterion refers to the facilities that the environment provides to add transformations in order to generate code for different target languages and platforms:
  ●: Transformations to generate code for different languages and platforms can be added and the approach already provides transformations for different languages and platforms.
  ◖: Transformations for code generation for different languages can be added but currently the approach only generates the code for one language and platform.
  ○: Transformations for code generation for different languages and platforms cannot be adapted.
- **User adaptiveness** (Seffah, Gulliksen, & Desmarais, 2005): This criterion evaluates the consideration of characteristics and skills of the target users in order to obtain better acceptance of the application:
  ●: Considered
  ◖: Considered but without any example that makes use of it.
  ○: Not considered
- **Coverage of the UI development levels:** The environment must offer support for the four UI development levels corresponding to the four levels of abstraction of the CRF:
  - Task and concepts: This level considers the logical activities that need to be performed in order to reach the end user´s goals; and the domain objects manipulated by these tasks.
  - Abstract User Interface (AUI): This level represents the UI in terms of interaction spaces (or presentation units), independently of which interactors are available and even independently of the modality of interaction (e.g., graphical, vocal, haptic).
  - Concrete User Interface (CUI): This level represents the UI in terms of "concrete interactors", which depend on the type of platform and media available and which have a number of attributes that more concretely define how the UI should be perceived by the end user.
  - Final User Interface (FUI): This level consists of source code, in any programming or markup language (e.g., Java, HTML5, VoiceXML, X+V) (Aquino et al., 2011).

  The score for this criterion is defined as follows.
  ●: Support for all the UI development levels

◖: Support for two or three UI development levels.
○: Support for only one or none UI development levels

- **UI design flexibility**: This criterion evaluates whether the approach offers different UI designs options for the design. According to (Aquino et al., 2011) this allow making the UI customizable and reusable:
  ●: UI design can be customized
  ◖: UI design options can be partially customized or with some effort.
  ○: UI design cannot be customized
- **Integration of UI development in the application development cycle**: The environment needs to consider the UI as part of the whole application development process (requirement elicitation, analysis, design, implementation and testing phases), i.e., the UI development should not be done in an isolated way, but maintaining and making explicit the link with the rest of the application (Gomaa et al., 2005). Note that implementation is not always done through code generation; therefore, this is tested in a separated criterion (see above):
  ●: The development of the UI is integrated within the application development.
  ◖: The development of the UI is partially integrated within the application development (integrated with some phases of the development cycle, but not with all).
  ○: The UI development is not integrated in the application development cycle, but done in an isolated way.

4. Model-Based Approaches for User Interface Generation

The 7 previous surveys together present an evaluation of 29 different environments (see Table 2). However by analyzing the total set of papers found through SLR and snowballing contains 40 different MB-UIDEs. Table 3 presents the complete set of MB-UIDEs and where we found them (in a survey paper, in a descriptive paper, or as the result of snowballing). Note that some approaches analyzed by previous surveys were not found in the descriptive papers. This is because these approaches are relatively old or because they are described not specifically as Model-Based or Model-Driven User Interface approaches in general, but created for a narrower scope. Examples are: DRIVE, which produces interfaces to databases rather than applications in general; SUPPLE, which is positioned as an automated design tool; SketchiXML, which is created for interface sketching; and FlowiXML, which is used for designing workflow management systems.

Table 3 MB-UIDEs and where they were found

Approach	Survey papers (29)	Descriptive papers (23)	Snowballing (17)
ITS	X		
HUMANOID	X		X
UIDE	X	X	X
AME	X		X
ADEPT	X		X
GENIUS	X		
TRIDENT	X		X
JANUS	X		X
MECANO	X	X	X
TADEUS	X	X	
MASTERMIND	X		
FUSE	X		X
TEALLACH	X	X	X
UI-TERESA	X	X	X
SUPPLE	X		
GOLIATH	X	X	
MARIA	X	X	X
DRIVE	X		
Dygimes	X	X	
CTTE	X	X	
Cedar	X	X	
Damask	X	X	
GrafiXML	X	X	
GUMMY	X		
Ideal XML	X	X	
Leonardi	X		
MASP	X	X	X
SketchiXML	X		
UsiComp	X	X	
OO-Method		X	X
Just-UI		X	X
UsiXML		X	
CoGenIVE		X	
MANTRA		X	X
WAINE		X	
XMobile		X	
CIAT-GUI		X	
LIZARD		X	
FlowiXML			X
DB-USE			X

The final list of MB-UIDEs has been compiled from the previous sources taking into account the following inclusion and exclusion criteria.

Inclusion criteria:

- The approaches must target specifically the development of UI or be relevant for this context.
- Recent approaches not included in the analyzed surveys that meet the other criteria.

Exclusion criteria:

- Tools used to edit models.
- Approaches without enough documentation.
- Approaches that evolved into another.
- Approaches in an early stage of development.

We do not consider in our evaluation the MB-UIDEs for which the documentation is not detailed enough to be analyzed or that represent an older version of a new one already considered in our study. Specifically, we omitted AME (Märtin, 1996), ITS (Wiecha, Bennett, Boies, Gould, & Greene, 1990), DRIVE (Mitchell & Kennedy, 1996) and Leonardi[7], because they do not provide enough information for being analyzed. We also omitted HUMANOID (Luo, Szekely, & Neches, 1993) and UIDE (Sukaviriya, Foley, & Griffith, 1993), which have become MASTERMIND (Szekely, Sukaviriya, Castells, Muthukumarasamy, & Salcher, 1995), and which has been included instead. We omitted CTTE (Mori, Paternò, & Santoro, 2002) because it is only for developing and analyzing task models. We omitted Damask (Lin & Landay, 2002) because rather than generating UIs from models, is targeted towards prototyping UIs.

As the result of the step *Extract MB-UIDE from descriptive papers* (see Figure 3), our evaluation includes the following additional MB-UIDEs not included yet in previous analyses: Just-UI (P. J. Molina, Meliá, & Pastor, 2002), OO-Method (J. C. Molina & Pastor, 2004), CoGenIVE (Cuppens, Raymaekers, & Coninx, 2005), MANTRA (Botterweck, 2007), WAINE (Delgado, Estepa, & Estepa, 2007), XMobile (Viana & Andrade, 2008), CIAT-GUI (A. I. Molina et al., 2012) and LIZARD (Marin, Ortin, Pedrosa, & Rodriguez, 2015). Instead of describing some of the tools (GrafiXML, IdealXML, SketchiXML) presented by previous authors, we decided to describe UsiXML, which is not only a UIDL, but also a conceptual methodology that encompasses the use of these tools (Vanderdonckt, 2008).

[7] http://www.leonardi-free.org and http://www.w4.eu

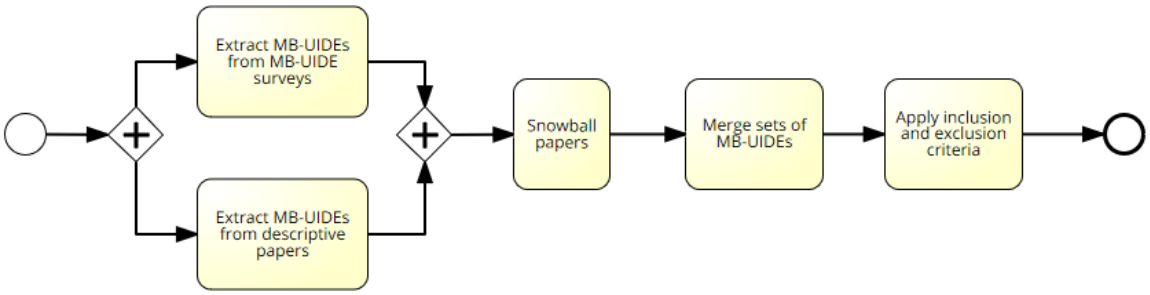

Figure 3 Methodology to determine the set of MB-UIDEs to be assessed

As a result of the step *Snowball papers* (27 papers), our evaluation includes the following additional MB-UIDEs not included yet either in previous analysis or in the SLR: FlowiXML (Guerrero, Vanderdonckt, & Gonzalez, 2008) and DB-USE (Vi Tran, Vanderdonckt, et al., 2010).

5. Evaluation of MB-UIDEs

By applying the presented assessment framework we evaluated to what extent each approach exploits the advantages of MDD. Ideally, such evaluation should have been performed through a controlled case-based lab test with each of the tools. Specially, for a few criteria such as User adaptiveness, UI design flexibility, or Ease of use, which depend on the opinion of specific users. However, for some of the MB-UIDEs the tools are not (freely) available (any more), which makes it impossible to subject them to a lab-test. In other cases, we faced pernicious installation problems. These issues limited the possibility of performing a case-based lab test with all the analyzed approaches. Therefore, to ensure an equal test for all approaches, we decided to perform a documentation-based assessment of all the selected MB-UIDEs based on the information about the features and utilization of the approaches that is available. We gathered, read and analyzed as much literature and documentation as possible on each MB-UIDE. In addition, to avoid a bias in the evaluation along the criteria that can depend on users' opinions, we tried to make the metrics as objective as possible. For instance, for the Ease of use, the two first criteria (*Needed tools and their interoperability* and *Amount of documentation*) are totally objective, and only the third criterion (*Clarity of documentation*) may be considered subjective.
In the following, we explain in detail the evaluation of the selected MB-UIDE approaches.

An overview of the models used by the studied approaches is provided in Table 4. As it can be observed in the table, the models that are used the most in automatic UI generation are domain and task models. Almost all MB-UIDEs use either one or the other, and many MB-UIDEs use both like TRIDENT (Bodart & Vanderdonckt, 1995), TADEUS (Stary, 2000), MASTERMIND (Szekely et al., 1995), FUSE (Lonczewski & Schreiber, 1996), TEALLACH (Griffiths et al., 2001), UI TERESA (Berti, Correani, Paterno, & Santoro, 2004), UsiXML (Limbourg, Vanderdonckt, Michotte, Bouillon, & Florins, 2004), FlowiXML (Guerrero et al., 2008), MASP (Feuerstack, Blumendorf, Schwartze, & Albayrak, 2008), DB-USE (Vi Tran et al., 2010), CIAT-GUI (A. I. Molina et al., 2012) and

LIZARD (Marin et al., 2015). Presentation, dialog and user models are not so commonly used (used by 10, 9 and 7 approaches respectively). The context, interaction, device, platform and transformation models are each used by 2 approaches. The least used models are the mapping model (only used by UsiXML), data service model (only used by LIZARD), service and layouting models (only used by MASP (Blumendorf, Lehmann, Feuerstack, & Albayrak, 2008)), event model (only used by MARIA) (Paterno, Santoro, & Spano, 2009), view model (only used by LIZARD), data and validation model (only used by XMobile) and workflow model (only used by FlowiXML (Guerrero et al., 2008)). Two special cases are MANTRA (Botterweck, 2007) and GUMMY (Meskens, Vermeulen, Luyten, & Coninx, 2008). MANTRA starts from an AUI model without using any of the mentioned models. However, as (Botterweck, 2007) explains, it is possible to start from a task model and transfer the task model into the AUI model. Unlike other approaches, GUMMY aims to generate a Concrete User Interface from a set of existing UI that are all created for the same application, without using the analyzed models in the previous table.

Table 5 shows the languages used by all the analyzed approaches for the generated UIs and the supporting tools.

The languages presented in the first column of Table 5 are used (as input languages) for designing the UI. In some cases, domain-specific languages have been defined for storing the resulting UIs in MB-UIDE. For example, MECANO uses MIMIC for expressing both generic and application-specific interface models (Puerta, 1996). JANUS uses the JANUS Definition Language (that is an extended AM CORBA IDL and ODMG ODL) (Da Silva, 2001). TERESA, SUPPLE, MARIA and UsiXML have their own languages: TERESA XML (Mori, Paterno, & Santoro, 2004), SUPPLE´s functional specification language (Gajos & Weld, 2004), MARIA XML (Paterno et al., 2009) and UsiXML (Limbourg, Vanderdonckt, Michotte, Bouillon, & López-Jaquero, 2005) respectively. The second column presents the languages used for the code generation (the output of the approaches expressed in these languages) by the MB-UIDE´s tools. C++ is an example of a language used for code generation by approaches like JANUS, MASTERMIND, FUSE and CoGenIVE. Other approaches use a textual description for a UIMS or generate UI code in programming languages.
Regarding the tools used by MB-UIDEs, we note that most of them are research prototypes, except for OO-Method, which is supported by the Oliva Nova[8] tool and is used by the company *Integranova* to commercially develop real-life software applications.
Table 6 analyzes first the *Ease of use* criterion for the presented approaches according to the explained metrics.

[8] Olivanova software: http://software.olivanova.com/uk/

Table 4 Models used by the studied approaches

Mb-UIDE / Models	Domain	Task	Presentation	Dialog	User	Context	Interaction	Device	Platform	Transformation	Mapping	Data service	Service	Layouting	Event	View	Data and validation	Workflow
ADEPT (1992)	○	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○
GENIUS (1993)	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○
TRIDENT (1993)	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
JANUS (1993)	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
MECANO (1994)	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
TADEUS (1994)	●	●	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○
MASTERMIND (1995)	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
FUSE (1996)	●	●	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○
TEALLACH (1999)	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
OO-Method (2001)	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Just-UI (2002)	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
UI TERESA (2003)	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Dygimes (2003)	○	●	●	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○
SUPPLE (2004)	●	○	○	○	●	○	○	●	○	○	○	○	○	○	○	○	○	○
GOLIATH (2004)	●	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○
UsiXML (2004)	●	●	○	○	○	●	○	○	○	●	●	○	○	○	○	○	○	○
CoGenIVE (2005)	○	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○
MANTRA (2006)	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
WAINE (2007)	●	○	●	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○
XMobile (2008)	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○
GUMMY (2008)	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
FlowiXML (2008)	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●
MASP (2008)	●	●	○	○	○	●	●	○	○	○	○	○	●	●	○	○	○	○
MARIA (2009)	○	●	○	●	○	○	○	○	○	●	○	○	○	○	●	○	○	○
DB-USE (2010)	●	●	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○
CIAT-GUI (2012)	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
UsiComp (2012)	○	●	○	○	●	○	○	○	●	○	○	○	○	○	○	○	○	○
Cedar (2013)	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
LIZARD (2015)	●	●	○	○	○	○	○	●	●	○	○	●	○	○	○	●	○	○
	20	18	10	9	7	2	2	2	2	2	1	1	1	1	1	1	1	1

Table 5 Languages used by MB-UIDEs for the generated UI and supporting tools

MB-UIDE	Language for the UI at design level	Language for the UI code generation	Tool
ADEPT	Communication Sequential Process	Textual description for UIMS	Interface builder
GENIUS	Petri-net based dialogue description	Textual description	ER diagram editor
TRIDENT	Activity Chaining Graph	Textual description	SEGUIA, SIERRA
JANUS	Janus Definition Language	C++	OOA-Tool
MECANO	MIMIC	-	Browser tool, Intelligent designer tool
TADEUS	Dialogue graph notation	Textual description for UIMS	TADEUS
MASTERMIND	MDL	C++	Mastermind prototyping support C++ code generator
FUSE	Hierarchic Interaction graph Template	C++	FIRE/FLUID, BOSS, PLUG-IN
TEALLACH	Hierarchical tree with state objects	Java	TEALLACH
OO-Method	OASIS	C# or ASP running on .NET or .NET 2.0; and EJB, JSP, or Java Server Faces running on Java	OlivaNOVA
Just-UI	OASIS	Visual Basic; Java; JSP; ASP; Cold Fusion.	CARE technologies S.A.
UI TERESA	TERESA XML	XHTML and VoiceXML	TERESA
Dygimes	XML, WSDL	Java AWT, Java Swing, Java-enabled Mobile Phone	CTT annotation tool, Spatial layout constraint tool, and Runtime library
SUPPLE	SUPPLE´s functional specification language	Swing, AWT, HTML	SUPPLE, ARNAULD, ABILITY MODELER
GOLIATH	Caml	Java	GOLIATH´s design tool
UsiXML	UsiXML	Java, XUL, XHTML	GrafiXML, IdealXML, SketchiXML, GUILayout++
CoGenIVE	VRIXML	C++	CoGenIVE
MANTRA	Ecore	C-Sharp or XHTML with embedded PHP	Set of Eclipse-based tools
WAINE	ASL	HTML, Java script	WAINE engine
XMobile	XForms, XIML and XML	SuperWaba, J2ME MIDP, J2ME DOJA and J2ME Permsonal Java based on Java	XMobile with interface component framework (XFormUI), and a User Interface Generator
GUMMY	UIML	Java Swing, .Net, DVB MHP	GUMMY, external UIML renderer
FlowiXML	UsiXML	HTML, SCXML	ATOMS extension for FlowiXML
MASP	Not specified	HTML, VoiceXML	Task tree editor as an Eclipse plugin, layouting tool, task tree

			simulator
MARIA	MARIA XML	XHTML, HTML5, JSP +WS access, VoiceXML, X + V, SMIL	MARIA
DB-USE	UsiXML	Java and VB.Net	Model editor, UI builder, Function editor, Code generator.
CIAT-GUI	-	XAML and .Net	CIAT-GUI
UsiComp	UsiXML; Balsamiq mockup exported to XML file	Java	UsiComp Editor and Runtime modules
Cedar	Currently: relational database; planned: XML	C# for the Windows Presentation Foundation	Task, Domain, AUI, CUI and workflows visual tools
LIZARD	UsiXML	Java and XML for Adroid; C# and XAML for Windows phone	Ecore model editor of the Eclipse IDE; LIZARD

Table 6 Ease of use of the MB-UIDE approach

Mb-UIDE: Ease of use criteria	**Needed tools and their interoperability**	**Amount of documentation**	**Clarity of documentation**	**Global evaluation**
ADEPT	○	◖	○	○
GENIUS	○	○	◖	○
TRIDENT	◖	●	◖	◖
JANUS	●	○	○	○
MECANO	◖	○	○	○
TADEUS	●	○	○	○
MASTERMIND	◖	○	○	○
FUSE	◖	○	○	○
TEALLACH	○	●	○	○
OO-Method	◖	●	●	●
Just-UI	◖	○	◖	◖
UI TERESA	◖	○	◖	◖
Dygimes	◖	◖	◖	◖
SUPPLE	◖	◖	◖	◖
GOLIATH	◖	○	○	○
UsiXML	◖	●	●	●
CoGenIVE	●	◖	◖	◖
MANTRA	●	○	○	○
WAINE	◖	◖	○	◖
XMobile	◖	◖	◖	◖
GUMMY	◖	◖	◖	◖
FlowiXML	●	○	○	○
MASP	◖	○	◖	◖
MARIA	●	◖	◖	◖
DB-USE	○	●	●	◖
CIAT-GUI	●	○	◖	◖
UsiComp	●	◖	◖	◖
Cedar	●	◖	◖	◖
LIZARD	◖	◖	○	◖

As shown in Table 6 JANUS, TADEUS, MANTRA, CoGenIVE, FlowiXML, MARIA, CIAT-GUI, UsiComp and Cedar score the best for the sub criterion *Needed tools and their interoperability*, as they each have one single tool for all the phases. Other approaches like TRIDENT, MASTERMIND, GOLIATH and SUPPLE have different tools that are interoperable. LIZARD is interoperable with the Eclipse IDE, where the model instances can be manipulated using the Ecore model editor. GUMMY relies on an external UIML renderer to provide the visual representation of its UIML description, using proxies to communicate between GUMMY and the external renderer. Only TRIDENT, TEALLACH, OO-Method, UsiXML and DB-USE offer enough documentation, which is in addition the clearest for OO-Method, UsiXML and DB-USE. OO-Method and UsiXML score globally the highest. Observe that the clarity of documentation cannot be objectively measured unless by means of a survey; for the purpose of this paper, the authors have assessed the clarity to the best of their knowledge.
Table 7 presents a summary of the analysis of the MB-UIDEs for the seven other criteria from the proposed assessment framework.

Using the framework for the evaluation of MB-UIDEs we can observe that more recent tools score better than older ones, yet only for the criteria *generation of code, code generation extensibility* and *the coverage of the UI development levels*. In general, however, more recent approaches do not always have better scores than the older ones. For example, long existing MB-UIDEs sometimes score better than recent ones for *UI design flexibility* and *integration of UI development in the application development*. We therefore discuss the specificities of each criterion below.

Ease of use: The ease of use is negatively affected by the lack of sufficient documentation and, where present, its lack of clarity. The models and their notation are complex, often hard to learn and use (Da Silva, 2001), and generally MB-UIDEs do not provide enough and clear documentation. As said before, only OO-METHOD and UsiXML have the maximum score in this respect. The interoperability of the tools to develop de UI is not sufficient in the majority of the approaches: they require developers to use different tools. Approaches like DB-USE (Vi Tran, 2010) require developing models (like task and database models) in external tools and then to import them. Also other environments combine several tools and require importing from one tool to another, like in FUSE, MASTERMIND or SUPPLE, which need four and three tools, respectively.

Table 7 Evaluation of the MB-UIDE approaches

Mb-UIDE /Criteria	Ease of use	Genera-tion of code	Code generation extensibility	User adapti-veness	Coverage of the UI development levels	UI design flexibi-lity	Integration of UI development in the application development cycle
ADEPT	○	○	○	●	◖	○	○
GENIUS	○	○	○	○	◖	○	○
TRIDENT	◖	○	○	○	◖	◖	○
JANUS	○	◖	○	○	○	●	○
MECANO	○	○	○	◖	◖	◖	○
TADEUS	○	○	○	◖	◖	○	○
MASTERMIND	○	◖	○	○	○	○	○
FUSE	○	◖	○	◖	◖	○	○
TEALLACH	○	◖	○	◖	●	○	○
OO-Method	●	●	●	○	◖	●	●
Just-UI	◖	●	●	○	◖	○	●
UI TERESA	◖	◖	●	○	●	○	○
Dygimes	◖	◖	●	○	●	◖	◖
SUPPLE	◖	◖	○	●	◖	◖	◖
GOLIATH	○	◖	○	○	◖	○	○
UsiXML	●	◖	●	●	●	●	○
CoGenIVE	◖	◖	◖	◖	◖	○	○
MANTRA	○	◖	●	○	◖	○	○
WAINE (2007)	◖	◖	◖	◖	●	●	◖
XMobile	◖	◖	●	○	◖	◖	○
GUMMY	◖	◖	●	○	◖	●	◖
FlowiXML	○	◖	◖	○	●	○	○
MASP	◖	◖	●	◖	●	●	◖
MARIA	◖	◖	●	○	●	○	○
DB-USE	◖	◖	◖	●	●	○	○
CIAT-GUI	◖	◖	●	○	●	○	○
UsiComp	◖	◖	◖	◖	●	○	○
Cedar	◖	◖	◖	●	●	●	○
LIZARD	◖	◖	●	○	◖	○	○

Code generation extensibility: This criterion is met by the most recent approaches. In general, the analyzed MB-UIDEs that provide facilities to add transformations to generate code for different languages are model-driven approaches. This criterion is well-supported by OO-Method, Just-UI, UI TERESA, Dygimes, UsiXML, MANTRA, XMobile, GUMMY, MARIA, MASP, CIAT-GUI and LIZARD, which provide support to define UI in different platforms by transformations that are implemented by the tool. It is partially supported by CoGenIVE, WAINE, FlowiXML, DB-USE, UsiComp and Cedar.

User adaptiveness: The analyzed approaches do not consider individual user preferences and characteristics, except 1) ADEPT and DB-Use, which cover this criterion with a user model (see Table 4), making groups of users according to the set of preferences and characteristics they share, 2) SUPPLE, which adapts the User Interface to the user´s individual work style and personal preferences, 3) UsiXML, with its GrafiXML tool, which allows specifying a user in a context (which is characterized by attributes such as task experience, disabilities, motivation, experience with interaction devices, preferences), and 4) Cedar which adapts the UI by removing features that are not required by certain end-users, and is more focused on the context of use and the user´s preferences. Although UsiComp claims to employ a user model, there is no example of use, to our knowledge. WAINE does not allows complete user adaptativeness: users are categorized into groups according to their role to have specific security option, but user´s preferences are not taken into account. MASP has a layout model generator where the designer can load contexts-of-use scenarios that contain preferences of the user, but not their characteristics and skills.

Coverage of the UI development levels: TEALLACH, UI-TERESA, Dygimes, UsiXML, WAINE, FlowiXML, MASP, MARIA, DB-USE, CIAT-GUI, MASP, UsiComp and Cedar are the approaches that incorporate all the UI development levels starting with the task and concepts, followed by the AUI, the CUI, and the FUI. With respect to the rest of the approaches, MANTRA, which begins with the AUI, but has the possibility of start with a task model, is closer to covering all the phases compare to the other approaches. All the other approaches use task and concepts or a functional interface specification to start from. Most of them use the abstract and concrete UI. The final UI is achieved by some of the most recent approaches (TEALLACH, OO-METHOD, Just-UI, UI-TERESA, Dygimes, SUPPLE, GOLIATH, UsiXML, CoGenIVE, MANTRA, WAINE; XMobile, GUMMY, FlowiXML, MASP, MARIA, DB-USE, CIAT-GUI, UsiComp, Cedar and LIZARD).

UI design flexibility: In general, the analyzed approaches do not incorporate flexibility for designing UIs. The majority of them obtain a UI with a default design and do not give the possibility to change design options. Only JANUS, OO-method with its extension of transformation templates (Aquino, Vanderdonckt, & Pastor, 2010), UsiXML, WAINE, GUMMY, MASP and Cedar. With the tool GrafiXML (for UsiXML), the designer can draw in direct manipulation any graphical UI by directly placing concrete objects and editing their properties. In WAINE customization is possible by changing the proposed default layout (e.g. Form to table, forbidding edition in the first and third fields, etc.). GUMMY allows drag and drop widgets at the concrete level, and has design flexibility by

allowing for different design options. MASP allows layouting based on several environment-related aspects such as distance. With Cedar, the UI can be adapted for different contexts of use, by using the user´s feedback about whether features are relevant or not, choosing possible alternative layout optimizations. As explained by (Akiki, Bandara, & Yu, 2013) "the CUI design tool supports placeholders upon deletion in addition to complete deletion of elements which could be manually replaced and mapped to the AUI model". SUPPLE uses an optimization algorithm to find the most appropriate widgets according to the individual abilities of a user or the characteristics of the device, but prevents designer input from being made at the CUI level. This makes it difficult to adopt it for enterprise applications, due to the absence of designer control on the final design of the UI. MECANO and TRIDENT deal partially with this criterion. In MECANO, at low level the interface objects can be polished and assigned widgets. TRIDENT´s tool generates a first layout to be further refined by the visual designer. Dygimes allows changing the choices for widgets to be rendered; this can be indicated by adding a rule that contains the full name of the abstract interaction object to be mapped into a concrete one in the templates.

Integration of UI development in the application development: As we can see in Table 7, this is the criterion that is met the least. As also pointed by (Goderis, 2008), in the majority of the studied approaches, except for OO-Method and its extension Just-UI, it is the programmer who has to provide the actual link between the application and its UI. Hence the programmer still lacks the afore-mentioned support in specifying the complex control flows. There are approaches like TERESA (Mori et al., 2004) that represent the interaction abstractly but they do not support interaction modelling together with persistence and functionality. With MARIA (Paterno et al., 2009) it is possible to generate UIs for various platforms from existing UIs, able to keep functioning with the rest of the application. Using TEALLACH (Griffiths et al., 2001) and GOLIATH (Julien, Ziane, & Guessoum, 2005), developers have to define the methods of the application. Dygimes (Coninx, Luyten, Vandervelpen, Van den Bergh, & Creemers, 2003) allows the application logic to describe operations as web services that are linked to the application. MASP (Feuerstack, Blumendorf, Kern, et al., 2008) has a service model which connects backend services to application tasks, but it is still the developer who needs to make the link with the application. This kind of UI generation, where the majority of approaches need to integrate the UI with the rest of the application later, separates the UI development from the rest of application development and therefore lacks support to aid the developer in linking the underlying application with the UI in an integrated design and development effort.

6. Discussion

6.1 Limitations

The limitations of this study relate to the limitations inherent to the search of relevant papers, to the assessment framework and to the evaluation.
While the systematic search process offers some guarantee for completeness, we can never claim 100% certainty that no interesting MD-UIDEs exist beyond the ones we evaluated. All elements contributing to completeness have been taken into account, like the use of two large databases, a sufficient broad set of keywords, and additional backward and forward snowballing to search for additional papers. The fact that we identified 11 extra environments on top of the environments already covered by existing surveys is an indication of reasonable completeness of our search.
Also our assessment framework may present some limitations. The proposed criteria have not been yet approved by the scientific community. However, we have selected those criteria that have been referenced as important features for MB-UIDEs by relevant authors in the field. Objectivity of the evaluation has been strived for by formulating and describing the metrics applied per criterion. Nevertheless, the criterion *Ease of use* may be considered as a partly subjective criterion due to the difficulty to assess clarity of documentation. In this area, there is still a need for more objective measures, like an experimental assessment with UI developers. This would however be quite challenging to achieve, as it would require an extensive test lab and a sufficiently large set of users for testing all 29 environments.

6.2. Identification of Research Gaps

Although many diverse MB-UIDE approaches have been developed, they do not yet fully exploit the benefits and potentialities of MDD. Looking at the different evaluation criteria, and the criteria that are met the least by the tools, we have specifically identified the following research gaps for which further research is needed:

- **Difficult to use.** Even without considering the difficulties to download and install the software for the different MB-UIDs, the majority of the approaches score badly on "Ease of use". One of the main hurdles for ease of use is that designers need to use a specific language to create UI (input) models, which implies the need to learn those languages. The input models and their notations should therefore be easy to learn and build. This implies the availability of sufficient and clear documentation, two criteria on which the majority of the environments scored badly. Additionally, there is a need for environments with high-level tool interoperability, where it is not necessary to export and import models from one tool to another. OO-Method and UsiXML are the environments that score well on the "ease of use" criterion overall, but nevertheless, the sub criterion *Needed tools and their interoperability* still can be improved.
- **Lack of UI design flexibility:** The majority of the analyzed approaches only allow obtaining a UI with a default design where the designer has no control over the

generated UI. UI designers need more flexible approaches for the automatic generation of UIs from models. This flexibility can be provided through UI options that can be selected by the designer upon which the UI is generated according to the chosen options. When tools do not offer sufficient design options, UI code will need to be changed after the generation, and re-generation will then overwrite the changes. The approach should therefore give the designer sufficient options through the generation so that there is no (or less) need to change the code after generation. The UI design flexibility should support the possibility of incorporating new design options making the design process easier for designers. JANUS, OO-Method, UsiXML, WAINE, Gummy, MASP and Cedar provide support for UI design flexibility. The fact that there are already six approaches that support UI design flexibility demonstrates the feasibility of providing options to manipulate the design and as such generate different versions of the UI. Surprisingly, apart from Cedar, the most recent approaches do not support UI design flexibility, which indicates that there is room for improvement to make UIs customizable and usable in different contexts.

- **Lack of complete development support.** While a number of tools offer code generation, including code generation extensibility, in order to produce an accurate application with not too much effort and at low cost, the approaches should not only generate the UI code, but also the full functional application. This is necessary to allow the co-design of a UI with its underlying application. The large majority of approaches fails on this criterion. Only the commercial tool of OO-Method and its extension Just-UI can generate the code of the full application. On the one hand this demonstrates that the knowledge to achieve full functional code generation exists. On the other hand, it also indicates that how to achieve full functional code generation is not yet understood by the majority of MB-UIDEs. Moreover, complete development support does not pertain to code generation only. By now, only the commercial tool of OO-Method provides a complete development support. While this demonstrates the feasibility of an all-encompassing approach, this relies on models specifically tailored for the OO-method approach. Research on general MDD principles, especially for the integration of UI development with the rest of the application is still required.

Although most of the criteria are reached by one or several approaches, there is not a single approach that fulfills all the mentioned criteria at once. In order to take advantage of all the necessary characteristics of MB-UIDE, it is necessary to develop an approach that fully reaches all the advantages of previous MB-UIDEs and adheres to the principles of MDD ensuring the integration of UI development with the development of the rest of the application.

7. Conclusions

The design and implementation of UIs and the underlying application is an iterative process that cycles between design and development until a satisfactory product is achieved. To support this process, multiple MB-UIDE approaches have already been developed. This paper has proposed an assessment framework that identifies and gives metrics to evaluate MB-UIDE approaches according to the most important criteria presented in the literature and to which extent they reach the desirable benefits of MDD. Based on this framework, the paper has evaluated 29 existing MB-UIDE approaches. This evaluation can help developers to find the most appropriate approach according to their specific needs and purpose.

In general, the evaluation shows that current approaches have a number of shortcomings that make interfaces difficult to build, and it shows that the majority of approaches focus on the development of UIs without having a complete integration in the application development cycle. Many analyzed approaches require various tools for generating the final UI, and in some cases also require the use of external tools that are not interoperable. At the very best, the provided UI tools are integrated in a homogenous development environment that also allows the generation of the underlying application that is linked with the interface.

Therefore, further research needs to be developed in the area of MB-UIDE. Specifically, from the evaluation of the existing approaches, we have identified important research gaps in the development of MB-UIDE that open potential future research opportunities in this area.

Ease of use by providing an integrated approach is a first important issue to address. Next to providing an integrated approach for the design and development of the UI only, also integration with application development is an important open issue. In order to decrease development efforts and improve the quality of the whole application, UI development should be considered as part of the whole application development process, and hence also as part of the requirement elicitation, analysis, design, implementation and testing phases. In other words, UI development should not be done in an isolated way, but in an integrated way and by means of an explicit and maintained link with the rest of the application.

8. Acknowledgments

This project was funded by the VLIR-UOS program under the Cuba Network project ZIUS2015AP033.

9. References

Akiki, P. A., Bandara, A. K., & Yu, Y. (2013). RBUIS: simplifying enterprise application user interfaces through engineering role-based adaptive behavior. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems* (pp. 3–12). Conference Proceedings, ACM.

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

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

Section, Springer, Berlin.
Aquino, N., Vanderdonckt, J., & Pastor, O. (2010). Transformation templates: adding flexibility to model-driven engineering of user interfaces. In *SAC'2010* (pp. 1195–1202). Conference Proceedings, Sierre: ACM Press, New York.
Berti, S., Correani, F., Paterno, F., & Santoro, C. (2004). The TERESA XML language for the description of interactive systems at multiple abstraction levels. In *Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages* (pp. 103–110). Conference Proceedings.
Bézivin, J. (2004). In Search of a Basic Principle for Model Driven Engineering. *The European Journal for the Informatics Professional*, *5*(2), 21–24.
Blumendorf, M., Lehmann, G., Feuerstack, S., & Albayrak, S. (2008). Executable models for human-computer interaction. In *International Workshop on Design, Specification, and Verification of Interactive Systems* (pp. 238–251). Conference Proceedings, Springer.
Bodart, F., & Vanderdonckt, J. (1995). Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide. In *Design, Specification and Verification of Interactive Systems, DSV-IS'95* (pp. 262–278). Conference Proceedings, Citeseer.
Botterweck, G. (2007). A model-driven approach to the engineering of multiple user interfaces. In *Models in Software Engineering* (pp. 106–115). Book Section, Springer.
Brambilla, M., Cabot, J., & Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool.
Cabot, J., & Teniente, E. (2006). Constraint Support in MDA tools: a Survey. In *European Conference on Model Driven Architecture-Foundations and Applications* (pp. 256–267). Conference Proceedings, Springer.
Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, *15*(3), 289–308. Journal Article.
Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J., & Creemers, B. (2003). Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In *International Conference on Mobile Human-Computer Interaction* (pp. 256–270). Conference Proceedings, Springer.
Cuppens, E., Raymaekers, C., & Coninx, K. (2005). A model-based design process for interactive virtual environments. In *International Workshop on Design, Specification, and Verification of Interactive Systems* (pp. 225–236). Conference Proceedings, Springer.
Da Silva, P. P. (2001). User interface declarative models and development environments: A survey. In *Interactive Systems Design, Specification, and Verification* (pp. 207–226). Book Section, Springer.
Delgado, A., Estepa, A., & Estepa, R. (2007). Waine: automatic generator of web based applications. In *Third International Conference on Web Information Systems and Technologies* (pp. 226–233).
Engel, J., Herdin, C., & Märtin, C. (2014). Evaluation of Model-based User Interface Development Approaches. In *International Conference on Human-Computer Interaction* (pp. 295–307). Conference Proceedings, Springer.
Feuerstack, S., Blumendorf, M., Kern, M., Kruppa, M., Quade, M., Runge, M., & Albayrak, S. (2008). Automated usability evaluation during model-based interactive system development. In *Engineering Interactive Systems* (pp. 134–141). CHAP, Springer.
Feuerstack, S., Blumendorf, M., Schwartze, V., & Albayrak, S. (2008). Model-based layout generation. In *Proceedings of the working conference on Advanced visual interfaces* (pp. 217–224). Conference Proceedings, ACM.
Gajos, K., & Weld, D. S. (2004). SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces* (pp. 93–100). CONF, ACM.
Goderis, S. (2008). *On the separation of user interface concerns: A Programmer's Perspective on the Modularisation of User Interface Code*. Book, ASP/VUBPRESS/UPA.
Gomaa, M., Salah, A., & Rahman, S. (2005). Towards a better model based user interface development environment: A comprehensive survey. In *MICS 5*. Conference Proceedings.
Griffiths, T., Barclay, P. J., Paton, N. W., McKirdy, J., Kennedy, J., Gray, P. D., … da Silva, P. P. (2001). Teallach: a model-based user interface development environment for object databases. *Interacting with Computers*, *14*(1), 31–68. Journal Article.

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

Guerrero-García, J., Gonzalez-Calleros, J. M., Vanderdonckt, J., & Muñoz-Arteaga, J. (2009). A theoretical survey of user interface description languages: Preliminary results. In *Web Congress, 2009. LA-WEB'09. Latin American* (pp. 36–43). Conference Proceedings, IEEE.

Guerrero-García, J., González-Calleros, J. M., Vanderdonckt, J., & Muñoz-Arteaga, J. (2011). A theoretical survey of user interface description languages: complementary results. *UsiXML 2011*, 229–236. Journal Article.

Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM SYSTEMS JOURNAL*, *45*(3), 451–461.

Jha, N. K. (2005). Low-power system scheduling, synthesis and displays. In *IEE Proceedings-Computers and Digital Techniques* (Vol. 152, pp. 344–352). Conference proceedings.

Julien, D., Ziane, M., & Guessoum, Z. (2005). GOLIATH: An extensible model-based environment to develop user interfaces. In *Computer-Aided Design of User Interfaces IV* (pp. 95–106). Book Section, Springer.

Karanam, M. (2015). MDA Tool Support for Model Driven Software Evolution: A Survey. *Issues*, *1*(1), 11–17. Journal Article.

Kennard, R., & Leaney, J. (2010). Towards a general purpose architecture for UI generation. *Journal of Systems and Software*, *83*(10), 1896–1906. Journal Article.

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. *Engineering* (Vol. 2). http://doi.org/10.1145/1134285.1134500

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

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., & López-Jaquero, V. (2005). USIXML: a language supporting multi-path development of user interfaces. In *Engineering human computer interaction and interactive systems* (pp. 200–220). Book Section, Springer.

Lin, J., & Landay, J. A. (2002). Damask: A tool for early-stage design and prototyping of multi-device user interfaces. In *In Proceedings of The 8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing)* (pp. 573–580). Conference Proceedings.

Lonczewski, F., & Schreiber, S. (1996). The FUSE-System: an Integrated User Interface Design Environment. In *CADUI* (Vol. 96, pp. 37–56). Conference Proceedings.

Luo, P., Szekely, P., & Neches, R. (1993). Management of interface design in HUMANOID. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems* (pp. 107–114). Conference Proceedings, ACM.

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

Märtin, C. (1996). Software Life Cycle Automation for Interactive Applications: The AME Design Environment. In *CADUI* (pp. 57–76). Conference Proceedings.

Mellor, S. J. (2004). *MDA distilled: principles of model-driven architecture*. Book, Addison-Wesley Professional.

Meskens, J., Vermeulen, J., Luyten, K., & Coninx, K. (2008). Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In *Proceedings of the working conference on Advanced visual interfaces* (pp. 233–240). Conference Proceedings, ACM.

Mitchell, K. J., & Kennedy, J. B. (1996). DRIVE: an environment for the organised construction of user-interfaces to databases. In *Proceedings of the 1996 international conference on Interfaces to Databases*. Conference Proceedings, British Computer Society.

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

Molina, J. C., & Pastor, O. (2004). MDA, OO-Method y la tecnología OlivaNova Model Excecution. *I Taller Sobre Desarrollos Dirigidos Por Modelos, MDA Y Aplicaciones. Málaga*. Journal Article.

Molina, P. J., Meliá, S., & Pastor, O. (2002). Just-UI : A User Interface Specification Model. In *CADUI'2002*

(pp. 63–74). Conference Proceedings, Dordrecht: Kluwer Acad. Pub.

Mori, G., Paterno, F., & Santoro, C. (2004). Design and development of multidevice user interfaces through multiple logical descriptions. *Software Engineering, IEEE Transactions on*, *30*(8), 507–520. Journal Article.

Mori, G., Paternò, F., & Santoro, C. (2002). CTTE: support for developing and analyzing task models for interactive system design. *Software Engineering, IEEE Transactions on*, *28*(8), 797–813. Journal Article.

Myers, B. A. (1993). *Why are human-computer interfaces difficult to design and implement* (Report). DTIC Document.

Myers, B. A. (1995). User interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *2*(1), 64–103. Journal Article.

Myers, B. A., & Rosson, M. B. (1992). Survey on user interface programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 195–202). Conference Proceedings, ACM.

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

Paternò, F. (2003). From model-based to natural development. *IST PROGRAMME*. Journal Article.

Paterno, F., Santoro, C., & Spano, L. D. (2009). MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *16*(4), 19. Journal Article.

Puerta, A. R. (1996). The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. In *CADUI* (Vol. 96, pp. 19–36). Conference Proceedings.

Schlungbaum, E. (1996). *Model-based user interface software tools current state of declarative models*. Book, Graphics, Visualization & Usability Center, Georgia Institute of Technology.

Schmidt, D. C. (2006). Model-Driven Engineering. *IEEE Computer*, *39*(2), 25–31.

Seffah, A., Gulliksen, J., & Desmarais, M. C. (2005). *Human-Centered Software Engineering-Integrating Usability in the Software Development Lifecycle* (Vol. 8). Book, Springer Science & Business Media.

Souchon, N., & Vanderdonckt, J. (2003). A review of XML-compliant user interface description languages. In *Interactive Systems. Design, Specification, and Verification* (pp. 377–391). Book Section, Springer.

Stary, C. (2000). TADEUS: seamless development of task-based and user-oriented interfaces. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *30*(5), 509–525. Journal Article.

Sukaviriya, P., Foley, J. D., & Griffith, T. (1993). A second generation user interface design environment: The model and the runtime architecture. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems* (pp. 375–382). Conference Proceedings, ACM.

Szekely, P. A., Sukaviriya, P. N., Castells, P., Muthukumarasamy, J., & Salcher, E. (1995). Declarative interface models for user interface construction tools: the MASTERMIND approach. In *EHCI* (pp. 120–150). Conference Proceedings, Citeseer.

Vanderdonckt, J. (2008). Model-driven engineering of user interfaces: Promises, successes, and failures. In *ROCHI'2008* (pp. 1–10). Conference Proceedings.

Viana, W., & Andrade, R. M. C. (2008). XMobile: A MB-UID environment for semi-automatic generation of adaptive applications for mobile devices. *Journal of Systems and Software*, *81*(3), 382–394. Journal Article.

Vi Tran, T. A. (2010). *User interface generation from task, domain and user models: DB-USE approach* (Thesis). Louvain School of Management.

Vi Tran, T. A., Vanderdonckt, J., Kolp, M., & Wautelet, Y. (2010). Generating User Interface for Information Applications from Task, Domain and User models with DB-USE. In *1st International Workshop on USer Interface eXtensible Markup Language UsiXML'2010* (pp. 183–194). Conference Proceedings.

Wiecha, C., Bennett, W., Boies, S., Gould, J., & Greene, S. (1990). ITS: a tool for rapidly developing interactive applications. *ACM Transactions on Information Systems (TOIS)*, *8*(3), 204–236. Journal Article.

Appendix 1

Exclusion criteria for the systematic search.

- Short papers
- Papers analyzing approaches in an early stage
- Books of proceedings
- Papers with techniques for a specific aspect or development phase (e. g. adaptation technique, prototyping technique, layouting technique, consistency check technique, requirement elicitation phase) (or comparison of techniques)
- Papers with analysis of languages or notations (or comparison of languages or notations)
- Papers with tools used to edit some kind of model
- Papers for which the full text is not available
- Paper discussing a prediction tool
- Paper with usability assessment
- Paper analyzing principles of unified UI development
- Paper presenting specification of interactive questionnaires
- Paper with a taxonomy to evaluate task models
- Paper with a web based ontology editor
- Paper with authoring tool
- Paper with a discussion of patterns

Table 8 presents the number of discarded papers according to these exclusion criteria.

Table 8 Number of discarded papers and the causes

Cause	**Discarded**
Short papers	15
Early stage	19
Book of proceedings	15
Technique for only a specific aspect or development phase comparison of techniques	49
Language or notation, or comparison among languages or notations	21
Tool for some kind of model	26
Paper not available	6
Remaining reasons	35
Total	186