Gerrit Meixner, Fabio Paternò, Jean Vanderdonckt

Past, Present, and Future of Model-Based User Interface Development

Vergangenheit, Gegenwart und Zukunft der modellbasierten Entwicklung von Benutzungsschnittstellen

Model-Based User Interface Development, MBUID, User Interface Management Systems, UIMS, User Interface Software and Technology, Challenges

Zusammenfassung. Dieser Artikel gibt einen Überblick über die letzten 30 Jahre der Forschung im Bereich der modellbasierten Entwicklung von Benutzungsschnittstellen. Dazu wird zunächst die Geschichte aufgearbeitet, bevor wesentliche Konzepte und aktuelle Ansätze beschrieben werden. Der letzte Abschnitt zeigt aktuelle Herausforderungen auf und gibt Implikationen, wie diesen zukünftig begegnet werden kann.

Summary. This article presents the past, present and future of model-based user interface development. After 30 years of research there has been significant success in modeling user interfaces. This article aims to give a comprehensive overview of the history, describes important aspects and current approaches, lists actual challenges of model-based user interface development and gives implications for the next generation.

1. Introduction

The development of user interfaces (UI) for interactive systems became a time-consuming and, therefore, costly task with the diffusion of the graphical UIs, as was shown in a study of Myers and Rosson (Myers and Rosson, 1992). By analyzing a number of different software applications, it was found that about 48% of the source code, about 45% of the development time, about 50% of the implementation time, and about 37% of the maintenance time are required for aspects regarding UI. These values still seem to be valid today, because the spread of interactive systems as well as their requirements have drastically increased over the last years. Today, developers of UI for interactive systems have to face several challenges: 1. Heterogeneity of end users: an interactive system is normally used by several different end users. End users differ with respect to, e.g., their preferences, capabilities, speaking different languages and level of experience; 2. Heterogeneity of computing platforms, input ca-

pabilities and modalities: today, there is a large diversity of computing platforms (e.g., Smartphone, Desktop PC, PDA, embedded devices) using different input capabilities (e.g., keyboard, mouse, (multi-) touch, data gloves) with different interaction modalities (e.g., graphic, speech, haptics, gesture); 3. Heterogeneity of programming/markup languages and widget toolkits: for developing a UI, developers use different programming/markup languages (e.g., Java, C++, HTML) with different widget libraries (e.g., Swing, Qt, GTK+) and 4. Heterogeneity of working environments: today, many workflows in the real world are supported by interactive systems through the pervasiveness of computing devices, i.e. that end users have to cope with different contextual constraints (e.g., noisy environments).

Model-Based User Interface Development (MBUID) is one approach which aims to cope with the above mentioned challenges and which aims to decrease the effort needed to develop UIs.

This paper starts with a historical overview of the research field of User Interface Management Systems (UIMS) and

MBUID, then, we describe current, important approaches and finally, we discuss future challenges that must be solved in order to cope with the challenges and to achieve the promised goals.

2. History

2.1 User Interface Management Systems (UIMS)

The term and the first concepts of a UIMS were coined and developed first in 1982 by Kasik (Kasik, 1982), but a system by Newman from 1968 – the Reaction Handler – is called the first UIMS, because it integrated several properties of a UIMS (Löwgren,1988; Hix, 1990).

The outcome of a workshop on software tools for UI development at the ACM SIGGRAPH conference in 1987 was, amongst other things, a common definition of a UIMS (Betts et al., 1987):

"A User Interface Management System is a tool (or tool set) designed to encourage interdisciplinary cooperation in the rapid development, tailoring and
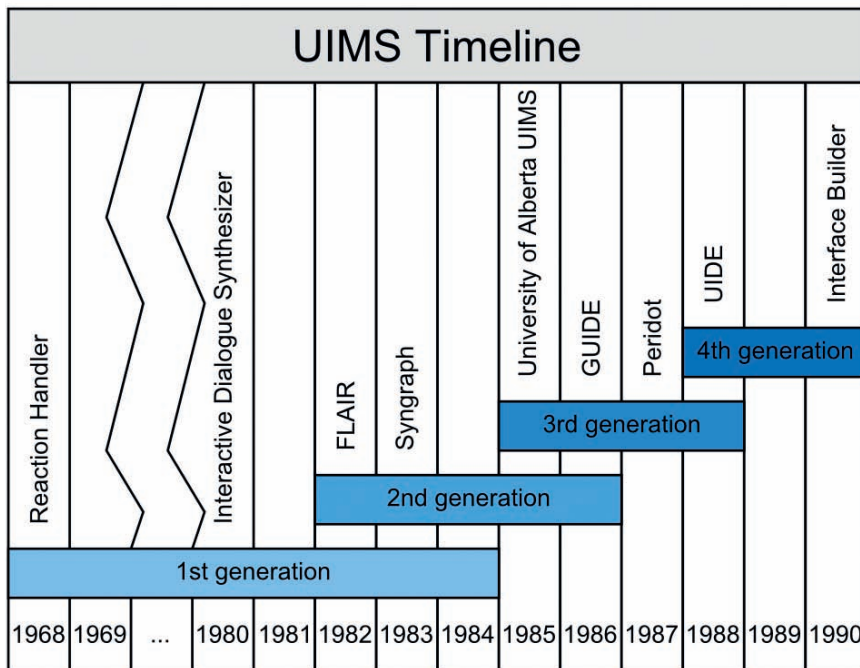
Fig. 1: Chronological overview of the UIMS generations.

management (control) of the interaction in an application domain across varying devices, interaction techniques and UI styles. A UIMS tailors and manages (controls) user interaction in an application domain to allow for rapid and consistent development. A UIMS can be viewed as a tool for increasing programmer productivity. In this way, it is similar to a fourth generation language, where the concentration is on specification instead of coding."

Deborah Hix analyzed different UIMS approaches that she divides into four generations based on the target audience and the methods of the UIMS (Hix, 1990). In the early 1980s, most UIs were teletype UIs (text-based dialogue system) with a keyboard and a monitor as the common environment. The first (approximately 1968–1984) and the second (approximately 1982–1986) UIMS generation targeted, therefore, only teletype UIs, whereas the third (approximately 1985-1988) and fourth (approximately 1988-1990) generation targeted graphical direct manipulation UIs. Fig. 1 gives a chronological overview about the four different UIMS generations and denominates different examples of each generation.

The first UIMS generation often used formal grammars (e.g., Backus-Naur-Form) supplemented by a conventional programming language to specify (parts) of the UI. This generation only targeted programmers. The second generation extended their functional capabilities and used mainly the notation of a transition network to specify parts of the UI. Also the second generation needed a lot of programming knowledge for specification. Therefore, it also targeted only programmers. The third generation shifted the view away from programming the UI towards more designing the UI. The third generation targeted more (complex) graphical UIs with direct manipulation as the main interaction style. To better support the interface designer and developers (who used UIMS), the fourth generation started to target the improvement of the UI of a UIMS themselves by using the WIMP paradigm. This should reduce the required amount of programming skills which was needed to use a UIMS. Furthermore, in the fourth generation concepts from the area of artificial intelligence (e.g., knowledge bases, expert systems) were integrated to better support developers specifying a UI with a UIMS.

Research on UIMS was a well-established area in the Human-Computer-Interaction (HCI) community at that time, but they were not widely available and used in the industry. The missing general acceptance of UIMS is caused by several major problems (Myers, 1987): 1. UIMS are too hard to use: especially the first three UIMS generations were hard to use for non-programmers like e.g., UI designer. UI designers had to learn programming techniques and languages in order to be able to use these UIMS at all. 2. UIMS are too limited in the types of UIs they can create: only a few UIMS directly addressed WIMP UIs. WIMP UIs are much more complex then teletype UIs, because WIMP UIs e.g., supported different interaction techniques, allowed multiples interaction techniques to be available and operated at the same time. 3. UIMS are not portable: UIMS were too tightly coupled with a computing platform, operating system or widget toolkit. The main reason was that at that time UIMS directly had to deal with low-level aspects like managing data of the various input devices (e.g., mouse, keyboard) and "drawing" the interaction objects of the UI on the screen of the output device.

To solve these problems in the upcoming generations Myers (Myers, 1995) suggested 1. more research and development in the area of graphical specification techniques and UI development processes and 2. more investigation in the area of developing (high-level) UI models. Furthermore, Green (Green, 1985) suggested 1. to start the development of the UI in with a task model and 2. to (better) integrate UIMS in software engineering processes to receive a better integration to the core of the application functionality.

2.2 Model-Based User Interface Development (MBUID)

The evolution from UIMS to the new paradigm of MBUID (Da Silva, 2000) is based on the fact that through the late 1980s modeling languages (e.g., object-oriented programming languages) for creating richer and more complex UIs were developed which were in turn used by MBUID systems to generate more sophisticated UIs (Puerta and Szekely, 1994; Szekely, 1996).

Schlungbaum (Schlungbaum, 1996) defines two necessary criteria for a UI tool to be a model-based user interface

development environment (MBUIDE): 1. "MBUIDE must include a high-level, abstract and explicitly represented (declarative) model about the interactive system to be developed (either a task model or a domain model or both)" and 2. "MBUIDE must exploit a clear and computer-supported relation from (1) to the desired and running UI. That means that there is some kind of automatic transformation like knowledge-based generation or simple compilation to implement the running UI."

These necessary criteria are conforming to the former mentioned suggestions. MBUID could, therefore, be a possible solution to overcome the major problems of UIMSs (Puerta and Szekely, 1994) which could then lead to a general acceptance in the industry. Furthermore, the interest in MBUID increased again due to the increasing diversity of platforms.

Currently, there are four identified generations of MBUID (Schlungbaum, 1996; Da Silva, 1999; Paternó, Santoro and Spano, 2009). Fig. 2 gives a chronological overview about the four different MBUID generations and denominates different important developments during the generations.

The first generation (approximately 1990–1996) focused on identifying and abstracting relevant aspects of a UI. Tools in this generation mainly used one universal declarative UI model which integrated all relevant aspects of a UI. Approaches (mainly engineering-centric tools) focused on the fully automatic generation of the UI instead of an integrated holistic MBUID process. Examples for the first generation are UIDE, AME or HUMANOID.

The second generation (approximately 1995–2000) focused on the extension of the UI model by integration of other distinct models into a MBUIDE and expressing the high-level semantics of a UI. Therefore, the UI model is structured into several other models like e.g., task model, dialog model or presentation model. With the second generation, developers were able to specify, generate and execute UI. Much emphasis has been done on the integration of task models (e.g., CTT) into MBUID. Furthermore, a user-centered development (UCD) approach for UI on the basis of a task model has been recognized as crucial for the effective design of UIs.
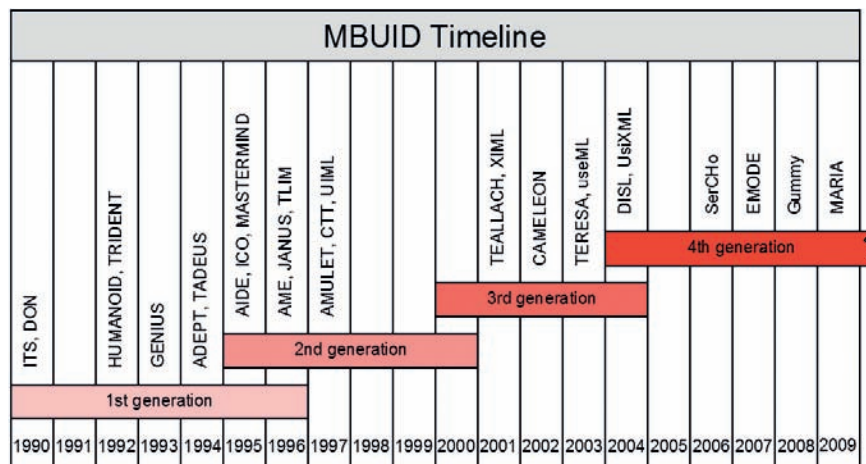

Fig. 2: Chronological overview of the MBUID generations.

Examples for the second generations are ADEPT, TRIDENT or MASTERMIND.

The third generation (approximately 2000–2004) was mainly driven by the plethora of new interaction platforms and devices. Mobile devices like e.g., smartphones or PDAs, became popular. Indeed, as Myers, Hudson and Pausch indicated while discussing the future of UI tools, the wide platform variability encourages a return to the study of some techniques for device-independent UI specification (Myers, Hudson and Pausch, 2000) Then, the system might choose appropriate interaction techniques taking all of these into account. Developers and designers had to face the challenge of developing a UI for several different devices with different constraints (e.g., screen size). An expressive integrated MBUIDE became more relevant than in the generations before. Examples for the third generation are TERESA or Dygimes.

The current fourth generation (approximately 2004–today) is focusing on the development of context-sensitive UIs for a variety of different platforms, devices and modalities (multi-path development) and the integration of web-applications. Central elements of most of the current approaches are models which are stored (mostly) as XML-based languages to enable easy import and export into authoring tools. Furthermore, one research focus is on the optimization of the automatically generated UIs by ensuring a higher degree of usability. Today, MBUID approaches are often called "model-driven" and not "model-based" anymore.

Model-driven UI development puts models at the center of the development process, which are then (semi-)automatically transformed into an executable code or rendered by an interpreter. Current examples of the fourth generation are given in the next section.

3. Current Approaches

Today, there is an almost common understanding of the abstraction layers and type of models which need to be considered for the development of current UIs within MBUID, but there is still no consensus (and no standards) about the information (semantics) the different models have to contain. Therefore, this section shortly introduces the three core models (which have direct influence on the content and appearance of a UI), the Cameleon Reference Framework and gives some examples of the current fourth generation MBUIDE.

3.1 The Three Core Models

Task model: a task model represents a description of the tasks which can be accomplished by users of interactive systems. These tasks can be hierarchically decomposed into subtasks until they are on the level of elementary actions. Tasks are related by temporal relations and can have conditions which regulate their execution.

Dialog model: a dialog model describes the set of actions (or tasks) the user can carry out within various system
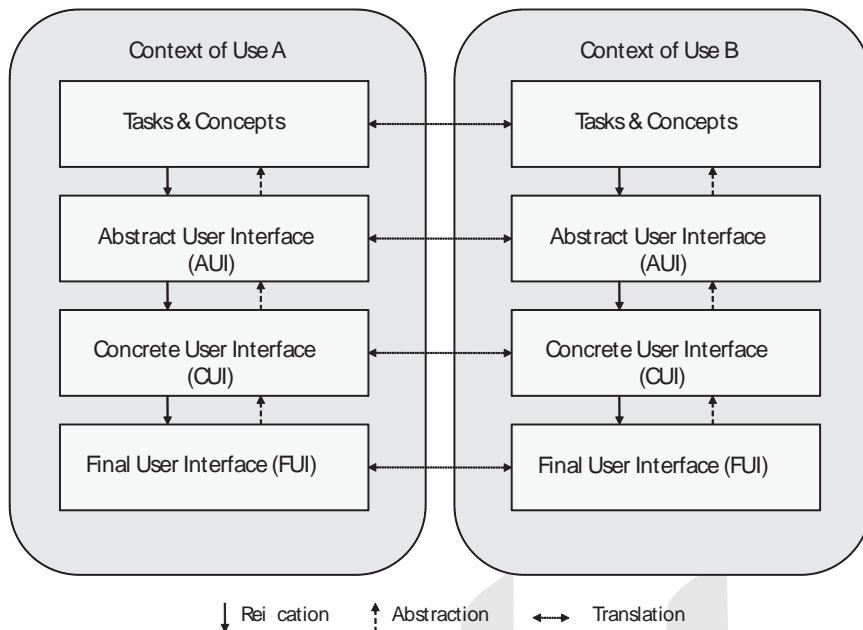
Fig. 3: The (simplified) Cameleon Reference Framework.

states, and the transitions between states that are triggered by certain actions. Not being a stand-alone model, it is derived from the task model by evaluating the temporal relation between tasks. It also connects the tasks with corresponding interaction elements, forming a bridge between the task and the presentation model.

Presentation model: a presentation model represents (a hierarchal composition of) the visual, haptic or auditory elements that a UI offers to its users.

Different UI aspects, such as the presentation and the dialog model, but not limited to it, can be specified by a UI Description Language (UIDL) (Souchon and Vanderdonckt, 2003).

3.2 The Cameleon Reference Framework

Different frameworks were developed over the last years to capture conceptually the important parts of a MBUID process. Szekely introduces a generic architecture for MBUID already in 1996 (Szekely, 1996). In 2000, Da Silva describes an architecture for UI development with a MBUID approach (Da Silva, 2000). Finally, in 2002, the Cameleon Reference Framework (CRF) (Calvary et al., 2002) resulted from the EU-funded FP 5 CAMELE-ON project and has been subsequently

refined and revised (Calvary et al., 2003). It describes a framework that serves as a reference for classifying UIs that support multiple targets, or multiple contexts of use on the basis of a model-based approach. The framework covers both the design-time and runtime phases of multi-target UIs. Furthermore, the CRF provides a unified understanding of context-sensitive UIs rather than a prescription of various ways or methods for tackling different steps of development. As such, it has now become widely accepted in the HCI community to locate each related work.

As depicted in Fig. 3, the framework describes different layers of abstraction, which are important for MBUID, and their relationships among each other: the Task and Concepts level considers, e.g., the hierarchies of tasks that need to be performed in a specific temporal order in order to achieve the users' goals (during the interaction with a UI); the Abstract UI (AUI) expresses the UI in terms of Abstract Interaction Objects (AIOs) (Vanderdonckt and Bodart, 1993). These AIOs are independent of any platform or modality (e.g., graphical, vocal, haptic). Furthermore, AIOs can be grouped logically; the Concrete UI (CUI) expresses the UI in terms of Concrete Interaction Objects (CIOs) (Vanderdonckt and Bodart, 1993). These CIOs are modality-dependent, but

implementation-language-independent. The CUI defines more concretely how the UI is perceived by the users, and the Final UI (FUI) expresses the UI in terms of implementation-dependent source code. A FUI can be represented in any UI programming language (e.g., Java UI toolkit) or mark-up language (e.g., HTML). A FUI can then be interpreted or compiled.

Between these levels there are different relationships (Limbourg and Vanderdonckt, 2009) e.g., Reification covers the inference process from high-level abstract descriptions to runtime code. The CRF recommends a four-step reification process: a Concepts-and-Tasks Model is reified into an Abstract UI which in turn leads to a Concrete UI. A Concrete UI is then turned into a Final UI, typically by means of code generation techniques; Abstraction is an operation intended to map a UI representation from one non-initial level of abstraction to a higher level of abstraction. In the context of reverse engineering, it is the opposite of reification; Translation is an operation that transforms a description intended for a particular target into a description at the same abstraction level but aimed at a different target. It is not needed to go through all steps: one could start at any level of abstraction and reify or abstract depending on the project.

3.3 CTT+MARIA

The ConcurTaskTrees (CTT) notation (Paternò, 1999) has represented an important contribution towards engineering task models making them exploitable in various contexts at both design and runtime. It has a set of features that make it suitable to easily represent activities that need to be carried out to reach the users' goals: hierarchical structure, temporal relations, icons to indicate task allocation, and a set of attributes to indicate various aspects (e.g., task type, task objects and relevant platforms for task execution). Recently, the possibility of better specifying pre-conditions has been added. Such pre-conditions can also be considered by the associated interactive simulator, which is included in the ConcurTaskTrees Environment (a publicly downloadable tool for editing and analyzing task models). The CTT specifications can be saved in XML

format in order to include and exploit them in other tools and environments. CTT and the associated tool have been exploited over time in various application domains e.g., interactive safety-critical systems (such as ATC), ERP applications and service engineering.

The MARIA language (Paternò, Santoro and Spano, 2009) addresses different abstraction layers: the Abstract and Concrete UI. Also in this case there is an associated publicly downloadable tool (MARIAE). This language has been developed based on the experiences gathered with previous approaches in order to: support a Data Model, which is useful for specifying the format of input values, association of various data objects to the various interactors; specify events at abstract/concrete levels, which can be property change events or activation events (e.g., access to a web service or a database); include an extended dialog model, obtained through conditions and CTT operators for event handlers, thus allowing specification of parallel input; support UIs including complex and Ajax scripts with the possibility of continuously updating of fields without explicit user request; and describe a dynamic set of UI elements with the possibility of conditional connections between presentations and changes to only a part of a UI. The associated tool supports the editing of Abstract UIs in the MARIA language, which can be derived from a task model or created from scratch. The editor supports browsing the specification through an interactive tree view of the specification and a graphical representation of the elements of a selected presentation, in addition to showing the XML specification. The editor allows the editing through drag-and-drop of the elements and their attributes. From the abstract description, it is possible to derive concrete descriptions for various platforms: desktop, mobile, vocal, multimodal. Each concrete description can be presented and edited in modalities similar to those for the abstract specifications. From the concrete descriptions, it is possible to obtain implementations for various implementation languages (XHTML, HTML5, JSP + WS access, VoiceXML, X+V, SMIL) through associated transformations.

3.4 UsiXML

The USer Interface eXtensible Markup Language (UsiXML) is structured according to four levels of abstraction defined by the CRF, including the context model and the quality model. UsiXML relies on a transformational approach that progressively moves among levels to the FUI (Limbourg et al., 2005). The transformational methodology of UsiXML allows the modification of the development sub-steps, thus ensuring various alternatives for the existing sub-steps to be explored and/or expanded with new sub-steps. As such, UsiXML supports model-driven engineering of UIs as defined by the Object Management Group (OMG). Designers can shape the UI of any new interactive application by specifying and describing it in the UIDL, without requiring programming skills usually found in markup languages and programming languages. UsiXML allows cross-toolkit development of an interactive application. A UI of any UsiXML-compliant application runs in all toolkits that implement it. UsiXML supports device independence: a UI can be described in a way that remains autonomous with respect to the devices used in the interactions (e.g., mouse, screen, keyboard, voice recognition system). In case of need, a reference to a particular device can be incorporated. UsiXML supports platform independence: a UI can be described in a way that remains autonomous with respect to the various existing computing platforms (e.g., mobile phone, Pocket PC, Tablet PC, kiosk, laptop, desktop, and wall screen). In case of need, a reference to a particular computing platform can be incorporated. UsiXML supports modality independence: a UI can be described in a way that remains independent of any interaction modality (e.g., graphical interaction, vocal interaction, 3D interaction, virtual reality interaction). In case of need, a reference to a particular modality can be incorporated. UsiXML allows reusing elements previously described in anterior UIs to compose a UI in new applications.

Historically, the first version of UsiXML resulted from the EU-funded FP5 Cameleon project and has been continuously supported by the following projects: FP6 Similar, FP7 Human, FP7 Selfman, FP7 Serenoa, and ITEA2 UsiXML. UsiXML V2.0 is now accessible via the UsiXML End User Club (Calvary et al., 2011).

3.5 useML+DISL+UIML

This approach uses three different XML-based languages at the different abstraction layers of the CRF to model UIs (Meixner, Breiner and Seissler, 2011).

useML

The Useware Markup Language (useML) 1.0 was developed to support a user-centered development (UCD) process (ISO 9241-210) with a modeling language representing the results of the initial task analysis. Accordingly, the use model (task model) abstracts platform-independent tasks into use objects (UO) that make up a hierarchically ordered structure. Furthermore, the leaf tasks of a use model are described with a set of elementary use objects (eUO) representing atomic interactive tasks: inform, trigger, select, enter and change. In Version 2.0, useML was extended by five temporal operators to support temporal relationships as well as it provides the possibility to define multiple executions or (pre-/post-) conditions that can be attached to tasks of the model (Meixner, Breiner and Seissler, 2011). This information can be used later in the transformation process to derive a dialog model. useML is supported by Udit – an interactive editor and simulator for use models. Udit is also able to transform use models into DISL models.

DISL

The abstract UI is modeled with the Dialog and Interface Specification Language (DISL), which was developed at the University of Paderborn, Germany, as a modeling language for platform- and modality-independent UIs for mobile devices. DISL focuses on scalability, reactivity, easy usability for developers, and low demands on processing power and memory consumption. An important precondition to the systematic development of UI is the strict separation of structure, presentation and behavior of a UI. DISL supports only 8 generic (meaning platform and modality-independent) widgets, but allows the extension for future generic widgets. Recently, an interactive DISL editor has been developed.
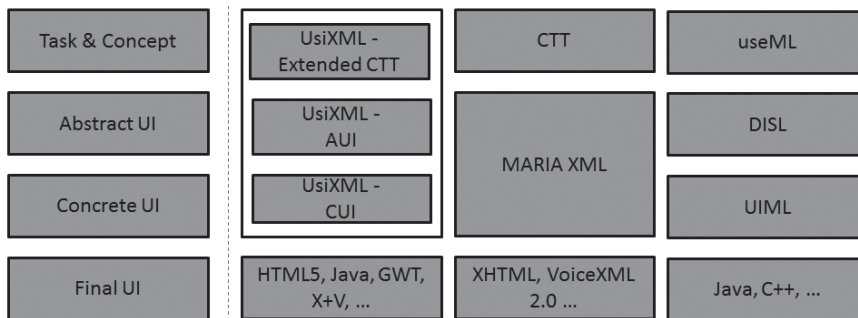
Task & Concept	UsiXML - Extended CTT	CTT	useML
Abstract UI	UsiXML - AUI		DISL
Concrete UI	UsiXML - CUI	MARIA XML	UIML
Final UI	HTML5, Java, GWT, X+V, ...	XHTML, VoiceXML 2.0 ...	Java, C++, ...

Fig. 4: Relating the current approaches with the CRF.

C1. Need to ensure quality properties of a model
C2. Need to cover semantics, syntax, and stylistics
C3. Difficulty of identifying the minimal amount of models
C4. Risk of model proliferation
C5. Support annotation-based UI design
C6. Support (de)composition
C7. Support multi-path development of UIs
C8. Support multi-fidelity
C9. Support method engineering
C10. High threshold
C11. Low ceiling
C12. Wide walls
C13. Unpredictability
C14. Lack of propagation of modifications
C15. System dependent and private models
C16. Need for a common User Interface Description Language (UIDL)
C17. Need for improved effort incrementality
C18. Need for advanced modeling for dynamic aspects
C19. Need for powerful transformation and rendering engines
C20. Need to ensure model traceability

Fig. 5: The 20 challenges of MBUID (Vanderdonckt, 2008).

UIML

The User Interface Markup Language (UIML) has been developed as a specification for an abstract meta-language that can provide a canonical XML representation of any UI and has been standardized in version 4.0 by the OASIS. The UI description is implementation-language-independent since it uses a generic vocabulary to specify the interaction objects. These interaction objects can be translated into interaction objects by using the peers-element, which is an addition to the vocabulary to map the CIOs to their representation in the target language. There are peers for several languages, including Java Swing, XHTML and the .NET components. UIML is not only able to describe the presentation of the UI; it is also capable of specifying its behavior.

3.6 Relating the Current Approaches

Fig. 4 visualizes the three current approaches which are compliant to the CRF. On the left side, the abstraction layers of the CRF are shown, followed by the UsiXML approach, followed by the CTT+MARIA approach and finally, by the useML+DISL+UIML approach.

4. Future Challenges

Even after more than 30 years of research, MBUID is not a common practice in the daily industrial software development.

In this section, we describe several important future challenges which have to be solved by the upcoming fifth MBUID generation. Several well-known researchers in the MBUID community have defined challenges and drawbacks of the last MBUID generations, e.g., (Puerta and Szekely, 1994), (Szekely, 1996), (Vanderdonckt, 2008), (Calvary and Pinna, 2008). In the following, we will give a summary of the most important challenges which are mainly based on the 20 identified challenges by Vanderdonckt (see Fig. 5) and which have to be addressed in future. Most recognized challenges can be classified into the following main categories: standardization, holistic model-driven development process, tool support as well as real-word usage and case studies.

4.1 Standardization

In 1999, Da Silva recognized already that standard notations for MBUID are necessary: "The use of a standard notation may be useful in order to describe different UI models using a common set of constructors. In fact, these constructors may facilitate the comparison and the reuse of UI models and their MBUIDEs. For instance, the reuse of UI models may be difficult these days since they are based on several notations [...]. Further, the reuse of UI models can be essential for make MBUIDEs scalable for real applications." (da Silva, 2000) Furthermore, Calvary and Pinna state that "... there is a need of reconciliation to clarify the state of the art." (Calvary and Pinna, 2008).

Challenges and Discussion

C2: generally, standard notations have rigorously defined semantics, syntax and stylistics.

C3: a standardized framework with different entry points would support developers in finding the ideal entry point for the development process. Generally, the development of a (more) usable UI should start by conducting a task and context analysis followed by task modeling activities. (Semi-)automatic transformations support developers and reduce the amount of manual work to deduce the other models. In practice, developers will need larger case studies, real world MBUID implementations and document-

ed experiences for getting the right feeling which models are really needed in in their type of project.

C15, C16: publicly and freely available standardized languages would be the key for developing associated interoperable open tools and development environments. Having an unambiguous standard notation, the exchange of models between tools and MBUIDE will be improved. Basic model operations and algorithms could be defined in an abstract language (programming language independent) so that the tools which implement the model operations and algorithms are able to reuse the knowledge of the algorithms. During the last 20 years of research in MBUID, many different UIDLs have been developed, but they are often limited in terms of adoption in real-world industrial software development.

C18: recently much effort has been investigated by different groups to define powerful notations for modeling the dynamic aspects. Among others, there are the DISL and State Chart XML (SCXML). The dialog model of DISL has been incorporated into the OASIS standard of the UIML. SCXML is an XML-based markup language for providing a generic state-machine based execution environment based on Harel statecharts which will be certainly standardized by the W3C (currently SCXML is a working draft).

Up to now, some efforts have been done in the area of standardization. UIML 4.0 has been standardized in 2009 by the OASIS (Helms et al., 2009). ANSI/CEA-2018 (published in 2008) defines an XML-based language for task model descriptions relevant to consumer electronics devices. The MBUI Incubator group at W3C has evaluated research on model-based UIs as a framework for authoring web applications and with a view to proposing work on related standards published as a report (http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/). Currently, a working group is being established at W3C (http://www.w3.org/2011/01/mbui-wg-charter) to initialize standardization with focus on: use cases and requirements, specification of meta-models for interchange of models between authoring tools for (context aware) UIs for web-based interactive application front ends,

specification of a markup language and API which realize the meta-models, test assertions and test suite for demonstrating interoperability, model-based UI design primer and Open Source Implementations.

4.2 Holistic Model-Driven Development Process

As stated by Löwgren: "If UIMSs (and analogously MBUIDE) could be methodologically integrated in software development processes, they would significantly contribute to the development of even better application systems." (Löwgren, 1988), Model-Driven Engineering (MDE) has become an important paradigm in the software development community. MDE uses models as primary artifacts during the analysis, design, implementation and maintenance phases of the system lifecycle and employs automated model management mechanisms for model analysis, validation and transformation to support systematic and efficient software development. In the industry, MDE is common practice. Several companies have developed mature commercial development environments. But there is a clear lack of harmonization between MDE and MBUID approaches: while the MDE community has focused on the issue of genericity (e.g., generic transformation languages, generic meta-model extension techniques, generic visual language definition mechanisms), the MBUID community has focused on UI specific aspects (e.g., definition and refinement of task, dialog and presentation models). Although the core concepts of both approaches are largely similar (based on models), they developed mostly independently within different communities. Not surprisingly, there are different types of stakeholders involved in both approaches that have grown accustom to their own vocabularies, modeling techniques and processes. Research is, therefore, fragmented into groups focusing on different abstractions, languages and tools, which are parts of larger, mostly disjoint communities, each publishing within their own conferences and journals. In addition, MBUID has focused primarily on fragmented UI models, largely neglecting the relation to other software views, such as models of relat-

ed workflows. This often resulted in solutions that are incompatible with other MDE approaches at the conceptual level or at the level of transformation tools. MDE and MBUID must be combined and treated as one holistic approach for model-driven development of interactive systems. Bridging both domains is indispensable for future sustainable design and development of complex interactive systems. Furthermore, Calvary and Pinna support our argument of a holistic model-driven development approach for interactive systems by stating that "... horizontal collaboration between the functional core and the UI are not yet well supported." (Calvary and Pinna, 2008).

Challenges and Discussion

C7, C8, C9: a holistic model-driven development approach for interactive systems which is integrated in a UCD methodology should be developed. Models could then be developed and later refined when more information has been acquired. The development and evaluation of mockups and prototypes is a vital part in UCD. Therefore, tools have to be developed which visualize (parts of) the model in a way which is suitable for the communication between different stakeholders in the development process.

Currently, there are several researchers working on the integration of MDE, MBUID, UCD and usability concepts into a holistic process, e.g., Meixner, Breiner and Seissler show how a UCD process can be extended by UI models heading towards a user-centered MBUID process (Meixner, Breiner and Seissler, 2011). Lallemand analyzes the possible integration of usability at different stages of software development on the basis of MDE (Lallemand, 2011).

4.3 Tool Support

Tool support has always been one of the major challenges in the history of MBUID (Myers, 1987). Puerta and Szekely state that "... the main shortcoming of the model-based approach is that building models is difficult. A rich model describing all the features of an interface is a complex object, and hence non-trivial to specify." (Puerta and Szekely, 1994). Therefore, supporting the different types
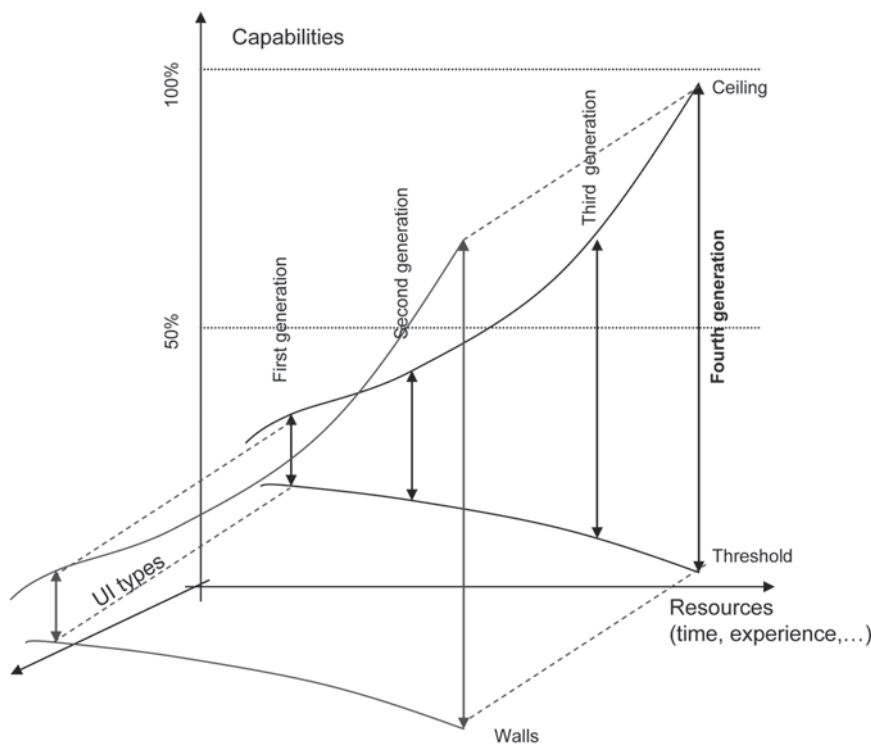
Fig. 6: Low threshold, high ceiling, and wide walls for MBUIDEs.

of involved stakeholders (e.g., programmers, UI designer, interaction designer) with tools they really need and accept is absolutely essential.

Challenges and Discussion

C1: a model should be at least complete, consistent, and correct. Having a standard notation with a clear defined semantics a model can easily be checked by e.g., model checking tools.

C5, C6, C10: for developing, maintaining and deploying UI models, developers should be supported by an easy-to-use WYSIWYG editor (interface builder). Such an editor prevents users from learning one or more particular specification language(s). Even if the developers have to learn a specification language, the automation of a portion of the development should reduce the development effort. Furthermore, an editor should have the features the different developers need, e.g., annotating further information to models or model fragments, possibility of "copy&paste" of interaction objects (widgets) from one model to another. Intelligent layout mechanisms should support developers by automatically providing possible arrangements for the new interaction objects.

C13, C14, C20: transformations must be transparent for developers, in a way they are able to understand the transformation and its effects (traceability). Furthermore, larger real-world case studies are needed to show possible side effects and to enhance model propagations and transformations. This could also lead to a more comprehensive understanding about side effects.

C11, C12, C19: MBUIDEs must be extensible regarding transformations. With standardized notations the community is able to build and share better transformations for a wide range of target platforms. Furthermore, the usability of automatically generated UIs is rather low. A fully automatic transformation approach seems currently not suitable. This challenge has often been mentioned in the history of MBUID (Calvary and Pinna, 2008; Myers et al., 1990; Myers, 1995). Presumably, the solution lies in semi-automatic transformation approaches. Transformations are supporting developers in standard tasks, but developers are able to refine manually the generated UI (tweaks, beautifications). Manual refinements of generated UIs are lost, when developers regenerate other draft designs of the UI, i.e. manual modifications have to be in-

corporated in the models again to ensure round-trip engineering. Besides manually tweaking the generated UI, the integration of Human-Computer-Interaction (HCI) patterns in transformation processes seems to be a promising approach to encounter generated UIs with a low level of usability. Furthermore, the integration of formalized standards and guidelines into a MBUIDE is desirable to support manual adaptations done by developers and designers (Meskens et al., 2011). The need for the integration of ergonomic criteria is also mentioned by Calvary and Pinna: "… one open issue specific to HCI is the modeling of ergonomic criteria." (Calvary and Pinna, 2008).

Low threshold and high ceiling (Myers, 1990) could be augmented by wide walls (Fig. 6) (Vanderdonckt, 2008): not only it is crucial that the threshold, with which designers and developers could create a UI with the MBUIDE, is as low as possible (i.e., the time required to create a UI is minimal) and that the ceiling is high (i.e., the capability to create a UI is large), but also the walls should be wide (i.e., the resources required to create different UIs should be minimal). The last criterion is vital: often MBUIDEs have been criticized for having a high threshold and a low ceiling. Nowadays, the new generations of MBUIDEs lower the threshold and increase the ceiling, but also enlarge the walls.

Currently, there are several excellent freely tools and MBUIDE available (see e.g., section 3.3 – 3.5). In the future, research and development on tool support is still essential for MBUID. Tool support should, therefore, be pushed after the community has developed clear standards and has defined a holistic model-driven development approach for interactive systems.

4.4 Real-World Usage and Case Studies

In 1990, Hix stated already that "eventually – probably in this decade – UIMSs will have sufficient functionality, usability, and accessibility to be found readily in real-world development environments." (Hix, 1990) Until now, MBUIDEs have not been (commercially) successful in real-world usage. Furthermore, Calvary and Pinna state

in 2008 that "… the global picture [of MBUID] has been demonstrated on simple case studies in HCI so far, we now aim at going beyond toy applications." (Calvary and Pinna, 2008).

Challenges and Discussion

C3, C4, C17: the future of MBUID could heavily benefit from larger case studies and real-world usage. Until now, only the research community has seen smaller UIs or UI fragments, which have been generated by MBUID processes. The development and the adaptation of UI models, languages and their MBUIDE are a key factor for the adoption by the industry. The development of domain-specific transformations and domain-specific languages at, e.g., CUI level could be a promising approach.

Requirements concerning the development of UIs for infotainment systems in the automotive industry are demanding for a flexible MBUID process. Currently, there is a large-scale public-funded research project in the German car industry which targets exactly this problem and tries to establish a real-world usage of MBUID in this complex domain. The project automotiveHMI (http://www. automotive-hmi.org) aims to improve the process of the development of UI in the automotive sector. An integrated approach based on standardized languages, models and interfaces targets to improve the development efficiency of all companies and involves from car manufacturers and component suppliers to producers of the used tools. The description based on abstract models also enables the convergence of new, multi-media and multimodal forms of interaction in completely new operating systems. The outcome of this project – the establishment of a mature MBUID process in the German car industry – could be a very important milestone for the future of MBUID.

5. Conclusion and Outlook

Modeling languages, transformations and development tools got much more expressive and easier to use, but UIs and their development became also more complex since the manifestation of the WIMP paradigm. Today, being in the post-WIMP area (e.g., Mixed Reality, Tangible UIs, 3D-UIs), UI development is a still complex and time-consuming task. In order to get an impression about the question if MBUID approaches could be successful in the future, we first gave a brief overview about the history of UIMS and MBUID. After explaining the core models of UI development and the different abstraction layers of the CRF, we gave a short overview about current approaches of the fourth generation of MBUID. The last section of this paper was about the main challenges of the upcoming fifth MBUID generation.

Currently, researchers are working hard to solve the main challenges. Despite all the problems and challenges researchers encountered during the last 20 years, we think that by solving the above mentioned main challenges, MBUID could be successful in the near future so that MBUID possibly gets accepted by companies which develop software for real-world usage and not only for showing the feasibility of an approach by demonstrating the automated generation of portions of a UI.

References

Betts et al.: Goals and Objectives for User Interface Software. Computer Graphics, 21(2), pp. 73–78, 1987.

Calvary, G. et al.: The CAMELEON Reference Framework, CAMELEON Project, September 2002, available at http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEON%20D1.1RefFramework.pdf, 2002.

Calvary, G. et al.: A Unifying Reference Framework for multi-target user interfaces. Interacting with Computers, 15(3), pp. 289–308, 2003.

Calvary, G.; Pinna, A.-M.: Lessons of Experience in Model-Driven Engineering of Interactive Systems. Proc. of the 1st International Workshop on Challenges in Model-Driven Software Engineering (ChaMDE), 2008.

Calvary, G. et al.: User Interface eXtensible Markup Language SIG. Proc. of IFIP TC13 Int. Conf. on Human-Computer Interaction, Lecture Notes in Computer Science, Vol. 6949, Springer, Berlin, pp. 693–695, 2011.

Da Silva, P. P.: User Interface Declarative Models and Development Environments: A Survey. Proc. of the 7th International Conference on Design, Specification, and Verification of Interactive Systems, pp. 207–226, 2000.

Green, M.: The University of Alberta User Interface Management System. Proc. of the 12th Annual Conference on Computer Graphics and Interactive Techniques, pp. 205–213, 1985.

Helms, J. et al.: Human-Centered Engineering with the User Interface Markup Language. In A. Seffah, J. Vanderdonckt, M. Desmarais (eds.), Human-Centered Software Engineering, Springer, London, pp. 141–173, 2009.

Hix, D.: Generations of user-interface management systems. IEEE Software, 7(5), pp. 77–87, 1990.

Kasik, D. J.: A User Interface Management System, Proc. of the 9th Annual Conference on Computer Graphics and Interactive Techniques, pp. 99–106, 1982.

Lallemand, C.: Toward a closer integration of usability in software development: a study of usability inputs in a model-driven engineering process. Proc. of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems, pp. 299–302, 2011.

Limbourg, Q. et al.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004. Lecture Notes in Computer Science, Vol. 3425, Springer, Berlin, pp. 200–220, 2005.

Limbourg, Q.; Vanderdonckt, J.: Multi-Path Transformational Development of User Interfaces with Graph Transformations, In A. Seffah, J. Vanderdonckt, M. Desmarais (eds.), Human-Centered Software Engineering, Springer, London, 2009.

Löwgren, J.: History, State and Future of User Interface Management Systems. ACM SIGCHI Bulletin, 20(1), pp. 32–44, 1988.

Meixner, G.; Breiner, K.; Seissler, M.: Model-Driven Useware Engineering. In H. Hussmann, G. Meixner, D. Zuehlke (eds.), Model-Driven Development of Advanced User Interfaces, Springer, Heidelberg, 2011.

Meskens, J. et al.: GUIDE2ux: a GUI design environment for enhancing the user experience. Proc. of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing System, pp. 137–142, 2011.

Myers, B.: Gaining General Acceptance for UIMSs. Computer Graphics, 21(2), pp. 130–134, 1987.

Myers, B.: User Interface Software Tools. ACM Transactions on Computer-Human Interaction, 2(1), pp. 64–103, 1995.

Myers, B.; Rosson, M. B.: Survey on User Interface Programming. Proc. of the 10th Annual CHI Conference on Human Factors in Computing Systems, pp. 195–202, 2000.

Myers, B.; Hudson, S.; Pausch, R.: Past, present, and future of user interface software tools. ACM Transactions on Computer-Human Interaction, 7(1), pp. 3–28, 2000.

Paternò, F.: Model-based Design and Evaluation of Interactive Applications, Springer Verlag, 1999.

Paternó, F.; Santoro, C.; Spano, L. D.: MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. ACM Transactions on Computer-Human Interaction, 16(4), 2009.

Puerta, A. R.; Szekely, P.: Model-based interface development. CHI'94 Conference companion on Human factors in computing systems, pp. 389–390, 1994.

Schlungbaum, E.: Model-Based User Interface Software Tools – Current State of Declarative Models. Technical Report, 96–30, Graphics, Visualization and Usability Center, Georgia Institute of Technology, 1996.

Souchon, N.; Vanderdonckt, J.: A Review of XML-Compliant User Interface Description Languages. Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems. Lecture Notes in Computer Science, Vol. 2844, Springer, Berlin, pp. 377–391, 2003.

Szekely, P.: Retrospective and Challenges for Model-Based Interface Development. Proc. of the 3rd International Eurographics Workshop, pp. 1–27, 1996.

Vanderdonckt, J.; Bodart, F.: Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. Proc. of the ACM Conf. on Human Factors in Computing Systems. ACM Press, New York, pp. 424–429, 1993.

Vanderdonckt, J.: Model-Driven Engineering of User Interfaces – Promises, Successes, Failures, and Challenges. Proc. of the 5th Annual Romanian Conference on Human-Computer Interaction, pp. 1–10, 2008.

1 Gerrit Meixner is Senior Researcher at the German Research Center for Artificial Intelligence (DFKI) where he leads the Center for Human-Machine-Interaction (ZMMI). He is scientific coordinator of the German automotiveHMI project. He is co-demonstration chair for EICS 2012, chair of the W3C Model-based User Interfaces Working Group and member of the Academic Advisory Board of USID Foundation (India).

2 Fabio Paternò is Research Director at CNR-ISTI, Pisa, Italy, where he leads the Laboratory on Human Interfaces in Information Systems. He wrote a book on Model-Based Design and Evaluation of Interactive Applications. He has been the scientific coordinator of five EU projects (MEFISTO, GUITARE, EUD-Net, CAMELEON, OPEN) and one of the main investigators in several others. He is Chair of IFIP WG 2.7/13.4. He was appointed ACM Distinguished Scientist in 2010.

3 Jean Vanderdonckt is Professor of Computer Science at Louvain School of Management, Université catholique de Louvain (Belgium) where he leads the Louvain Interaction Laboratory (www.lilab.be). The mission of this lab is about engineering interactive information systems, based on usability engineering and human-computer interaction. He is the scientific coordinator of the European ITEA2 UsiXML project and involved in FP7 Human, FP7 Selfman, and FP7 Serenoa European projects. He is co-editor in chief of the Springer HCI Series.