Adaptive model-driven user interface development systems

Adaptive Model-Driven User Interface Development Systems

Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu

{pierre.akiki, arosha.bandara, yijun.yu}@open.ac.uk

Computing and Communications Department

The Open University, Walton Hall, Milton Keynes, United Kingdom

Adaptive user interfaces (UIs) were introduced to address some of the usability problems that plague many software applications. Model-driven engineering formed the basis for most of the systems targeting the development of such UIs. An overview of these systems is presented and a set of criteria is established to evaluate the strengths and shortcomings of the state-of-the-art, which is categorized under architectures, techniques, and tools. A summary of the evaluation is presented in tables that visually illustrate the fulfillment of each criterion by each system. The evaluation identified several gaps in the existing art and highlighted the areas of promising improvement.

Categories and Subject Descriptors: **D.2.11 [Software Engineering]**: Software Architectures – Domain-specific architectures; **D.2.2 [Software Engineering]**: Design Tools and Techniques – User interfaces; **H.5.2 [Information Interfaces and Presentation]**: User Interfaces – User-centered design

General Terms: Design, Human Factors

Additional Key Words and Phrases: Adaptive user interfaces, model-driven engineering

1. INTRODUCTION

The user interface (UI) layer is considered one of the key components of software applications since it connects their end-users to the functionality. Well-engineered and robust software applications could eventually fail to be adopted due to a weak UI layer. Some user interface development techniques such as: universal design [Mace et al. 1990], inclusive design [Keates et al. 2000], and design for all [Stephanidis 1997] promote the concept of making one UI design fit as many people as possible. Yet, a UI is not independent from its context-of-use, which is defined in terms of the user, platform, and environment [Calvary et al. 2003]. The "one design fits all" approach is unable to accommodate all the cases of variability in the context-of-use, in many cases leading to a diminished user experience. Building multiple UIs for the same functionality due to context variability is difficult since the scope of variability cannot be completely known at design-time and there is a high cost incurred by manually developing multiple versions of the UI. Adaptive UIs have been promoted as a solution for context variability due to their ability to automatically adapt to the context-of-use at runtime. User interfaces capable of adapting to their context-of-use are also referred to as multi-context or multi-target [Fonseca 2010]. A key goal behind adaptive UIs is *plasticity* denoting a UI's ability to preserve its usability across multiple contexts-of-use [Coutaz 2010]. Norcio and Stanley [1989] consider that the idea of an adaptive UI is straightforward since it simply means that: "*The interface should adapt to the user; rather than the user adapting to the system*" (p. 399)

This work is supported by The Open University and ERC Advanced Grant 291652.

The published version will appear in ACM Computing Surveys volume 47, issue 1 in March 2015.

but they note that in spite of the simplicity of the definition, there are some difficult and complex problems relating to adaptive UIs. In our study of the literature, we noticed that some of these problems are technical and are related to devising systems that can support the development of adaptive UIs, while others are related to human factors such as the end-user acceptance of these UIs. Realizing the abstract properties illustrated in Fig. 1, could help in handling some of the technical and human problems related to adaptive UIs.

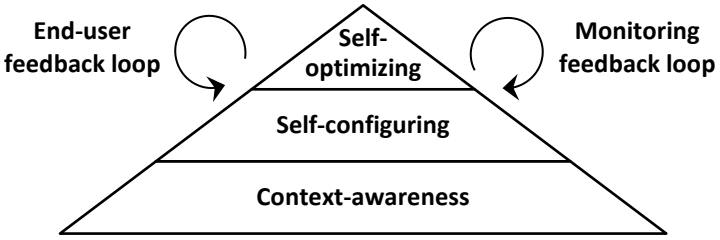

Fig. 1. Self-* Properties of Adaptive User Interfaces

Salehie and Tahvildari [2009] present a hierarchy of adaptability properties for software systems, referred to as self-* properties. This hierarchy demonstrates different complexity levels in software application adaptability. We consider that the following properties on this hierarchy are applicable to the domain of adaptive UIs:

— **Context-awareness** "*indicates that a system is aware of its context, which is its operating environment*" (p. 5). If the UI is aware of its context and is able to detect context changes, then it can trigger adaptations (e.g., based on a set of rules) in response to those changes in order to preserve its usability.

— **Self-configuring** "*is the capability of reconfiguring automatically and dynamically in response to changes*" (p. 5). To keep the UI adaptation rules up to date with an evolving context-of-use (e.g., if a user's computer skills improve), there is a need for a mechanism that can reconfigure these rules by monitoring such changes. Another type of rule reconfiguration could be based on the end-users' feedback. For example, the end-user may choose to reverse a UI adaptation or select an alternative. Keeping the end-users involved in the adaptation process could help in increasing their awareness and control, thereby improving their acceptance of the system.

— **Self-optimizing** "*is the capability of managing performance and resource allocation in order to satisfy the requirements of different users*" (p. 5). To adapt this definition to user interfaces, we can say that a UI can self-optimize by adapting some of its properties. For example, adding or removing features, changing layout properties (e.g., size, location, type, etc.), providing new navigation help, etc.

The triplet (user, platform, and environment) forming the context-of-use can be considered as categories of aspects that could promote adaptive UI behavior. The *user* can have an impact on changing the context in terms of variable needs. The needs could be monitored through each user's behavior upon using the system or be predefined through a set of dynamically configurable rules. For example, the behavior of physically disabled users can be monitored through the speed and accuracy of their mouse clicks and hovering, enabling the UI to be adapted accordingly. On the other hand a user's countries of birth and residence could be used to adapt the UI according to predefined, dynamically configurable rules based on

cultural preferences. The definition of *platform* can accommodate both physical devices (e.g., phone, tablet, laptop, etc.), operating systems, and different types of application platforms (e.g., web, desktop, rich internet application, etc.) [Aquino et al. 2010]. Variability in screen size and the available UI widgets are common examples of aspects that could spur platform related adaptive UI behavior. Changes in the *environment* such as: distance from display devices and mobility, could also incur a change in the context hence requiring the UI to adapt.

Numerous applications suffer from usability problems because their UIs do not cater for context variability. Enterprise applications such as enterprise resource planning systems are but one example of such applications [Topi et al. 2005]. Adaptive UIs have been suggested for enhancing usability in these applications by catering to the variable user needs [Singh and Wesson 2009]. Many approaches were proposed for developing adaptive UIs targeting different types of software systems based on aspects such as: accessibility [Gajos et al. 2010], concurrent tasks [Bihler and Mügge 2007], culture [Reinecke and Bernstein 2011], natural context [Blumendorf et al. 2007], platform [Demeure et al. 2008], etc. Other aspects such as: mental workload [Byrne and Parasuraman 1996] and cognition [Solovey et al. 2011] were discussed in non-technical works and could be targeted by future systems.

This survey primarily targets at the topic of adaptive UI development systems that adopt a model-driven approach. This topic spans the disciplines of software engineering and human-computer interaction. We focus mostly on the systems that adopt model-driven engineering (MDE) since it offers several advantages and has been receiving the most attention in the literature. Our main aim is to demonstrate the strengths and shortcomings of the state-of-the-art and offer recommendations for future research.

The scope of this paper is narrowed down progressively in Sections 2 and 3. Section 2 discusses the different approaches to UI development and adaptation, and it also evaluates these approaches based on criteria taken from the literature to justify our focus on the model-driven approach. Section 3 primarily provides an overview of early model-based UI development systems and justifies why we focused on the latest generation of systems. The evaluation criteria based on which we assess the state-of-the-art are established in Section 4 either based on recommendations from the literature or by combining features from multiple existing systems.

Our evaluation of the state-of-the-art, which we classified into the dimensions of architectures, techniques, and tools, is presented in Sections 5, 6, and 7 respectively. We believe that a comprehensive system targeting the development of adaptive UIs should provide a reference architecture depicting the various characteristics of the proposed approach, a practical technique to achieve the sought after adaptive behavior based on this reference architecture, and a support tool for stakeholders to develop UIs and adapt them with the proposed adaptation technique.

Finally, our conclusions and future outlook are given in Section 8 based on the results of the evaluation we conducted.

2. APPROACHES TO USER INTERFACE DEVELOPMENT AND ADAPTATION

The existing approaches for developing adaptive user interfaces could be classified under two categories: window managers and widget toolkits, and model-driven engineering. Window managers provide a programming model to control the UI's appearance while widget toolkits are reusable code-based libraries of UI components that can support adaptation capabilities. On the other hand, the model-driven engineering (MDE) approach does not rely directly on code for creating the UIs but on higher level specifications from which the UI could be derived. In MDE, the adaptive behavior is usually applied to one of the levels of abstraction before deriving the final user interface that gets presented to the end-user.

This section provides an overview of the traditional approach to UI development and compares the approaches undertaken for developing adaptive user interfaces. We established the following criteria based on the existing literature and will use them as a basis for evaluating and comparing the approaches:

— **Checking** the adaptive behavior is important to avoid conflicting outputs since this behavior is defined by humans and is thereby error-prone. For example, if a procedure is defined to eliminate part of the UI for a given context-of-use, having the ability to check for a dependency between the removed part and the rest of the UI is important to maintain the UI's functionality [Bergh et al. 2010].

— **Completeness** is defined in terms of the types of user interfaces that can be produced using a certain UI development approach [Florins 2006]. Some approaches might be only suitable for developing a particular type of user interfaces such as WIMP UIs. This criterion could be the same as *generality*, which is the ability of applying the solution to a variety of cases [Myers et al. 2000].

— **Control over the UI** is related to the level of details that the designer can manipulate and the predictability of the final outcome [Florins 2006]. Some automated approaches only allow high-level designer input, hence decreasing the control and the predictability of the outcome; while others allow lower-level input such as control over the concrete widgets. Designer input helps in providing different versions of the UI, one of which is designed by a human and others adapted for a particular purpose. Fully-mechanized UI construction has been criticized in favor of applying the intelligence of human designers for achieving higher usability [Pleuss et al. 2010]. It would be better if the designer could manipulate a concrete object rather than its abstraction [Demeure et al. 2009].

— The **cost of developing adaptive UIs** is an important factor that could affect the adoption of this approach. Cost is one of the factors affecting the success of any interactive computer system from the vendor's point of view [Mayhew 1999].

— The **learning curve** is affected by how common an approach is in a market or software-development company. It has also been related to the *threshold* that indicates how difficult it is to use a system for constructing UIs [Myers et al. 2000].

— **Technology independence** allows a UI development approach to cover a wider range of existing technologies and to take into consideration new technologies that could emerge in the future. One approach promoting technology independence is the use of UI description languages (UIDLs) such as: UsiXML [Limbourg and Vanderdonckt 2004], UIML [Abrams et al. 1999], etc.

— **Traceability** *"is the ability to establish degrees of relationship between two or more products of a development process, especially products having a predecessor-*

successor or master-subordinate relationship to one another" [Galvao and Goknil 2007] (p. 314). In adaptive UI development, it could be necessary to trace the adaptation performed and revert back to the original UI either partially or fully.

2.1 Traditional Development : Programming, Event, Scripting, and Markup Languages

Using *programming languages* for user interface development has been investigated for some time. The **Mikey** [Olsen,Jr. 1989] system and its predecessor **MIKE** [Olsen,Jr. 1986] are early propositions for managing user interfaces using programming languages. Mikey provided an example of applying Pascal to develop UIs for the Apple Macintosh and MIKE was an attempt towards a User Interface Management System (UIMS). Another approach focused on using object-oriented languages [Schmucker 1987]. The first attempts in UI development at Xerox PARC used interpreted programming languages such as Smaltalk and Dlisp that allow developers to easily make changes and test the new UI version. Although this feature was lost with compiled languages like C++, it persists in other languages such as those used for hypertext markup (e.g., HTML and XHTML).

Event languages allowed developers to control various UI related events (e.g., input and output). Early research work on these languages included the **Sassafras** [Hill 1986] and the **University of Alberta UI Management System** [Green 1985]. These types of languages became popular with modern commercial graphical user interface presentation technologies such as: Visual Basic Forms, .NET Windows Forms, Java Swing, etc. Therefore, event languages became part of the visual UI design tools in integrated development environments (IDEs) such as: Visual Studio, Eclipse, etc. Modern languages like the Windows Presentation Foundation (WPF) combine markup languages with programming languages to separate the language used for designing the UI from that used for coding the functionality behind it.

Today, the traditional approach to UI development employs one of the existing presentation technologies. There is a variety of software applications types each relying on different presentation technologies. The following are some examples:

—Desktop Applications: .NET Windows Forms and WPF, Java Swing and AWT, etc.

—Web Applications: HTML, XHTML, CSS, VRML, etc.

—Rich Internet Applications (RIA): Silverlight, XUL, Flex, etc.

Although the traditional development approach has a low learning curve, high completeness, and control over the UI, it has several disadvantages when developing adaptive UIs. The two main disadvantages are technological dependency and the high difficulty in adapting the UI to various contexts-of-use without significantly increasing the development cost. Sections 2.2 and 2.3 present two UI development approaches, namely window managers and widget toolkits, and MDE, which were adopted by many systems for developing adaptive UIs.

2.2 Window Managers and Widget Toolkits

Window managers provide developers with a programming model to control the way the user interface appears on the screen. Yet, a direct use of window managers proved to be tedious hence toolkits were developed to make UI construction easier. Toolkits provide a library of widgets and a framework for managing UI creation

using this library. Each widget is a component that can manage its own appearance on the screen. Early efforts towards toolkits were the **Apple Macintosh Toolbox** [Huxham et al. 1986] and the **Andrew Toolkit** [Palay 1989].

There are approaches that operate on the window level and could be classified as being adaptable rather than adaptive, indicating that manual adaptation is performed by the user. One approach allows HTML based UIs to be adapted by the end-users through a toolkit with predefined adaptation operations that could store changes in a central repository as Cascading Style Sheets (CSS) [Nebeling and Norrie 2011]. UI **Façades** [Stuerzlinger et al. 2006] were presented as a technique for allowing end-users to adapt UIs by the means of drag-and-drop whereby any part of a window can be moved to a different location either inside the same window or to another one.

Toolkit-based approaches for adaptive user interfaces have been explored extensively in the literature and attempt to address specific UI adaptation problems. A molecular architecture is offered alongside a toolkit called **Ubit** to provide UI adaptation operations such as: magic lenses, transparent tools, and semantic zooming [Lecolinet 2003]. The **caring, sharing widgets** are presented as part of a toolkit that offers widgets with multiple built-in output modalities that can be swapped based on different aspects such as: screen size, processing power, etc. [Crease et al. 2000]. A system called **Fruit** also focuses on multi-modality to support the needs of users with disabilities and those operating in special environments [Kawai et al. 1996]. The **selectors** are semantic-based controls that can be presented in a variety of ways in order to replace classical widgets that have a fixed appearance [Johnson 1992]. The **Ubiquitous Interactor** targets device independent UIs by separating the presentation from user interaction and services [Nylander et al. 2004]. Widget-level adaptation is also promoted by **WAHID**, which allows the incorporation of adaptive behavior in new and legacy applications based on internal and external architectures [Jabarin and Graham 2003]. Both **ICON** [Dragicevic and Fekete 2001] and **SwingStates** [Appert and Beaudouin-Lafon 2006] are toolkits based on Java Swing. ICON provides an editor that supports the configuration of input devices allowing them to be connected to graphical software interactions whereas SwingStates uses state-machines to extend existing Java Swing widgets with new interaction techniques.

Widget toolkits reduce the cost of developing adaptive UIs when compared to the traditional development approach and maintain completeness and control. Yet, there are downsides to using this approach. Widget toolkits do not improve technological independence since they are tightly coupled with a single programming language or presentation technology (e.g., selectors with C++, WAHID with MFC, ICON and SwingStates with Java Swing, etc.). Also, they are in many cases hard to extend or non-extensible and do not support traceability. As indicated by Demeure et al. [2008], toolkit-based approaches do not support temporal operators on tasks (e.g., sequence, interleaving, etc.) in a similar manner to MDE (e.g., ConcurrTaskTree [Paternò 1999]), which results in losing the transformation that changes the UI. Another disadvantage of toolkit-based approaches is their inability to perform checking of the overall adaptation impact. One example of such checking is the dependency between UI tasks when the adaptation eliminates certain tasks based on changes in the context-of-use [Bergh et al. 2010].

2.3 Model-Driven Engineering

Model-driven development (MDD) is promoted by approaches such as the Model-Driven Architecture (MDA), which provides a technology independent means for absorbing the effect of constant changes in technology and business requirements [Soley and OMG Staff Strategy Group 2000]. MDA is about using modeling languages as programming languages rather than merely as design languages since this can improve the productivity, quality, and longevity outlook [Frankel 2003]. MDA unites the Object Management Group's (OMG) well-established modeling standards with past, present, and future middleware technologies to integrate "what you have built, with what you are building and what you are going to build". Rather than focusing on yet another "next best thing", MDA raises the bar and designs portability and interoperability into the application at the model level [OMG 2013].

Model-driven engineering (MDE) has a wider scope than MDA's development activities and combines process and analysis with architectures [Kent 2002]. Since MDE aims to raise the level of abstraction of software applications, it can serve as a basis for devising adaptive UIs due to the possibility of applying different types of adaptations on the various levels of abstraction. This approach has received the most attention in the literature. We differentiate between the following model-driven approaches that can be used for developing user interfaces:

— **Static modeling** relies on models for UI design and eventually ends in a phase before code generation. By definition, static models cannot change at runtime hence are not useful beyond the development phase.

— **Generative runtime modeling** keeps the models alive at runtime to adapt the code-based UI artifacts that were generated at design-time.

— **Interpreted runtime modeling** does not require code generation for creating the UI. Instead, the models are interpreted at runtime to render the UI.

Runtime models constitute an important area of research in model-driven engineering [France and Rumpe 2007]. Also, runtime models are usually more suited for supporting adaptive behavior. However, in certain scenarios using runtime models while maintaining the generated code-based artifacts is insufficient for achieving the sought after adaptations. Some adaptive scenarios require support for actions such as: eliminating widgets; replacing a widget with another; adding new widgets that did not exist during the development phase; or composing a completely new UI from existing UIs. Performing such actions at runtime could be difficult when the user interface is based on generated artifacts. One problem, for example, is the inability to compose new UIs at runtime since the application is expecting to render the UI from code instead of models. Also, substituting a widget with another would be difficult since the types are hard coded, whereas with runtime interpretation the types could be switched in the model and the widget would be rendered accordingly. In contrast, with interpreted runtime models code generation is not needed for creating the UI but the models are interpreted and rendered at runtime thereby allowing more advanced adaptations. Additionally, by adopting interpreted runtime models, the adaptation could be delegated to a server hence the client machine will be merely responsible for rendering the UI from the adapted model. This method provides a clear separation of concerns. Another benefit of adopting interpreted runtime models is the ability to deploy UI modifications without recompiling the application.

When comparing the MDE approach for UI construction with traditional techniques, Myers et al. [2000] indicate that this approach suffers from a high learning curve. Yet, although the learning curve is generally higher for MDE than traditional development techniques, developers could quickly get used to MDE for devising UIs if the appropriate tool support is provided. Additionally, when assessing whether this learning curve is justifiable we can see that MDE is adding value to traditional and toolkit-based approaches by enhancing traceability, technology independence, and the ability to perform checking on the overall outcome of the UI adaptation. We can say that MDE is a viable approach to use and that other research works (e.g., [Florins 2006]) grade it positively in terms of its development costs. Furthermore, Myers et al. [2000] consider that MDE suffers from unpredictability and has a low ceiling indicating that it is incapable of producing advanced UIs. Yet, this consideration is made with fully-automated MDE-based approaches in mind. Nevertheless, other MDE-based approaches apply semi-automated procedures that allow advanced and predictable UIs to be produced by supporting designer input on all levels of abstraction, especially on the concrete UI. Therefore, we consider the ability of MDE to provide good completeness and control over the UI to be dependent on the implementation.

2.4 Summary

We can see that each of the previously discussed approaches has some advantages and disadvantages based on the criteria that we established. However, from our analysis of the approaches with respect to the criteria outlined previously (Table I), we think that model-driven engineering is overall better suited for devising adaptive UIs.

Table I. Comparison between Approaches to User Interface Development

	Traditional Development	Widget Toolkits	MDE
Checking			
Completeness			
Control Over the UI			
Development Cost of Adaptive UIs			
Learning Curve			
Technology Independence			
Traceability			

Legend

Good	
Average	
Poor	
Depends on the Implementation	

Due to the advantages provided by the model-driven approach in devising adaptive user interfaces, and due to its wide discussion in the literature we shall dedicate the remainder of the paper to explore MDE-based adaptive UI development systems. The next section covers early model-based UI development systems in addition to general-purpose frameworks and architectures, which could serve as a basis for modern adaptive model-driven user interface development approaches. In the remainder of the paper, we explore, evaluate, and compare adaptive UI architectures, techniques and tools that either partially or fully adopt the model-driven approach.

3. BACKGROUND

Many early model-based UI development systems were presented in the literature. We shall briefly discuss their strengths and shortcomings. Additionally, some works have presented frameworks and architectures that can serve as a basis for designing UIs and adaptive systems in general. We also provide an overview of these works and explain the potential of using them for devising adaptive model-driven UIs.

3.1 Model-Based User Interface Development

Model-based UI development (MBUID) has been around since the 1980's. Meixner et al. [2011] differentiate between four generations of MBUID systems. The first generation mainly focused on automatically generating UIs but did not provide an integrated MBUID process while the second generation provided developers with the ability to: specify, generate and execute UIs. The third generation mainly focused on the challenge of developing UIs for a variety of interaction platforms and the current (fourth) generation is focusing on the development of context-sensitive UIs. In the current generation of MBUID systems, models and transformations are at the heart of the development process, making UI development model-driven instead of model-based. An existing survey compared and analyzed 14 of the first and second generation MBUID systems, which are mostly concerned with improving model-based UI development or generating UIs from models [Da Silva 2001]. Therefore, this sub-section only provides an overview of these early MBUID systems and their strengths and shortcomings while the rest of the paper tackles recent (4th generation) systems that target the development of adaptive UIs based on a model-driven approach.

3.1.1 First and Second Generation MBUID Systems. A number of early systems were presented and primarily focused on improving UI development by making it simpler for developers to devise and maintain user interfaces.

Some systems simply focused on providing better means for UI development. **COUSIN** [Hayes et al. 1985] is a UIMS that targets the development of better quality UIs at a low cost by focusing on providing a level of abstraction in the sequencing of the UI dialog (ordering of input/output events). **ITS** [Wiecha et al. 1990], a four-layer tool-supported architecture, was an early attempt to represent UIs using multiple layers, primarily focused on separating the UI's implementation (actions layer), content (dialog layer), presentation (style rule layer), and interaction (style program layer). ITS allowed the same UI to be presented with multiple styles.

Enhancing the means by which we develop UIs is still an important problem. Yet, the rapid change in the way UIs are developed made such early UIMSs fall victim to the moving targets problem presented by Myers et al. [2000] to indicate that rapid development of technology can make it difficult for tools to keep up the pace.

Another group of systems mainly focused on leveraging MBUID for UI generation. **UIDE** [Foley et al. 1991] and **HUMANOID** [Szekely et al. 1992] focus on automatic UI generation for allowing designers to experiment with different design possibilities before producing the final user interface. **TADEUS** [Elwert and Schlungbaum 1995] provides a methodology with a supporting environment for generating graphical UIs from a model representing the interactive system. **GENIUS** [Janssen et al. 1993] presented a tool supported technique for generating UIs from data models (entity relationship diagrams) and used a model called dialogue net (based on petri nets) as a visual representation of the UI's dynamics. Other systems supporting UI generation

include **JANUS** [Balzert et al. 1996] and **FUSE** [Lonczewski and Schreiber 1996]. JANUS also supported the generation of the code that links the UI to the data.

Most of the early MBUID systems targeting automatic UI generation adopted a simple rule-based approach. One exception was **TRIDENT**, which presented tools for automatically generating interactive business application UIs [Vanderdonckt and Bodart 1993] and a generic architecture model for such applications [Bodart et al. 1995]. It considered more information for UI generation such as ergonomic rules that are represented using a complex hierarchy. Although such rules provide a more sophisticated generation technique, they could be tedious to implement considering their possibly large number (e.g., 3700 rules [Vanderdonckt and Bodart 1996]).

Some systems worked on improving model-based UI representation. **ADEPT** [Markopoulos et al. 1992] is a design environment that aims to incorporate the theory of modeling [Jacob 1986] instead of just creating a fast prototyping tool. **MASTERMIND** [Szekely et al. 1995] is UI development environment complementing HUMANOID and UIDE and focuses on the presentation model. **MECANO** [Puerta 1996] introduced an interface model called MIM and its modeling language MIMIC.

A common shortcoming in early systems (e.g., COUSIN, GENIUS, HUMANOID, UIDE, etc.) was the lack of a high level description of the UI, which was represented in different ways such as: application code (HUMANOID), ER diagrams (GENIUS), etc. Such description was later provided by the second wave of systems such as: ADEPT, MASTERMIND, etc. Yet, it was only at the end of the second generation of MBUID systems that task models were introduced to represent UIs at the highest level of abstraction with notations such as the **ConcurTaskTrees** (CTT) [Paternò et al. 1997]. Other systems such as **MOBI-D** [Puerta and Eisenstein 1998] investigated new techniques for mapping the task models to lower level UI models. Additionally, at this stage technology independent languages such as the User Interface Markup Language (**UIML**) [Abrams et al. 1999] were introduced for defining technology independent UI specifications from which technology specific UIs could be generated.

Developing multi-target UIs was considered at a basic level by this generation of MBUID systems. **AME** [Märtin 1996] offered tool support for the development of interactive systems by constructing UIs from object-oriented analysis models and adapting them to user-specific requirements. **AMULET** [Myers et al. 1997] is a framework aiming at making multi-operating system UI development easier.

Some earlier systems such as ITS indicated the possibility of adapting UIs for different uses such as: display size, resolution, and color-depth. Yet, UI consistency among different applications is more emphasized than adaptation (e.g., GENIUS, COUSIN, etc.). Even though some later systems support UI adaptation to users and environments (e.g., AME using standardized object classes), this support is more oriented towards manual development rather than adaptive behavior. Therefore, we can say that the major shortcoming in the first and second generations of MBUID systems is that they merely use the model-based approach for UI construction rather than take it further for devising adaptive behavior to support multi-context UIs.

3.1.2 Third Generation MBUID Systems. Some domain specific solutions were introduced in this generation such as **Teallach** [Griffiths et al. 2001] that applies the MBUID approach to devise UIs for object databases. However, the major contribution of this generation was a reference framework that provides guidance for model-driven

UI development using multiple levels of abstraction, in addition to the introduction of new UI description languages (UIDLs). Our review does not discuss UIDLs in details because they were covered by a previous survey [Guerrero-Garcia et al. 2009].

CAMELEON [Calvary et al. 2003] is a unified user interface reference framework that is based on two principles [Fonseca 2010]: A Model-based approach, and the coverage of both the design and runtime phases of multi-target user interfaces.
CAMELEON is a seminal research work in this generation of MBUID systems. It provides abstraction guidance for devising UIs based on a model-driven approach. As opposed to conventional UI development techniques that merely construct a concrete level (e.g., graphical buttons, text boxes, etc.), MDE introduces additional levels of abstraction that help in building multi-context user interfaces.

User interfaces are represented in CAMELEON on the following levels of abstraction:

1. Tasks and Domain Models: The task model is the highest level of abstraction that represents UI features as tasks. One possible representation for task models is the **ConcurTaskTrees** [Paternò 1999] notion that allows tasks to be connected with temporal operators. The domain model denotes the application's universe of discourse and can be represented using UML class diagrams. This level of abstraction relates to the *Computation Independent Model* (CIM) in MDA.
2. Abstract User Interface (AUI) Model: This level represents the UI independent of any modality such as: graphical, voice, gesture, etc. The AUI model can be represented using UIDLs such as: **TERESA XML** [Berti et al. 2004], **UsiXML** [Limbourg and Vanderdonckt 2004] and **MARIA** [Paterno' et al. 2009] (4th generation). The AUI relates to the *Platform Independent Model* (PIM) in MDA.
3. Concrete User Interface (CUI) Model: This level is modality dependent. For example, it can represent the UI in terms of graphical widgets such as: buttons, labels, etc. Possible UIDLs for representing concrete user interfaces include: **TERESA XML**, **UIML** [Abrams et al. 1999], **XIML** [Puerta and Eisenstein 2002], **UsiXML**, and **MARIA**. The CUI relates to MDA's *Platform Specific Model* (PSM).
4. Final User Interface (FUI): Represents the actual UI rendered with an existing presentation technology such as: HTML, Windows Forms, WPF, Swing, etc.

CAMELEON is a suitable reference for approaches wishing to adopt model-driven engineering of interactive systems. MDE can provide a basis for devising adaptive UIs since the levels of abstraction presented by CAMELEON allow different types of adaptive behavior to be implemented such as: Using the task model to adapt the feature-set and using the concrete UI model to adapt the layout.

3.2 Reference Architectures for Adaptive Systems

Some software architectures concerned with adaptive system layering can be related to any part (not just the UI) of an adaptive software system. These architectures form a reference for autonomic (self-managing) software systems. We will only give a brief overview of these architectures since more details can be found in an existing survey of autonomic software systems [Huebscher and McCann 2008].

The **MAPE-K** loop was created by IBM as a reference model for autonomic computing [IBM 2006]. MAPE-K considers software systems as a set of managed

resources that could range from an individual application to a more complex cluster. The MAPE-K loop is composed out of four functions with *knowledge* sharing:

— *Monitor*: In this phase, information is collected from the managed resources.

— *Analyze*: Analysis is performed in order to predict any future errors in the system.

— *Plan*: The planning phase prepares the actions required for fulfilling a goal.

— *Execute*: In this phase, the plan is executed and the dynamic updates are applied.

Rainbow is a framework that employs a control loop for managing self-adaptive systems and provides components that fulfill the phases of the MAPE-K loop [Garlan et al. 2004]. Rainbow's architecture layer is made out of the following components:

— The *Model Manager* provides access to the system's architectural model.

— The *Constraint Evaluator* constantly checks the model to see if a constraint has been violated in order to trigger an adaptation.

— The *Adaptation Engine* is responsible for executing the adaptation.

The **Three Layer Architecture** [Kramer and Magee 2007] is an architectural reference for self-managing software systems. It comprises the following layers:

— *Component Control* (bottom layer) is a self-managed set of interconnected components capable of reporting its status to the higher levels.

— *Change Management* (middle layer) executes actions that handle new situations.

— *Goal Management* (highest layer) is responsible for handling time consuming computations that attempt to achieve an outcome relevant to the sought after goal.

Although these architectures do not particularly target UIs, when combined with UI abstraction frameworks such as CAMELEON they could form the basis for an architecture that covers both model-driven UI representation and adaptive behavior.

4. CRITERIA FOR EVALUATING ADAPTIVE MODEL-DRIVEN UI DEVELOPMENT SYSTEMS

In order to conduct a sound and objective critical review of the existing systems, we setup the following criteria, drawing on direct recommendations from the literature and also by combining features from multiple existing systems.

The criteria we established are presented in Table II and each is classified under one of the following five categories: UI development, adaptive behavior development, general development support, engaging stakeholders, and output quality. The existing literature on adaptive model-driven UI development systems is quite diverse but we were able to classify each existing work under one or more of the following categories: architectures, techniques, and tools. Some of the criteria we established are implementation dependent. Hence, it can only be used to evaluate practical UI adaptation techniques or tools, whereas other criteria are also suitable for evaluating reference architectures as well. Therefore, Table II indicates the categories (architecture, technique, and tool) to which each criterion is applicable. Two of the criteria, namely completeness and control over the UI, were established in Section 2 and will be used again since we considered their measure of capability in MDE to be implementation dependent. We should note that we do not claim that our list of criteria is comprehensive. The literature mentions other criteria such as: path of least resistance [Myers et al. 2000] and power in combination [Olsen,Jr. 2007]. However, we needed to limit our list to criteria that we can uniformly apply across the surveyed works using the information that is publicly available.

Table II. Criteria for Evaluating Adaptive Model-Driven UI Development Systems

	Architectures	Techniques	Tools
User Interface Development			
Completeness		X	X
Control over the UI		X	X
Levels of abstraction	X	X	X
Modeling approach	X	X	
Preserving designer input on the UI		X	X
Adaptive Behavior Development			
+Extensibility			
-Adaptation types, aspects, and factors		X	X
-Adaptive behavior	X	X	X
Direct and indirect adaptation	X	X	
Trade-off analysis	X	X	
Visual and code-based adaptive behavior		X	X
Multiple data sources	X	X	
General Development Support			
Modeling, generation, and synchronization			X
IDE style UI			X
+Reducing solution viscosity			
-Flexibility			X
-Expressive leverage			X
-Expressive match			X
Threshold and ceiling			X
Integration in existing systems	X	X	
Engaging Stakeholders			
Empowering new design participants	X	X	X
User feedback on the adapted UI	X	X	
Output Quality			
Scalability		X	

The criteria are listed below and explained. We included one or more of the following codes after each criterion to indicate its applicability to architectures (AR), techniques (TE) and/or tools (TL). These codes reflect the data shown in Table II.

— **Completeness** (*refer to* Section 2) **(TE, TL)**

— **Control over the UI** (*refer to* Section 2) **(TE, TL)**

— Supporting both **direct and indirect adaptation** could make an approach fit for a wider variety of scenarios. User confusion can be reduced by providing the adapted UI as an alternative version (**indirect adaptation**) while maintaining access to the original UI version [McGrenere et al. 2002]. Yet, in some software systems such as ubiquitous applications it may be necessary to adapt the UI while the user is working (**direct adaptation**). One example is MASP, which adapts the UIs of smart home systems based on changes in the environment [Feuerstack et al. 2006]. **(AR, TE)**

— **Extensibility** is considered an important characteristic in any new UI development approach [Demeure et al. 2008]. We refined its meaning as follows:

  — **Extensibility of adaptation types, aspects, and factors** indicates that a UI adaptation approach is not restricted to a single type such as layout optimization but can include a variety of adaptation types such as: feature reduction, navigation help, etc. The approach should also be able to accommodate multiple adaptation aspects (e.g., accessibility, cognition, etc.) and support the adaptation of any UI factor (e.g., level of access to functions, layout grouping, widget type, etc.). **(TE, TL)**

  — **Extensibility of adaptive behavior** is the approach's capability to add new adaptive behavior at runtime as needed to support a variety of aspects and factors.

Contrary to this criterion some approaches might provide limited non-extensible adaptive behavior that is integrated within the system. **(AR, TE, TL)**

— **Empowering new design participants** by introducing new populations to the design process [Olsen,Jr. 2007]: In the case of adaptive user interfaces, new design participants could be non-developers such as: end-users, I.T. personnel, etc. For example, leveraging communities through crowdsourcing could prove useful for applications that require a lot of effort for defining the adaptive behavior. **(AR, TE, TL)**

— An **integrated development environment** (IDE) **style UI** (e.g., similar to Visual Studio or Eclipse) could provide the necessary ease-of-use for managing the UI and adaptive behavior artifacts of large-scale software systems. Developer familiarity and efficiency could be maintained if the tools supporting adaptive model-driven UI development adopt an interface style similar to that of the commercial IDEs. **(TL)**

— An approach that can **integrate in existing systems** without incurring a high integration cost or significantly changing the system could have a higher adoption rate since many systems are at a mature stage in their development life-cycle. Providing a new advance while maintaining legacy code is a good thing [Olsen,Jr. 2007]. **(AR, TE)**

— Supporting multiple **levels of abstraction** as suggested by CAMELEON [Calvary et al. 2003] offers independence of the implementation (task model), modality (abstract UI), and technology (concrete UI). Also, different levels may be more suitable for certain types of adaptation. Features can be reduced by adapting the highest level (e.g., product-line engineering [Pleuss et al. 2010]) and the layout can be optimized using various levels (e.g., graceful degradation [Florins and Vanderdonckt 2004a]). **(AR, TE, TL)**

— The selected **modeling approach** is important. Supporting *interpreted runtime modeling* allows more advanced adaptations to be conducted (*refer to* Section 2.3). Additionally, one of the major drawbacks of generative modeling approaches is that, over time, models may get out of sync with the running code [Coutaz 2010]. **(AR, TE)**

— **Modeling, generation, and synchronization** of all the levels of abstraction: Model-driven UI development tools should offer developers easy-to-use WYSIWYG editors and make transformations transparent to provide a better understanding of their effects [Meixner et al. 2011]. Tool-supported automated approaches must provide **predictability** to the developers using it [Myers et al. 2000], which in this case can be related to supporting WYSIWYG editors and transformation transparency. **(TL)**

— Supporting **multiple data sources** allows adaptations to be carried out in various situations. Adaptive behavior models can embody data based on studies, which is the case of adapting UIs to cultural preferences by MOCCA [Reinecke and Bernstein 2011]. Also, monitoring the user's behavior allows models to evolve and can be beneficial in other situations (e.g., targeting accessibility with MyUI [Peissner et al. 2012]). **(AR, TE)**

— **Preserving designer input on the UI** is important since automated choices without a rationale make adaptive UIs unpredictable. The success of UI development techniques could be negatively impacted by unpredictability [Myers et al. 2000]. UI adaptations will obviously override the input made by the designer. Yet, in some cases designers might want to preserve some characteristics (e.g., prioritizing the size of a widget over others) thereby enhancing the predictability of the outcome. **(TE, TL)**

— **Reducing solution viscosity** is achieved if a tool reduces the effort required to iterate on the possible solutions based on the following criteria [Olsen,Jr. 2007]:

— **Flexibility** denotes the ability to "*make rapid design changes that can then be evaluated by the users*" (p. 255). The tools we are evaluating should be *flexible* by providing the ability to devise both the UI models and the adaptive behavior in a way that allows easy testing and refinement. **(TL)**

— **Expressive leverage** "*is where the designer can accomplish more by expressing less*" (p. 255). We consider that expressive leverage can be achieved by promoting the reusability of UI model parts (e.g., the same way visual-components are reused in traditional IDEs) and the adaptive behavior (e.g., visual-parts or scripts). **(TL)**

— **Expressive match** "*is an estimate of how close the means for expressing design choices are related to the problem being solved*" (p. 255). One way to improve expressive match is by supporting visual-design tools for the UI models and innovative means for specifying *adaptive behavior visually*. **(TL)**

— **Scalability** is an important criterion that must be checked for every new system [Olsen,Jr. 2007]. If the scalability of an adaptation technique is not demonstrated using real-life scenarios its adoption for complex software systems could decrease. **(TE)**

— An ideal tool would have **low threshold and high ceiling** [Myers et al. 2000]. The "threshold" represents the difficulty in learning and using the tool, and the "ceiling" relates to how advanced the tool's outcome can be. **(TL)**

— **Trade-off analysis** between several potentially conflicting adaptations is essential for producing an optimal UI, especially in systems that target multiple adaptation aspects and factors. One example described in the literature is the trade-off between the user's vision and motor abilities [Gajos et al. 2007]. **(AR, TE)**

— **User feedback on the adapted UI** keeps the end-users in the loop of the adaptation process and provides awareness of automated adaptation decisions and the ability to override them when necessary. Creating an underlying representation for users and the automation to communicate is a challenge in human-automation interaction [Miller et al. 2005]. Keeping humans in the loop is considered one of the principles of adapting UIs based on MDE [Balme et al. 2004]. It can increase the end-users' UI control [McGrenere et al. 2002] and feature-awareness [Findlater and McGrenere 2007] affected by adaptive/reduction mechanisms. **(AR, TE)**

— **Visual and code-based** representations allow different stakeholders such as developers and I.T. personnel to implement **adaptive behavior**. Some techniques only support a textual representation such as cascading style sheets in Comet(s) [Demeure et al. 2008] and behavior matrices in FAME [Duarte and Carriço 2006]. Yet, others indicate that a visual notation can greatly simplify the creation of UI adaptation rules by hiding the complexity of programming languages [López-Jaquero et al. 2009]. **(TE, TL)**

We now use these criteria to evaluate the fourth (current) generation adaptive model-driven UI development systems. The reference architectures, practical techniques, and supporting tools are evaluated in Sections 5, 6, and 7 respectively.

5. REFERENCE ARCHITECTURES FOR ADAPTIVE USER INTERFACES

Architectures play a fundamental role in self-adaptive software systems [Oreizy et al. 1999]. An architecture-based approach is promoted for these systems [Kramer and Magee 2007] since it could build on existing work and offer generality, abstraction, scalability, etc. Following a reference architecture could help in realizing adaptive UIs in complex systems. Several architectures have been proposed as a reference for applications targeting adaptive UIs and other UI related features such as distribution, multimodality, etc. This section focuses on evaluating and comparing existing research works related to architectures of adaptive UIs. We briefly describe these architectures and argue their strengths and shortcomings, and conclude by comparing them. This section only evaluates references architectures. Existing adaptive UI techniques, whether based on a defined architecture or not, are discussed and evaluated in Section 6.

A ***3-Layer architecture*** was presented for devising adaptive smart environment user interfaces [Lehmann et al. 2010]. Due to the ubiquitous nature of its target applications, this architecture only supports *direct adaptations*. Information is read from sensors, and the environment context pillar is targeted hence *multiple data sources* are not supported. The *modeling approach* of this architecture is based on generative runtime models, which could be less flexible than interpreted runtime models for performing advanced adaptations. Additionally, the work does not specify whether the architecture is meant to support all the *levels of abstraction*. This architecture does not support *user feedback* but refers to another work [Brdiczka et al. 2007] that does not offer an architecture but uses user-feedback for refining initial situation models at runtime in order to improve the reliability of detected situations.

CAMELEON-RT is a reference architecture model for distributed, migratable, and plastic user interfaces within interactive spaces [Balme et al. 2004]. This architecture targets all context-of-use pillars (user, platform, and environment), and could be considered general-purpose due to its implementation neutrality. We consider that it provides a good conceptual representation of the *extensibility of adaptive* behavior through the use of open-adaptive components [Oreizy et al. 1999], which allow new adaptive behavior to be added at runtime. Also, both *direct and indirect adaptations* could in theory be implemented using these components. It follows the CAMELEON framework hence all the *levels of abstraction* are supported. This architecture depicts observers that collect data on the system, user, platform, and environment, and feed it to a situation synthesizer thereby supporting *multiple data sources*.

CEDAR is a reference architecture for stakeholders interested in developing adaptive enterprise application UIs based on an *interpreted runtime model-driven* approach [Akiki et al. 2012]. This architecture follows the *levels of abstraction* suggested by CAMELEON for representing its UI models. It supports both *direct and indirect adaptation* and the *extensibility of its adaptive behavior*, which is stored in a relational database. CEDAR presents components for supporting *trade-off analysis* and *user feedback* on the UI adaptations. Furthermore, CEDAR was evaluated by *integrating it into an existing enterprise application* called OFBiz [Akiki et al. 2014]. It also introduced a basic crowdsourcing approach for *empowering end-users* to participate in the UI adaptation process [Akiki et al. 2013b].

FAME is an architecture that targets adaptive multimodal UIs using a set of context models in combination with user inputs [Duarte and Carriço 2006]. It only targets modality adaptation hence it is not meant to be a general-purpose reference for adapting other UI characteristics. The adopted approach allows designer input on the CUI hence providing good *control over the UI*. *Adaptive behavior* can be *extended* using adaptive behavior matrices. FAME depicts support for *multiple data-sources* including device changes, environmental changes, and user inputs that feed into related models. A combination of the multiple data sources and the adaptive behavior matrices should be able to support both *direct and indirect adaptations*.

Malai is an architectural model for interactive systems [Blouin and Beaudoux 2010] and forms a basis for a technique that uses aspect-oriented modeling (**AOM**) for adapting user interfaces [Blouin et al. 2011]. The *extensibility of adaptive behavior* is poor since multiple presentations have to be defined at design-time by the

developer, to be later switched at runtime. Although Malai supports multiple *levels of abstraction*, the *modeling approach* relies on generating code (e.g., Swing, .NET, etc.) to represent the UI. Also, it does not describe *multiple sources* for acquiring adaptive behavior data. In theory, both *direct and indirect adaptations* can be supported. MALAI allows developers to define *feedback* that would help users to understand the state of the interactive system but the user cannot provide feedback on the adaptations (e.g., reverse an unwanted adaptation).

We noticed that several criteria were not addressed by most of the works reviewed in this section. For example, only CEDAR presented components for managing *trade-off analysis* and *user feedback*. In spite of the importance of *integration in existing software systems* that are in a mature development stage, asides from CEDAR, the evaluations were conducted by building new prototypes. Furthermore, *empowering new design participants* was only partially addressed by CEDAR while the other architectures did not incorporate any components for supporting this feature. After arguing the strengths and limitations of existing adaptive UI architectures, we present a visual evaluation and comparison in Table III that illustrates the extent to which each of the architectures fulfills the criteria we established in Section 4.

Table III. Visual Evaluation and Comparison of Adaptive Model-Driven UI Development Reference Architectures

Legend ● Completely fulfills ◖ Partially fulfills ○ Does not fulfill ◌ Not specified	**Direct and indirect adaptation**	**Empowering new design participants**	**Extensibility of adaptive behavior**	**Integrating in existing systems**	**Levels of abstraction**	**Modeling approach**	**Multiple data sources**	**Trade-off analysis**	**User feedback on the adapted UI**
3-Layer Architecture	◖	○	○	○	◌	◖	◖	○	○
CAMELEON-RT	●	○	●	○	●	◌	●	○	○
CEDAR	●	◖	●	●	●	●	●	●	●
FAME	●	○	●	○	◖	◌	●	○	○
MALAI	●	○	◖	○	●	◖	◖	○	○

6. TECHNIQUES FOR DEVISING ADAPTIVE MODEL-DRIVEN USER INTERFACES

Adaptive behavior can target a variety of UI characteristics. In order to provide a boundary for this work, we shall focus on the techniques that focus on at least one of the two UI adaptation types that are the most targeted in the literature, namely feature-set adaptation and layout optimization. We define a feature as a functionality of a software system and a minimal feature-set as the set with the least features required by a user to perform a job. An optimal layout is the one that maximizes the satisfaction of constraints imposed by certain adaptation aspects. An optimal layout is achieved by adapting the properties of concrete widgets such as: type, size, etc.

6.1 Feature-Set Adaptation Techniques

The functionality of software applications tends to increase with every release hence increasing the visual complexity. This phenomenon, referred to as "bloatware"

[McGrenere 2000], has a negative impact on usability especially for users who do not require the complete feature-set. It could be helpful to provide each end-user with a minimal feature-set that reduces unnecessary bloat present in feature-rich UIs. Since the existing solutions that are related to UI bloat, expect RBUIS [Akiki et al. 2013d], focus on design-time adaptation rather than runtime adaptive behavior, we did not evaluate them according to the criteria established in Section 4. Instead, we grouped them into categories and evaluated their strengths and shortcomings in general.

Several ***theoretical propositions*** were made for reducing a UI's feature-set based on the context-of-use. Providing a multi-layered user interface design is promoted for achieving *universal usability* [Shneiderman 2003]. Other researchers propose using *two UI versions*, one fully-featured and another personalized, for taming the bloat in feature-rich applications [McGrenere et al. 2002]. An early research work proposes the use of a *training wheels* UI that blocks advanced functionality from novice users [Carroll and Carrithers 1984]. These works present a sound theoretical basis, useful for providing the users of feature-bloated software applications with a minimal feature-set. Yet, the given examples, a basic text editor [Shneiderman 2003] and the Word 2000 menu [McGrenere et al. 2002], do not match the complexity of large-scale systems such as enterprise applications. Also, these works do not provide a technical implementation.

Approaches from ***software product-line*** (SPL) engineering [Pleuss et al. 2010] are used to tailor software applications and some particularly address tailoring user interfaces. *MANTRA* [Botterweck 2011] adapts UIs to multiple platforms by generating code particular to each platform from an abstract UI model. Although SPLs can be dynamic [Bencomo et al. 2008], the SPL-based approaches for UI adaptation focus on design-time (product-based) adaptation whereas runtime (role-based) adaptive behavior is not addressed.

Several ***commercial software applications*** use role-based tailoring of the UI's feature-set. *Microsoft Dynamics CRM* [Microsoft 2011] and *SAP's GuiXT* [Synactive GmbH 2010] offer such a mechanism, yet it is not generic enough to be used with other applications and it requires developing and maintaining multiple UI copies manually. An approach that operates at the model level could be more general-purpose.

Some approaches relied on ***decomposition*** to break the UIs into smaller fragments that fit the context-of-use better. *Graceful degradation* is used as a method for supporting UIs on multiple devices [Florins and Vanderdonckt 2004a] and could be used for decomposing/recomposing UIs. An initial UI is constructed for the platform with the least constraints, and then other versions are generated for the platforms with more constraints based on designer annotations. In concept, this method could be used for minimizing a UI's feature-set by decomposing it into smaller fragments. Yet, its main limitation lies in its reliance on designer annotations that would not work when the adaptations are only known at runtime. An interesting approach would be to support runtime annotations combined with automated procedures that can adapt the UI based on each user's behavior. Another approach called *(de)composition* complements some aspects of graceful degradation [Lepreux et al. 2007]. It aims towards supporting reusability at a high-level design without the need for applying constant copy/paste operations. Similar to the graceful

degradation approach, (de)composition could be in concept used for reducing the UI's feature-set. The authors mention the applicability of this approach both at design-time and runtime but no significant demonstration is made towards runtime scenarios since all the examples were restricted to design-time. Decomposing/composing UIs at runtime while maintaining functionality would require work that does not merely adjust the UI's layout but maintains and adapts the functionality behind it.

The main limitations in approaches attempting to target feature-set adaptation are: lack of a practical implementation mechanism, lack of generality of the solutions, or restriction to design-time adaptation without offering a runtime adaptive solution. Based on these limitations, we saw that more work is needed to provide a general-purpose, model-driven, and tool supported adaptive UI mechanism capable of reducing UI bloat at runtime by adapting the UI's feature-set based on the context-of-use. Therefore, we introduced Role-Based UI Simplification (**RBUIS**) [Akiki et al. 2013d] as a mechanism for improving usability by providing users with a minimal feature-set and an optimal layout based on the context-of-use. Since RBUIS also supports layout optimization, we shall discuss it in Section 6.2 and evaluate it according to the criteria established in Section 4.

6.2 Layout Optimization Techniques

Providing an optimal layout based on the context-of-use could improve usability by catering to the diverse end-user needs. For example, the usability of SAP (world's leading ERP system [Jacobson et al. 2007]) is mostly affected by navigation and presentation [Singh and Wesson 2009] and its UI does not adapt to each end-user's skills [Uflacker and Busse 2007]. Existing works use different approaches for adapting the UI layout. In this sub-section, we provide a brief description of each of these works and argue their strengths and shortcomings using the criteria we established in Section 4.

The COntext Mouldable widgeT (**Comet(s)**) was introduced as a set of widgets that support UI plasticity [Calvary et al. 2005a]. It provides an architectural-style for plastic UIs by combining the toolkit and model-based approaches [Demeure et al. 2008]. A "Comet" is capable of self-adapting or being adapted to the context-of-use.

Comet(s) target the adaptation of individual widgets but does not focus on the entire layout. Centralizing the adaptive mechanism could be more *scalable* than defining it in each widget and could make Comets a more interesting solution for adaptive UI functionality. Using a widget toolkit to represent the CUI provides good *control over the UI* and could theoretically be used to develop different types of UIs that can adapt to any context pillar therefore providing good *completeness*. The e*xtensibility of the adaptive behavior* is claimed to be supported through style-sheets but the *adaptation types* are not extensible since each Comet can only adjust its own shape, whereas different types of adaptation (e.g., feature-set, navigation, etc.), which might be more related to the overall user interface design, cannot be supported by this architectural-style. One of the goals of Comet(s) is to sustain UI adaptation at any level of abstraction: tasks and concepts, abstract, concrete, and final UI as elicited in model-driven approaches [Calvary et al. 2003]. Therefore, the *levels of abstraction* are embodied in what are referred to as the Logical Consistency (LC), Logical Model (LM), Physical Model (PM), and technology primitives. Comet(s) does not present a means for reading adaptation data from *multiple data sources* as presented by CAMELEON-RT for example. We consider the *modeling approach* to be

poor since it is necessary to have a code-based implementation as opposed to the possibility of using interpreted runtime models. Although it is not explicitly described, the use of style-sheets can support both *direct and indirect adaptations*.

DYNAmic MOdel-bAsed user Interface Development (**DynaMo-AID**) is a design process and runtime architecture for devising context-aware UIs and is part of the Dygimes UI framework [Coninx et al. 2003]. Its runtime architecture includes three major modules namely context monitoring, functional core, and presentation that are linked by a dialog controller [Clerckx et al. 2005]. The final UI is rendered from task models after adapting them to the operating environment and device.

DynaMo-AID is limited to WIMP-style UIs and only targets the environment context hence has low *completeness*. The *levels of abstraction* supported by DynaMo-AID are restricted to the task model from which the final UI is generated. The support of interpreted runtime models provides a good *modeling approach* but since designer input is not supported on the CUI the *control over the UI* could be negatively affected. *Adaptive behavior* is extensible but is restricted to one *type of adaptation*, namely, the UI dialog. Due to the pervasive nature of its target applications (e.g., tourist guide mobile application called Imogl for an open air museum [Clerckx et al. 2006]), DynaMo-AID only supports *direct adaptation*s and environment sensors as a *data source*. This architecture is particularly criticized for using what is referred to as a "Task Tree Forest" [Blouin et al. 2011]. The critics note that since each tree corresponds to the tasks possible in a given context, the combinatorial explosion would affect the approach's *scalability* when it is applied to complex systems.

Supple supports automatic generation of UIs adapted to each user's abilities (e.g., motor and vision), devices, tasks, and preferences [Gajos et al. 2010]. It relies on a high-level interface specification, device model, and user traces to generate the UI.

In terms of *completeness* Supple's approach is particularly well suited for box-like UIs due to the existence of a vocabulary of interactions for this UI type. However, although not tested, its creators indicate that it is not limited to such UI types especially if new vocabularies of interactions could be identified [Gajos et al. 2010]. Supple *interprets and renders UI models* at runtime hence making the fulfillment of more advanced adaptations easier. Yet, the adopted technique generates the UI from a high-level model (one *level of abstraction*), which prevents designer input from being made especially at the CUI level hence provides less *control over the UI*. The inability to have human input at the different levels of abstraction, at least at design-time, makes the approach difficult to adopt for large-scale systems such as enterprise applications. Supple has built-in algorithms for adapting the UI and does not provide a means for *extending the adaptive behavior* through either a v*isual or code-based* representation. The only *adaptation type* supported by Supple is layout optimization. Vision and motor capabilities are the primary supported *adaptation aspects*, and 40 UI factors (e.g., font size, widget size, etc.) are supported. Supple does not provide a means for *extending adaptation types, aspects, and factors*. Also, it has been criticized [Peissner et al. 2012] for exceeding acceptable performance times. This criticism could be justified by observing some of its worst-case scenarios that could span over 30 seconds when computing the most appropriate UI layout. This timing is not appropriate for software systems looking for high efficiency. One advantage that Supple has over other systems lies in performing true layout optimization due to its ability to quantify UI quality. The quantification is achieved by using a cost function

to compare UI versions in order to determine the most optimal one. This approach also allows Supple to support *trade-off analysis*, which was demonstrated for a fixed number of adaptation aspects, namely motor and vision capabilities [Gajos et al. 2007]. Supple is complemented by a system called **Arnauld** [Gajos and Weld 2005], which is responsible for eliciting user preferences in order to adapt the UI at runtime. This process could serve as a *feedback mechanism* but the sole reliance on runtime elicitation can be time consuming and might not provide sufficient data in comparison to leveraging *multiple data sources*. Supple primarily targets *indirect adaptation* since it builds up a user-model over time based on preference elicitation.

The Multi-Access Service Platform (**MASP**) is a UI management system targeting ubiquitous UIs for smart homes [Feuerstack et al. 2006]. MASP uses a model-driven approach to support: multimodality [Blumendorf et al. 2008], distribution [Blumendorf et al. 2007], synchronization [Blumendorf et al. 2006], and adaptation [Schwartze et al. 2009]. Although it demonstrates powerful capabilities in UI distribution and multimodality, we focus on its adaptation capabilities to stay within our scope.

Adopting a box-based layout [Feuerstack et al. 2008] for repositioning different UI segments at runtime using content scaling prevents widget level feature-adaptation and decreases *completeness*. The *modeling approach* bases the final UI on generated code or markup (apache velocity templates) [Blumendorf et al. 2008] hence allowing less advanced adaptations to be performed at runtime as opposed to a fully-dynamic approach. MASP uses *direct adaptation* whenever a context change is detected due to the ubiquitous nature of the target smart home systems. The primary *adaptation type* supported by MASP is layouting based on several environment-related *aspects* such as distance and spots (e.g., distance from particular physical objects). Also, a limited number of UI adaptation factors are supported (e.g., orientation, size, rearrangement of predefined UI groups, etc.) and no means is provided *for extending the adaptation types, aspects, and factors*. As described in its UI construction technique [Feuerstack 2008], MASP supports all the *levels of abstraction* suggested by CAMELEON. It also supports designer input on the CUI to provide *control over the UI*. Since it targets ubiquitous applications, MASP's *data sources* are restricted to environment sensors as indicated by the 3-Layer architecture [Lehmann et al. 2010]. MASP provides a tool to visually divide the layout into boxes but does not support the definition of *visual and code-based adaptation rules*, which could cover a variety of layout optimization factors that go beyond changing the font-size, and layout grouping.

One technique uses aspect-oriented modeling (**AOM**) for adapting UIs [Blouin et al. 2011] based on the **MALAI** architecture (reviewed in Section 5).

This approach requires several UI presentations to be defined at design-time and a weaver is used to associate these presentations to instrument classes that handle the way the UI functions at runtime. It provides *completeness* because it targets post-WIMP UIs and could logically target others as well since the UI is generated to code, hence also providing good *control over the UI.* The *adaptive behavior* could be *extended* but this can only be done at design-time since the *modeling approach* relies on code. Hence, the UI variations have to be manually defined by the developer. *Scalability* is demonstrated by taming the combinatorial explosion of complex interactive system adaptations. The meta-model does not support the addition of *adaptation types, aspects, and factors*. Also, no mechanism is provided for adding *adaptive behavior visually*.

MyUI is a user interface development infrastructure for improving accessibility through adaptive UIs [Peissner et al. 2012]. It uses an open pattern repository for defining adaptation rules. User interfaces are specified as an abstract model that is represented using a notation based on state charts.

MyUI is presented as a *general-purpose* infrastructure but it was only demonstrated with basic interactive television UIs. It does not support all the *levels of abstraction* suggested by CAMELEON but only relies on an abstract model to automatically generate the final UI. Hence, the designer's *control over the UI* is reduced due to the lack of designer input on the concrete UI. MyUI supports both *direct and indirect adaptations* since the users can swipe a card to let the system identify who they are and customize the UI based on their profile; sensors are also able to detect gestures (e.g., leaning towards the screen due to poor vision) and perform direct UI adaptations accordingly. It is possible to *extend the adaptive behavior* by modifying the state-chart models; however this extension is performed at development-time and could require a redeployment of the application. MyUI shows the possibility of supporting *multiple adaptation data sources* since the patterns it uses for defining the adaptation rules can embody expert knowledge and the system has the ability to acquire environment data using sensors. Although MyUI allows the end-users to reverse the adaptations, its *feedback* mechanism can be enhanced further by offering users an explanation of the reason behind the adaptation. The *adaptive behavior* (adaptation rules) in MyUI are *defined visually* using a state-chart model; however the basic accessibility adaptation examples (e.g., changing font-size) that were presented do not demonstrate whether the state-chart notation has the potential for defining more advanced usability related adaptations such as changing the layout grouping, widget types, etc.

Role-Based UI Simplification (**RBUIS**) [Akiki et al. 2013d] is a mechanism for improving the usability of enterprise application UIs by providing users with a minimal feature-set and an optimal layout based on the context-of-use.

RBUIS is based on the **CEDAR** architecture. Therefore, it supports *direct and indirect adaptation* and adopts interpreted runtime models as a *modeling approach*, supports the *levels of abstraction* suggested by CAMELEON, and can be *integrated into existing systems*. RBUIS supports *extensible visual and code-based adaptive behavior*. For example, layout optimization adaptive behavior can be represented by dynamic workflows that can incorporate both visual and code-based programming constructs. RBUIS implements the trade-off analysis and user-feedback components proposed by CEDAR. It also supports *the extension of adaptation aspect and factors* using goal models. The *scalability* of the algorithms behind RBUIS was evaluated using both a complexity analysis and runtime load-testing. RBUIS also partially supports the *preservation of designer input* on the UI using constraints [Akiki et al. 2013c] and the *empowerment of end-users* for participating in the adaptation process through the means of crowdsourcing [Akiki et al. 2013b].

We noticed that several criteria were not addressed by the majority of the works reviewed in this section. For example, *preserving designer input on the UI* after the adaptive behavior has been applied has only been partially addressed by RBUIS. The same applies for *empowering new design participants* such as engaging and leveraging end-user communities to participate in UI adaptation through the means of crowdsourcing. Aside from Supple and RBUIS, most techniques do not offer any insight on managing *trade-offs* between possibly conflicting adaptations. Supple,

MyUI, and RBUIS support, to different extents, *user feedback on the adapted UI*. Comet(s) should in concept allow end-users to explore possible design alternatives but this point was left for future work. Aside from RBUIS, the techniques we reviewed were evaluated by developing new prototype systems instead of showing the ability to *integrate in existing software systems*. For example, MASP was evaluated by (re)building home automation applications such as: energy, cooking, and health assistants; Supple was evaluated by developing a variety of simple UI dialogs such as: email client, ribbon and print dialogs, etc. An *extensible number of aspects and factors* is only supported by RBUIS using goal models, whereas other systems merely supported a limited number them. For example, Supple supports 40 factors and targets a limited number of adaptation aspects related to physical impairments. We noticed that very few works conducted *scalability* tests, which are important to demonstrate if the technique works with large-scale and complex UIs. We present a visual evaluation and comparison of the layout optimization techniques in Table III.

Table IV. Visual Evaluation and Comparison of Adaptive Model-Driven Layout Optimization Techniques

Legend ● Completely fulfills ◖ Partially fulfills ○ Does not fulfill ◌ Not specified	**Completeness**	**Control over the UI**	**Direct and indirect adaptation**	**Extensibility of adaptive behavior**	**Extensibility of adaptation types, aspect, factors**	**Empowering new design participants**	**Integrating in existing systems**	**Levels of abstraction**	**Modeling approach**	**Multiple data sources**	**Preserving designer input**	**Scalability**	**Trade-off analysis**	**User feedback on the adapted UI**	**Visual and code-based adaptive behavior**
AOM (MALAI)	●	●	●	◖	○	○	○	●	◖	◖	○	●	○	○	◖
Comet(s)	●	●	●	●	○	○	○	●	◖	◖	○	◌	○	○	◖
DynaMo-AID	◖	◖	◖	●	○	○	○	◖	●	◖	○	◖	○	○	◖
Supple	◖	◖	●	○	○	○	○	◖	●	◖	○	◖	◖	●	○
MASP	◖	●	◖	●	○	○	○	●	◖	◖	○	◌	○	○	◖
MyUI	◖	◖	●	◖	○	○	○	◖	◖	●	○	◌	○	◖	◖
RBUIS	◖	●	●	●	●	◖	●	●	●	●	◖	●	●	●	●

7. TOOLS SUPPORTING ADAPTIVE MODEL-DRIVEN USER INTERFACE DEVELOPMENT

The adoption of a technique depends largely on giving researchers and practitioners the means of applying ideas without resorting to low level implementation [Cheng et al. 2009]. The model-driven approach to UI development can serve as a basis for devising adaptive UIs due to the possibility of applying different types of adaptations onto various levels of abstraction. Yet, implementing adaptive model-driven UIs requires the tools that support a definition of the necessary UI models and adaptive behavior. In this regard, existing tools still lack many features required for supporting adaptive model-driven UIs. This section provides an overview of the state-of-the-art tools for developing (adaptive) model-driven UIs and evaluates them according to their support of the criteria established in Section 4. The evaluation is based on the published research work, together with demonstration videos when available, and the associated tools publicly available.

A survey on model-driven engineering tools for developing UIs [Pérez-Medina et al. 2007] covered many existing tools including: ACCELEO, AndroMDA, ADT, AToM3, DSL Tools, Kermeta, ModFact, Merlin, MDA Workbench, MOFLON, OptimalJ, QVT Partners, SmartQVT, and UMLX. One of the conclusions made was that these tools are centered on MOF hence support the creation of domain-specific models. However, since these tools are not directly meant for supporting (adaptive) model-driven UIs we shall not consider them as part of our review.

There are some commercial tools that partially support MDE in UI development. However, these tools were not intended for developing adaptive UIs. **Leonardi** is one example; it provides free (www.leonardi-free.org) and commercial (www.w4.eu) versions of its application composer. This composer allows designers to visually define the UIs that could be interpreted at runtime. The creators of Leonardi (W4) specify three challenges for business applications: offering high quality user experience, developing software at low cost with minimum technical experience, and providing scalable applications that can accommodate constant business and technological changes.

They claim to face these challenges by supporting MDE agile in Leonardi. This is practically achieved by not generating code from the UI design. Instead, the UI is *interpreted at runtime* through an application engine. We should note that MDE agile is plausible but we noticed some limitations in the way it is applied in Leonardi. Since it is a rapid application development (RAD) tool, Leonardi only supports the concrete UI model and ignores the other *levels of abstraction*. Also, this tool is coupled to a certain extent with Java and does not provide specifications for developing application engines for other technologies. Leonardi is *not intended for developing adaptive UIs* hence it does not offer any tool support for adaptive UI behavior. Other frameworks and tools with fewer features such as **OpenXava** (www.openxava.org) and **Himalia** (www.bit.ly/HimaliaDotNet) provide different model-driven approaches for developing UIs. Yet, the tight coupling of these tools with programming languages (e.g., Java, .NET, etc.) discourages their adoption as a general purpose solution.

Supple [Gajos et al. 2010] partially adopts model-driven UI development hence its tools do not support all the *levels of abstraction*. Basic information on the supporting tools is available in the published work but the tools are not available for the public. According to Gajos (http://bit.ly/SuppleSystem), Supple is still a research prototype and he hopes it could be made available for the public in the future.

Cedar Studio [Akiki et al. 2013a] is an IDE for supporting technical stakeholders such as software developers and I.T. personnel in developing and maintaining adaptive model-driven enterprise application UIs using the RBUIS [Akiki et al. 2013d] mechanism based on the CEDAR [Akiki et al. 2012] architecture. This IDE offers *control over the UI* by allowing stakeholders to provide their input on *all the levels of abstraction* using visual design tools. Cedar Studio also supports *visual-design and code-editing tools* for defining *extensible adaptive behavior and adaptation aspects and factors*. Its integrated testing, visual-design tools, and *IDE style UI* allow Cedar Studio to offer good *flexibility*. This IDE supports *automatic generation* between the UI models and the mapping rules can be visually changed to offer better *predictability*.

The **ConcurTaskTrees Environment (CTTE)** [Mori et al. 2002] (version 2.6.3) is a tool for developing and analyzing task models using the CTT notation. CTTE provides a mature UI for designers to devise task models. Yet, it does not provide

visual-design tools for all the *levels of abstraction* but it supports the *generation* of the AUI and CUI models in the **MARIA** [Paterno' et al. 2009] language from the CTT task model. The CUI model can be generated as a desktop, mobile, or vocal UI and the final UI can be generated to HTML or Voice XML. However, *synchronization* is not supported in case the CUI changes; also, the generation rules cannot be modified from the tool hence providing a low *predictability* of the generated CUI. MARIA also has a separate authoring environment to separate several *levels of abstraction*. Users can define transformation rules to map the AUI models to CUI models. These rules can be defined through a *visual mapping* between the AUI and CUI model elements. The ability to do so provides a better *predictability* of the generated outcome.

Several tools were presented for supporting the UIDL UsiXML [Limbourg and Vanderdonckt 2004]. Some of these tools such as **UsiComp** [García Frey et al. 2012] and **Xplain** [García Frey et al. 2010] are early-stage research prototypes that provide a limited number of features. The UI models representing the different levels of abstraction are designed in UsiComp inside a single document-style panel. This approach negatively affects the tool's *flexibility* and could prove to be tedious when designing UIs for large-scale complex systems. A multi-document *IDE style UI* could be more helpful for developers and I.T. personnel in managing a large number of artifacts (e.g., UI models, code files, etc.) in real-life software projects.

Similar tools such as: **SketchiXML** [Coyette and Vanderdonckt 2005], **IdealXML** [Montero and López-Jaquero 2007] and **GraphiXML** [Michotte and Vanderdonckt 2008], only target specific phases of the UI construction process hence do not support all the *levels of abstraction*. SketchiXML focuses on transforming manually drawn sketches to concrete UI models. This tool can generate a *predictable* CUI model from the drawn sketches especially if predefined widget sketches were loaded into the system. IdealXML is concerned with modeling task models and generating the abstract UI from them. GraphiXML provides a graphical-design tool for concrete UIs. Even though these tools do not support all the *levels of abstraction*, we consider that they provide a good *control over the UI* since the designer can manipulate the supported models.

Although it is still a limited prototype, UsiComp is the only one of these tools that supports all the *levels of abstraction* and directly targets UI adaptation by applying rules written in the Atlas Transformation Language (ATL) to the UI models. A demonstration showed how these rules could adapt the same UI models to different platforms (web and mobile). The *extensibility of the adaptive behavior* is limited since no clear demonstration is given on how these rules can be extended using the tool. A *visual representation* of these rules is not supported, and the automated *generation and synchronization* between the different levels of abstraction was not demonstrated.

A few supporting tools were presented as part of **MASP** [Feuerstack et al. 2008] including: a *task tree editor* as an Eclipse plugin, in addition to a *layouting tool* and a *task tree simulator* that were offered as standalone tools. The task tree editor can be used in order to model the various tasks, which are required to be supported by a segment of the application being developed. The layouting tool is used for generating layout models. This tool is provided with design models and context-of-use scenarios (device properties and user preferences) as input. The tool provides a box-based layout allowing the designer to specify properties related to containment, order, orientation, and size. However, MASP lacks a canvas-style visual-design tool for concrete UIs; this could have a negative impact on its *flexibility*. Feuerstack et al.

[2006] suggested an HTML WYSIWYG editor to alter the UI; this could prove less useful than a technology independent concrete UI editing tool when targeting multiple presentation technologies. Also, MASP's tools only support adding basic adaptations that are applied to a layout with predefined box-based groupings.

Gummy supports multi-platform graphical UI development [Meskens et al. 2008]. It can generate an initial design for each new platform using a combination of features from existing user interfaces. A key objective of Gummy is providing an environment that resembles traditional GUI development tools in order to allow designers to target new platforms without giving up their current work practices. Additionally, Gummy hides the high *levels of abstraction* from the designers, thereby allowing them to operate on the level of abstraction that they are the most familiar with, namely the CUI. Having such designer input on the CUI provides more *control over the UIs*. *Predictable generation and synchronization* is less required because Gummy hides the upper levels of abstraction from the designers. However, some characteristics of the high-level models, such as the temporal operations on CTT tasks models, are not easy to deduce from the CUI. Therefore, it is our belief that it would be better to expose these upper-level models for advanced designers who choose to modify them.

Damask [Lin and Landay 2008] is a tool for prototyping cross-device UIs. It is not intended for developing adaptive UIs but for supporting design-time automatic UI generation. Designers can define a UI layout for one device from which Damask can create an abstract model that it uses to generate the UI layouts for other devices. The employment of patterns in designing the initial UI helps Damask in *generating* more *predictable* UIs for other devices. The designers can refine the generated UI layouts if necessary, hence providing good *control over the UI*. Both web-style and voice UIs are supported for PCs and mobile phones.

Several criteria were not addressed by most of the surveyed tools. Some tools are intended for developing model-driven UIs but do not support adaptation capabilities. Hence, the extensibility of the adaptive behavior and the definition of visual and code-based adaptive behavior are only supported fully by Cedar Studio and partially by MASP, UsiComp, and MARIA. Also, apart from Cedar Studio and Leonardi, the tools do not provide a mature IDE style UI for easing the development process.

In spite of the heterogeneity in the types of platforms (e.g., desktop, web, mobile) for which UIs can be generated by some tools such as Damask and MARIA, the generated UIs only follow the WIMP style. Therefore, the tools we surveyed do not demonstrate a high level of *completeness* since the ability of the tools to support other UI types such as multi-touch tabletop UIs was not demonstrated.

Most of the tools we surveyed provide a visual-design canvas for the models they support. Therefore, we considered that they fulfill the *expressive match* criterion. We considered tools such as: Cedar Studio, Damask, Gummy, and Leonardi, which support a form of integrated testing to fulfill the *flexibility* criterion.

Besides Leonardi, the surveyed tools do not support reusability of UI model parts (e.g., the way visual components are reused in traditional IDEs). Also, besides Cedar Studio, adaptive behavior reusability is not demonstrated but in principle it could be possible in MARIA, MASP, and UsiComp, which support transformation rules. Hence, we consider Cedar Studio and Leonardi to fulfill the *expressive leverage* criterion.

Achieving a low *threshold* and a high *ceiling* is considered a major criticism regarding tools supporting model-driven UI development. Therefore, building models graphically was suggested to achieve a lower threshold [Vanderdonckt 2008]. We can say that Cedar Studio, Damask, Gummy, and Leonardi potentially have a lower threshold than other tools since they promote a development technique that could start with the CUI similar to the techniques adopted by classic IDEs, which are more familiar to designers. In terms of achieving a high ceiling, since most of the tools are research prototypes, it is hard to consider them as alternatives for commercial IDEs that can be used to develop real-life software applications. One exception is Leonardi, which is a commercial IDE but it does not support adaptive behavior. Hence, we considered the surveyed tools to partially fulfill the *threshold and ceiling* criterion.

We present a visual evaluation and comparison of the tools we reviewed in Table V.

Table V. Visual Evaluation and Comparison of Adaptive Model-Driven UI Development Tools

Legend ● Completely fulfills ◖ Partially fulfills ○ Does not fulfill	**Completeness**	**Control over the UI**	**Extensibility of adaptive behavior**	**Extensibility of adaptation types, aspect, factors**	**Empowering new design participants**	**IDE style UI**	**Levels of abstraction**	**Predictable modeling, generation, and sync.**	**Preserving designer input**	**Flexibility**	**Expressive leverage**	**Expressive match**	**Threshold and ceiling**	**Visual and code-based adaptive behavior**
CTTE (HIIS Lab)	◖	●	○	○	○	◖	◖	◖	○	◖	○	●	◖	○
Cedar Studio	◖	●	●	●	◖	●	●	●	◖	●	●	●	◖	●
Damask	◖	●	○	○	○	◖	◖	●	○	●	○	●	◖	○
GraphiXML	◖	●	○	○	○	◖	◖	○	○	◖	○	●	◖	○
Gummy	◖	●	○	○	○	◖	●	●	○	●	○	●	◖	○
IdealXML	◖	●	○	○	○	○	◖	◖	○	◖	○	●	◖	○
Leonardi	◖	●	○	○	○	●	◖	○	○	●	●	●	◖	○
MARIA (Tools)	◖	●	◖	○	○	◖	●	●	○	◖	◖	●	◖	◖
MASP (Tools)	◖	●	◖	○	○	◖	◖	◖	○	◖	◖	●	◖	◖
SketchiXML	◖	●	○	○	○	○	◖	●	○	◖	○	●	◖	○
UsiComp	◖	●	◖	○	○	○	●	◖	○	◖	◖	●	◖	◖

8. CONCLUSIONS AND FUTURE OUTLOOK

The existing literature presents several systems for developing adaptive model-driven UIs in an attempt to address some software usability problems. This review presented an overview and evaluation of these systems. We established a set of criteria by either directly drawing on recommendations from the literature or indirectly from features dispersed in multiple existing systems. We classified the state-of-the-art systems under: reference architectures, practical techniques, and supporting tools, and evaluated them according to the criteria we established. Based on this evaluation, we draw our conclusions and highlight the open issues that can be solved by future research.

We noticed that there is room for improving upon several criteria in the existing state-of-the-art architectures, techniques, and tools. We group the criteria that can be improved upon into three categories: **end-users**, **user interface designers**, and

technical characteristics. In the following paragraphs, we list these criteria, highlight their importance, and suggest some directions for future research.

End-users: *Empowering new design participants*, such as end-users, to participate in the UI adaptation process could help in engaging these stakeholders to increase their understanding and acceptance and in reducing the time needed to define the adaptive behavior. Many end-users are power-users who understand how the system works and could provide a valuable contribution towards adapting it to the different contexts-of-use.

A few approaches tried to empower end-users to participate in the UI adaptation process through the means of crowdsourcing. Adaptable Gimp [Lafreniere et al. 2011] was presented as a socially adaptable alternative of the GNU image manipulation tool Gimp. It allows the community to customize its UI by creating task-sets in a wiki. Another approach [Nebeling et al. 2012] allows HTML-based UIs to be adapted by users through a toolkit with a predefined set of adaptations. The changes are stored in a central repository as Cascading Style Sheets (CSS), which could be applied for other users with similar needs. However, these approaches are not model-driven hence making them technology-dependent and only focus on end-user manual adaptation. We presented an approach that complements our RBUIS mechanism by engaging end-users to help technical stakeholders in defining the adaptive UI behavior using a simple online tool [Akiki et al. 2013b]. Yet, more work is still required in this area for creating advanced easy-to-use tools, which expose the adaptation techniques to non-technical end-users and allow them to define adaptive behavior. Additionally, conducting elaborate empirical studies would help in testing how well these tools work with real-life scenarios.

User interface designers: In certain cases, UI designers might want to keep some UI characteristics intact even after the adaption occurs. For example, they might want to set a minimum size for a textbox based on their knowledge of the nature of the business that the UI is supporting. Since such human ingenuity is hard to replace with automated behavior, in some particular cases, *preserving designer input on the UI* is important for obtaining a more predictable post-adaptation outcome.

Some approaches attempted to address this criterion either directly or indirectly. Smart templates were proposed for improving automatic generation of ubiquitous remote control UIs on mobile devices [Nichols et al. 2004]. Although these templates improve the ability to preserve designer input, specifying the various template variations could be time consuming and would be classified under adaptable rather than adaptive behavior. Raneburger et al. [2012], attempt to enhance the quality of generated UIs by using a graphical tree editor to add hints to the transformations (e.g., the alignment of a widget). However, UI designers might only work on the CUI level and leave the model transformations to the developers. Also, the authors state that a graphical WYSIWYG editor would improve on their approach. We presented a technique that attempts to preserve designer input by allowing UI designers to add constraints on the CUI model [Akiki et al. 2013c]. Yet, more work is still required to make this technique applicable in practice. One approach might be to devise an algorithm that would convert designer constraints into a constraint problem that can be handled by a constraint solver. The output of the solver would become part of the adaptation rules in order to maintain the designer's input.

Technical characteristics: The following technical criteria could be improved upon by future research targeting the development of support tools that help stakeholders in devising adaptive model-driven user interfaces:

— *Completeness* is related to the types of UIs (e.g., WIMP, tangible, etc.) that can be produced using a UI development system. A system with higher completeness can be used for developing a wider variety of software applications.

— A tool's *threshold and ceiling* respectively indicate how steep its learning curve is and how advanced its outcome can be.

The gap in completeness is shared between the existing UI adaptation techniques and their supporting tools. Most existing systems focus on adapting WIMP-style UIs whereas other UI types are not supported. Providing this support requires meta-models that can be used to create concrete UI models for different UI types. Also, the supporting IDEs would have to provide visual-design tools for defining these models.

We consider tool support to be crucial for the adoption of adaptive model-driven UI development by the software industry. However, the existing tools still require a lot of work before they can become comparable to existing commercial or open-source integrated development environments such as Visual Studio or Eclipse. The threshold and ceiling of these tools need improvement. Several existing tools such as: Cedar Studio, Damask, and Gummy are a good starting point. Separate tools coming from the same research groups could be merged together such as: GraphiXML, IdealXML, and SketchiXML on one hand, and MARIA and CTTE on the other hand. Merging these tools will make them more comprehensive before new enhancements can be added. More work is required to reach an industrial quality tool that can be used for the development of real-life projects. Testing these tools with software developers working on such projects could help in measuring their *threshold and ceiling* and in providing insights on ways to improve them.

As a final future outlook, we think that in addition to addressing the gaps that we identified in this review, packaging adaptive model-driven UI development systems as general-purpose products could increase their usefulness for real-life projects in the same way that existing commercial tools are useful for developing traditional UIs. The literature already offers several approaches and prototypes. Therefore, it might be the appropriate time for a joint venture between academics working on adaptive model-driven UIs and industrial partners with a real interest in adopting this approach for developing commercial applications.

9. ACKNOWLEDGEMENTS

This work is funded by the Computing and Communications department at The Open University, and ERC Advanced Grant 291652.

REFERENCES

ABRAMS, M., PHANOURIOU, C., BATONGBACAL, A.L., WILLIAMS, S.M. AND SHUSTER, J.E., 1999. UIML: An Appliance-Independent XML User Interface Language. *Computer Networks*, 31(11-16), pp.1695–1708.

AKIKI, P.A., BANDARA, A.K. AND YU, Y., 2013a. Cedar Studio: An IDE Supporting Adaptive Model-Driven User Interfaces for Enterprise Applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. London, UK: ACM, pp. 139–144.

AKIKI, P.A., BANDARA, A.K. AND YU, Y., 2013b. Crowdsourcing User Interface Adaptations for Minimizing the Bloat in Enterprise Applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. London, UK: ACM, pp. 121–126.

AKIKI, P.A., BANDARA, A.K. AND YU, Y., 2014. Integrating Adaptive User Interface Capabilities in Enterprise Applications. In *Proceedings of the 36th International Conference on Software Engineering*.

Hyderabad, India: IEEE/ACM.

AKIKI, P.A., BANDARA, A.K. AND YU, Y., 2013c. Preserving Designer Input on Concrete User Interfaces Using Constraints While Maintaining Adaptive Behavior. In *Proceedings of the 2nd Workshop on Context-Aware Adaptation of Service Front-Ends*. London, UK: CEUR-WS.org, pp. 9–16.

AKIKI, P.A., BANDARA, A.K. AND YU, Y., 2013d. 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*. London, UK: ACM, pp. 3–12.

AKIKI, P.A., BANDARA, A.K. AND YU, Y., 2012. Using Interpreted Runtime Models for Devising Adaptive User Interfaces of Enterprise Applications. In *Proceedings of the 14th International Conference on Enterprise Information Systems*. Wroclaw, Poland: SciTePress, pp. 72–77.

APPERT, C. AND BEAUDOUIN-LAFON, M., 2006. SwingStates: Adding State Machines to the Swing Toolkit. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*. Montreux, Switzerland: ACM, pp. 319–322.

AQUINO, N., VANDERDONCKT, J., CONDORI-FERNÁNDEZ, N., DIESTE, Ó. AND PASTOR, Ó., 2010. Usability Evaluation of Multi-Device/Platform User Interfaces Generated by Model-Driven Engineering. In *Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement*. Bolzano-Bozen, Italy: ACM, p. Article 30 – 10 Pages.

BALME, L., DEMEURE, R., BARRALON, N., COUTAZ, J., CALVARY, G. AND FOURIER, U.J., 2004. Cameleon-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In *Proceedings of the 2nd European Symposium on Ambient Intelligence*. Eindhoven, The Netherlands: Springer, pp. 291–302.

BALZERT, H., HOFMANN, F., KRUSCHINSKI, V. AND NIEMANN, C., 1996. The JANUS Application Development Environment-Generating More than the User Interface. In *Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces*. Computer-Aided Design of User Interfaces. Namur, Belgium: Presses Universitaires de Namur, pp. 183–206.

BENCOMO, N., SAWYER, P., BLAIR, G.S. AND GRACE, P., 2008. Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In *Proceedings of the 12th International Conference on Software Product Lines*. Limerick, Ireland: Lero Int. Science Centre, University of Limerick, pp. 23–32.

BERGH, J., SAHNI, D. AND CONINX, K., 2010. Task Models for Safe Software Evolution and Adaptation. In D. England, P. Palanque, J. Vanderdonckt, & P. Wild, eds. *Task Models and Diagrams for User Interface Design*. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 72–77.

BERTI, S., CORREANI, F., PATERNO, F. AND SANTORO, C., 2004. The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels. In *Proceedings the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages*. Advanced Visual Interfaces. Gallipoli, Italy, pp. 103–110.

BIHLER, P. AND MÜGGE, H., 2007. Supporting Cross-Application Contexts with Dynamic User Interface Fusion. In R. Koschke, O. Herzog, K.-H. Rödiger, & M. Ronthaler, eds. *Proceedings of INFORMATIK*. LNI. Bremen, Germany: GI, pp. 459–464.

BLOUIN, A. AND BEAUDOUX, O., 2010. Improving Modularity and Usability of Interactive Systems with Malai. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. New York, USA: ACM, pp. 115–124.

BLOUIN, A., MORIN, B., BEAUDOUX, O., NAIN, G., ALBERS, P. AND JÉZÉQUEL, J.-M., 2011. Combining Aspect-Oriented Modeling with Property-Based Reasoning to Improve User Interface Adaptation. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Pisa, Italy: ACM, pp. 85–94.

BLUMENDORF, M., FEUERSTACK, S. AND ALBAYRAK, S., 2006. Event-based Synchronization of Model-Based Multimodal User Interfaces. In A. Pleuss, J. V. den Bergh, H. Hussmann, S. Sauer, & A. Boedcher, eds. *Proceedings of the 2nd International Workshop on Model Driven Development of Advanced User Interfaces*. 9th International Conference on Model-Driven Engineering Languages and Systems. Genova, Italy: Springer-Verlag.

BLUMENDORF, M., FEUERSTACK, S. AND ALBAYRAK, S., 2008. Multimodal Smart Home User Interfaces. In K. Mukasa, A. Holzinger, & A. Karshmer, eds. *Proceedings of the Workshop on Intelligent User Interfaces for Ambient Assisted Living*. 13th International Conference on Intelligent User Interfaces. Gran Canaria, Spain: ACM.

BLUMENDORF, M., FEUERSTACK, S. AND ALBAYRAK, S., 2007. Multimodal User Interaction in Smart Environments: Delivering Distributed User Interfaces. In M. Mühlhäuser, A. Ferscha, & E. Aitenbichler, eds. *Constructing Ambient Intelligence*. Berlin: Springer-Verlag, pp. 113–120.

BODART, F., HENNEBERT, A.-M., LEHEUREUX, J.-M., PROVOT, I., SACRÉ, B. AND VANDERDONCKT, J., 1995. Towards a Systematic Building of Software Architecture: The TRIDENT Methodological Guide. In P. A. Palanque & Rémi Bastide, eds. *Design, Specification and Verification of Interactive Systems*. Proceedings of the Eurographics Workshop. Toulouse, France: Springer, pp. 262–278.

BOTTERWECK, G., 2011. Multi Front-End Engineering. In H. Hussmann, G. Meixner, & D. Zuehlke, eds. *Model-Driven Development of Advanced User Interfaces*. Springer, pp. 27–42.

BRDICZKA, O., CROWLEY, J.L. AND REIGNIER, P., 2007. *Learning Situation Models for Providing Context-Aware Services*, Springer.

BYRNE, E.A. AND PARASURAMAN, R., 1996. Psychophysiology and adaptive automation. *Biological Psychology*, 42(3), pp.249–268.

CALVARY, G., COUTAZ, J., DÂASSI, O., BALME, L. AND DEMEURE, A., 2005b. Towards a New Generation of Widgets for Supporting Software Plasticity: The "Comet." In R. Bastide, P. Palanque, & J. Roth, eds. *Engineering Human Computer Interaction and Interactive Systems*. Springer, pp. 306–324.

CALVARY, G., COUTAZ, J., THEVENIN, D., LIMBOURG, Q., BOUILLON, L. AND VANDERDONCKT, J., 2003. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15, pp.289–308.

CARROLL, J.M. AND CARRITHERS, C., 1984. Training Wheels in a User Interface. *Communications of the ACM*, 27(8), pp.800–806.

CHENG, B.H.C. ET AL., 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, & J. Magee, eds. *Software Engineering for Self-Adaptive Systems*. Springer, pp. 1–26.

CLERCKX, T., LUYTEN, K. AND CONINX, K., 2005. DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In R. Bastide, P. A. Palanque, & J. Roth, eds. *Engineering Human Computer Interaction and Interactive Systems*. Springer, pp. 77–95.

CLERCKX, T., VANDERVELPEN, C., LUYTEN, K. AND CONINX, K., 2006. A Task-Driven User Interface Architecture for Ambient Intelligent Environments. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*. Sydney, Australia: ACM, pp. 309–311.

CONINX, K., LUYTEN, K., VANDERVELPEN, C., BERGH, J.V.D. AND CREEMERS, B., 2003. Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In *Human-Computer Interaction with Mobile Devices and Services*. Proceedings of the 5th International Symposium on Human-Computer Interaction with Mobile Devices and Services. Udine, Italy: Springer, pp. 256–270.

COUTAZ, J., 2010. User Interface Plasticity: Model Driven Engineering to the Limit! In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Berlin, Germany: ACM, pp. 1–8.

COYETTE, A. AND VANDERDONCKT, J., 2005. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In *Proceedings of 10th IFIP TC 13 International Conference on Human-Computer Interaction*. Rome, Italy: Springer, pp. 12–16.

CREASE, M., BREWSTER, S. AND GRAY, P., 2000. Caring, Sharing Widgets: A Toolkit of Sensitive Widgets. In *In Proceedings of the 14th British Computer Society Human Computer Interaction Conference*. United Kingdom: Springer, pp. 257–270.

DEMEURE, A., CALVARY, G. AND CONINX, K., 2008. COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces. In T. C. Graham & P. Philippe, eds. *Proceedings of the 15th International Workshop on Interactive Systems Design Specification and Verification*. Kingston, Canada: Springer, pp. 225 – 237.

DEMEURE, A., MESKENS, J., LUYTEN, K. AND CONINX, K., 2009. Design by Example of Graphical User Interfaces Adapting to Available Screen Size. In V. Lopez-Jaquero, J. P. Molina, F. Montero, & J. Vanderdonckt, eds. *Proceedings of the 7th International Conference on Computer-Aided Design of User Interfaces*. Albacete, Spain: Springer-Verlag, pp. 277–282.

DRAGICEVIC, P. AND FEKETE, J.-D., 2001. Input Device Selection and Interaction Configuration with ICON. In A. Blandford, J. Vanderdonckt, & P. Gray, eds. *People and Computers XV—Interaction without Frontiers*. Springer, pp. 543–558.

DUARTE, C. AND CARRIÇO, L., 2006. A Conceptual Framework for Developing Adaptive Multimodal Applications. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*. Sydney, Australia: ACM, pp. 132–139.

ELWERT, T. AND SCHLUNGBAUM, E., 1995. Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In P. A. Palanque & R. Bastide, eds. *Proceedings of the Eurographics Workshop on Design, Specification and Verification of Interactive Systems*. Toulouse, France: Springer, pp. 193–208.

FEUERSTACK, S., 2008. *A Method for the User-centered and Model-based Development of Interactive Application*. PhD Thesis. Berlin, Germany: Technischen Universität Berlin.

FEUERSTACK, S., BLUMENDORF, M. AND ALBAYRAK, S., 2006. Bridging the Gap between Model and Design of User Interfaces. In R. L. Christian Hochberger, ed. *GI Jahrestagung (2)*. GI-Edition - Lecture Notes in Informatics. Dresden, Germany: GI, pp. 131–137.

FEUERSTACK, S., BLUMENDORF, M., LEHMANN, G. AND ALBAYRAK, S., 2006. Seamless Home Services. In A. Maña & V. Lotz, eds. *Developing Ambient Intelligence*. Springer, pp. 1–10.

FEUERSTACK, S., BLUMENDORF, M., SCHWARTZE, V. AND ALBAYRAK, S., 2008. Model-based Layout

Generation. In P. Bottoni & S. Levialdi, eds. *Proceedings of the Working Conference on Advanced Visual Interfaces*. Napoli, Italy: ACM, pp. 217–224.

FINDLATER, L. AND MCGRENERE, J., 2007. Evaluating Reduced-Functionality Interfaces According to Feature Findability and Awareness. In *Proceedings of the 13th International Conference on Human-Computer Interaction*. Springer, pp. 592–605.

FLORINS, M., 2006. *Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces*. PhD Thesis. Louvain, Belgium: Université Catholique de Louvain.

FLORINS, M. AND VANDERDONCKT, J., 2004b. Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*. Funchal, Madeira, Portugal: ACM, pp. 140–147.

FOLEY, J., KIM, W.C., KOVACEVIC, S. AND MURRAY, K., 1991. UIDE—an intelligent user interface design environment. In *Intelligent user interfaces*. pp. 339–384.

FONSECA, J.M.C., 2010. Model-Based UI XG Final Report. *http://bit.ly/ModelBasedUIXGFinalReport*.

FRANCE, R. AND RUMPE, B., 2007. Model-Driven Development of Complex Software: A Research Roadmap. In *Proceedings of the Workshop on the Future of Software Engineering*. International Conference on Software Engineering. Minneapolis, USA: IEEE, pp. 37–54.

FRANKEL, D., 2003. *Model Driven Architecture : Applying MDA to Enterprise Computing*, Wiley.

GAJOS, K. AND WELD, D.S., 2005. Preference Elicitation for Interface Optimization. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*. Seattle, USA: ACM, pp. 173–182.

GAJOS, K.Z., WELD, D.S. AND WOBBROCK, J.O., 2010. Automatically Generating Personalized User Interfaces with Supple. *Artificial Intelligence*, 174(12-13), pp.910–950.

GAJOS, K.Z., WOBBROCK, J.O. AND WELD, D.S., 2007. Automatically Generating User Interfaces Adapted to Users' Motor and Vision Capabilities. In *Proceedings of the 20th annual ACM Symposium on User Interface Software and Technology*. Newport, Rhode Island, USA: ACM, pp. 231–240.

GALVAO, I. AND GOKNIL, A., 2007. Survey of Traceability Approaches in Model-Driven Engineering. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*. Annapolis, Maryland, USA: IEEE, pp. 313–326.

GARCÍA FREY, A., CALVARY, G. AND DUPUY-CHESSA, S., 2010. Xplain: An Editor for Building Self-Explanatory User Interfaces by Model-Driven Engineering. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Berlin, Germany: ACM, pp. 41–46.

GARCÍA FREY, A., CÉRET, E., DUPUY-CHESSA, S., CALVARY, G. AND GABILLON, Y., 2012. UsiComp: An Extensible Model-Driven Composer. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Copenhagen, Denmark: ACM, pp. 263–268.

GARLAN, D., CHENG, S.-W., HUANG, A.-C., SCHMERL, B. AND STEENKISTE, P., 2004. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*, 37(10), pp.46–54.

GREEN, M., 1985. The University of Alberta User Interface Management System. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. San Francisco, California: ACM, pp. 205–213.

GRIFFITHS, T. ET AL., 2001. Teallach: a model-based user interface development environment for object databases. *Interacting with Computers*, 14(1), pp.31–68.

GUERRERO-GARCIA, J., GONZALEZ-CALLEROS, J.M., VANDERDONCKT, J. AND MUNOZ-ARTEAGA, J., 2009. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proceedings of the 2009 Latin American Web Congress*. Merida, Yucatan, Mexico: IEEE, pp. 36–43.

HAYES, P.J., SZEKELY, P.A. AND LERNER, R.A., 1985. Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. In *Proceedings of the 3rd SIGCHI Conference on Human factors in Computing Systems*. San Francisco, California: ACM, pp. 169–175.

HILL, R.D., 1986. Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction—the Sassafras UIMS. *ACM Transactions on Graphics*, 5(3), pp.179–210.

HUEBSCHER, M.C. AND MCCANN, J.A., 2008. A Survey of Autonomic Computing—Degrees, Models, and Applications. *ACM Computing Surveys*, 40(3), pp.7:1–7:28.

HUXHAM, F.A., BURNARD, D. AND TAKATSUKA, J., 1986. *Using the Macintosh Toolbox with C*, SYBEX.

IBM, 2006. An Architectural Blueprint for Autonomic Computing.

JABARIN, B. AND GRAHAM, T.C.N., 2003. Architectures for Widget-Level Plasticity. In J. Jorge, N. Jardim Nunes, & J. Falcão e Cunha, eds. *Interactive Systems. Design, Specification, and Verification*. Springer, pp. 124–138.

JACOB, R.J., 1986. A Specification Language for Direct-Manipulation User Interfaces. *ACM Transactions on Graphics*, 5(4), pp.283–317.

JACOBSON, S., SHEPHERD, J., D'AQUILA, M. AND CARTER, K., 2007. *The ERP Market Sizing Report, 2006–2011*, Boston, MA: AMR Research, Inc.

JANSSEN, C., WEISBECKER, A. AND ZIEGLER, J., 1993. Generating User Interfaces from Data Models and Dialogue Net Specifications. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human*

Factors in Computing Systems. Amsterdam, The Netherlands: ACM, pp. 418–423.

JOHNSON, J., 1992. Selectors: Going Beyond User-Interface Widgets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Monterey, California, USA: ACM, pp. 273–279.

KAWAI, S., AIDA, H. AND SAITO, T., 1996. Designing Interface Toolkit with Dynamic Selectable Modality. In *Proceedings of the 2nd Annual ACM Conference on Assistive Technologies*. Vancouver, British Columbia, Canada: ACM, pp. 72–79.

KEATES, S., CLARKSON, P.J., HARRISON, L.-A. AND ROBINSON, P., 2000. Towards a Practical Inclusive Design Approach. In *Proceedings on the 2000 Conference on Universal Usability*. Arlington, Virginia, USA: ACM, pp. 45–52.

KENT, S., 2002. Model Driven Engineering. In *Proceedings of the 3rd International Conference on Integrated Formal Methods*. Turku, Finland: Springer, pp. 286–298.

KRAMER, J. AND MAGEE, J., 2007. Self-Managed Systems: an Architectural Challenge. In *Proceedings of the Workshop on the Future of Software Engineering*. International Conference on Software Engineering. Minneapolis, USA: IEEE, pp. 259–268.

LAFRENIERE, B., BUNT, A., LOUNT, M., KRYNICKI, F. AND TERRY, M.A., 2011. AdaptableGIMP: Designing a Socially-Adaptable Interface. In *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology*. Santa Barbara, California, USA: ACM, pp. 89–90.

LECOLINET, E., 2003. A Molecular Architecture for Creating Advanced GUIs. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*. Vancouver, Canada: ACM, pp. 135–144.

LEHMANN, G., RIEGER, A., BLUMENDORF, M. AND ALBAYRAK, S., 2010. A 3-Layer Architecture for Smart Environment Models. In *Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications*. Mannheim, Germany: IEEE, pp. 636 –641.

LEPREUX, S., VANDERDONCKT, J. AND MICHOTTE, B., 2007. Visual Design of User Interfaces by (De)Composition. In *Proceedings of the 13th International Conference on Interactive Systems: Design, Specification, and Verification*. Dublin, Ireland: Springer, pp. 157–170.

LIMBOURG, Q. AND VANDERDONCKT, J., 2004. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications: Proceedings of Workshops in connection with the 4th International Conference on Web Engineering*. Munich, Germany: Rinton Press, pp. 325–338.

LIN, J. AND LANDAY, J.A., 2008. Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. Florence, Italy: ACM, pp. 1313–1322.

LONCZEWSKI, F. AND SCHREIBER, S., 1996. The FUSE-System: an Integrated User Interface Design Environment. In *Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces*. Namur, Belgium: Springer, pp. 37–56.

LÓPEZ-JAQUERO, V., MONTERO, F. AND REAL, F., 2009. Designing User Interface Adaptation Rules with T:XML. In 14th International Conference on Intelligent User Interfaces. Sanibel Island, Florida, USA: ACM, pp. 383–388.

MACE, R.L., HARDIE, G. E J. AND PLACE, J.P., 1990. Accessible Environments: Toward Universal Design. In *Design Intervention: Toward a More Humane Architecture*. Center for Accessible Housing, North Carolina State University.

MARKOPOULOS, P., PYCOCK, J., WILSON, S. AND JOHNSON, P., 1992. Adept-A task based design environment. In *Proceedings of the 25th Hawaii International Conference on System Sciences*. Hawaii, USA: IEEE, pp. 587–596.

MÄRTIN, C., 1996. Software Life Cycle Automation for Interactive Applications: The AME Design Environment. In *Proceedings of the 2nd International on Computer-Aided Design of User Interfaces*. Namur, Belgium: Presses Universitaires de Namur, pp. 57–76.

MAYHEW, D.J., 1999. The Usability Engineering Lifecycle. In *Proceedings of the Extended Abstracts of the 17th Conference on Human Factors in Computing Systems*. Pittsburgh, Pennsylvania: ACM, pp. 147–148.

MCGRENERE, J., 2000. "Bloat": The Objective and Subject Dimensions. In *Proceedings of the Extended Abstracts of the 18th Conference on Human Factors in Computing Systems*. The Hague, The Netherlands: ACM, pp. 337–338.

MCGRENERE, J., BAECKER, R.M. AND BOOTH, K.S., 2002. An Evaluation of a Multiple Interface Design Solution for Bloated Software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Minneapolis, Minnesota, USA: ACM, pp. 164–170.

MEIXNER, G., PATERNÒ, F. AND VANDERDONCKT, J., 2011. Past, Present, and Future of Model-Based User Interface Development. *i-com*, 10(3), pp.2–11.

MESKENS, J., VERMEULEN, J., LUYTEN, K. AND 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*. Napoli, Italy: ACM, pp. 233–240.

MICHOTTE, B. AND VANDERDONCKT, J., 2008. GrafiXML, a Multi-target User Interface Builder Based on UsiXML. In *Proceedings of the 4th International Conference on Autonomic and Autonomous Systems*. Cancun, Mexico: IEEE, pp. 15 –22.

MICROSOFT, 2011. Role based UI - Dynamics CRM 2011.

MILLER, C., FUNK, H., WU, P., GOLDMAN, R., MEISNER, J. AND CHAPMAN, M., 2005. The Playbook™ Approach to Adaptive Automation. In *Proceedings of the 49th Human Factors and Ergonomics Society Annual Meeting*. Florida, U.S.A.: SAGE Publications, pp. 15–19.

MONTERO, F. AND LÓPEZ-JAQUERO, V., 2007. IdealXML: An Interaction Design Tool. In G. Calvary, C. Pribeanu, G. Santucci, & J. Vanderdonckt, eds. *Computer-Aided Design of User Interfaces V*. Springer, pp. 245–252.

MORI, G., PATERNÒ, F. AND SANTORO, C., 2002. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Transactions on Software Engineering*, 28(8), pp.797–813.

MYERS, B., HUDSON, S.E. AND PAUSCH, R., 2000. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 7(1), pp.3–28.

MYERS, B.A. ET AL., 1997. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering*, 23(6), pp.347–365.

NEBELING, M., LEONE, S. AND NORRIE, M., 2012. Crowdsourced Web Engineering and Design. In M. Brambilla, T. Tokuda, & R. Tolksdorf, eds. *Web Engineering*. Springer, pp. 31–45.

NEBELING, M. AND NORRIE, M.C., 2011. Tools and Architectural Support for Crowdsourced Adaptation of Web Interfaces. In *Proceedings of the 11th International Conference on Web Engineering*. Paphos, Cyprus: Springer, pp. 243–257.

NICHOLS, J., MYERS, B.A. AND LITWACK, K., 2004. Improving Automatic Interface Generation with Smart Templates. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*. Funchal, Madeira, Portugal: ACM, pp. 286–288.

NORCIO, A.F. AND STANLEY, J., 1989. Adaptive Human-Computer Interfaces: A Literature Survey and Perspective. *IEEE Transactions on Systems, Man, and Cybernetics*, 19, pp.399–408.

NYLANDER, S., BYLUND, M. AND WÆRN, A., 2004. The Ubiquitous Interactor - Device Independent Access to Mobile Services. In *Proceedings of 4th International Conference on Computer-Aided Design of User Interfaces*. Funchal, Portugal: Kluwer, pp. 269–280.

OLSEN,JR., D.R., 1989. A Programming Language Basis for User Interface Management. In *Proceedings of the 7th ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI '89. Austin, USA: ACM, pp. 171–176.

OLSEN,JR., D.R., 2007. Evaluating User Interface Systems Research. In *Proceedings of the 20th ACM SIGCHI Symposium on User Interface Software and Technology*. Newport, Rhode Island, USA: ACM, pp. 251–258.

OLSEN,JR., D.R., 1986. MIKE: The Menu Interaction Kontrol Environment. *ACM Transactions on Graphics*, 5(4), pp.318–344.

OMG, 2013. Object Management Group - Model-Driven Architecture. *http://www.omg.org/mda*.

OREIZY, P. ET AL., 1999. An Architecture-Based Approach to Self-Adaptive Software. *Intelligent Systems and Their Applications, IEEE*, 14(3), pp.54–62.

PALAY, A.J., 1989. The Andrew Toolkit: An Overview. In *Proceedings of the 1988 Winter USENIX Technical Conference*. Dallas, Texas: USENIX Association, pp. 9–21.

PATERNÒ, F., 1999. *Model-based Design and Evaluation of Interactive Applications* 1st ed., London, UK: Springer.

PATERNÒ, F., MANCINI, C. AND MENICONI, S., 1997. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the 6th International Conference on Human-Computer Interaction*. Sydney, Australia: Chapman and Hall, pp. 362–369.

PATERNO', F., SANTORO, C. AND 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*, 16(4), pp.19:1–19:30.

PEISSNER, M., HÄBE, D., JANSSEN, D. AND SELLNER, T., 2012. MyUI: Generating Accessible User Interfaces from Multimodal Design Patterns. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. EICS '12. Copenhagen, Denmark: ACM, pp. 81–90.

PÉREZ-MEDINA, J.-L., DUPUY-CHESSA, S. AND FRONT, A., 2007. A Survey of Model Driven Engineering Tools for User Interface Design. In M. Winckler, H. Johnson, & P. Palanque, eds. *Task Models and Diagrams for User Interface Design*. Springer, pp. 84–97.

PLEUSS, A., BOTTERWECK, G. AND DHUNGANA, D., 2010. Integrating Automated Product Derivation and Individual User Interface Design. In *Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems*. Linz, Austria: Universitat Duisburg-Essen, pp. 69–76.

PUERTA, A. AND EISENSTEIN, J., 1998. Interactively Mapping Task Models to Interfaces in MOBI-D. In *Proceedings of Eurographics Workshop on Design, Specification and Validation of Interactive Systems*. Abingdon, United Kingdom: Springer, pp. 261–273.

PUERTA, A. AND EISENSTEIN, J., 2002. XIML: A Common Representation for Interaction Data. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*. San Francisco, California, USA: ACM, pp. 214–215.

PUERTA, A.R., 1996. The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. In *Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces*. Namur, Belgium: Presses Universitaires de Namur, pp. 19–36.

RANEBURGER, D., POPP, R. AND VANDERDONCKT, J., 2012. An Automated Layout Approach for Model-Driven WIMP-UI Generation. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Copenhagen, Denmark: ACM, pp. 91–100.

REINECKE, K. AND BERNSTEIN, A., 2011. Improving performance, perceived usability, and aesthetics with culturally adaptive user interfaces. *ACM Transactions on Computer-Human Interaction*, 18, pp.1–29.

SALEHIE, M. AND TAHVILDARI, L., 2009. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4, pp.1–42.

SCHMUCKER, K.J., 1987. MacApp: An Application Framework. In R. M. Baecker & W. A. S. Buxton, eds. *Human-computer Interaction*. Morgan Kaufmann Publishers Inc., pp. 591–594.

SCHWARTZE, V., FEUERSTACK, S. AND ALBAYRAK, S., 2009. Behavior-Sensitive User Interfaces for Smart Environments. In *Proceedings of the 2nd International Conference on Digital Human Modeling*. San Diego, USA: Springer, pp. 305–314.

SHNEIDERMAN, B., 2003. Promoting Universal Usability with Multi-Layer Interface Design. In *Proceedings of the Conference on Universal Usability*. CUU '03. Vancouver, Canada: ACM, pp. 1–8.

DA SILVA, P.P., 2001. User Interface Declarative Models and Development Environments: A survey. In *Proceedings of the 7th International Conference on Design, Specification, and Verification of Interactive Systems*. Limerick, Ireland: Springer, pp. 207–226.

SINGH, A. AND WESSON, J., 2009. Evaluation Criteria for Assessing the Usability of ERP Systems. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*. Emfuleni, South Africa: ACM, pp. 87–95.

SOLEY, R. AND OMG STAFF STRATEGY GROUP, 2000. Model Driven Architecture. *http://bit.ly/ModelDrivenArch*.

SOLOVEY, E.T. ET AL., 2011. Sensing Cognitive Multitasking for a Brain-Based Adaptive User Interface. In *Proceedings of the 29th SIGCHI Conference on Human Factors in Computing Systems*. Vancouver, Canada: ACM, pp. 383–392.

STEPHANIDIS, C., 1997. Towards the Next Generation of UIST: Developing for all Users. In *Proceedings of the 7th International Conference on Human-Computer Interaction*. HCI International '97. New York, NY, USA: Elsevier Science Inc., pp. 473–476.

STUERZLINGER, W., CHAPUIS, O., PHILLIPS, D. AND ROUSSEL, N., 2006. User Interface Façades: Towards Fully Adaptable User Interfaces. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*. Montreux, Switzerland: ACM, pp. 309–318.

SYNACTIVE GMBH, 2010. GuiXT - Simplify and Optimize the SAP ERP User Interface. *http://bit.ly/SAPGuiXTSimplifyUI*.

SZEKELY, P., LUO, P. AND NECHES, R., 1992. Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In *Proceedings of the 10th ACM SIGCHI Conference on Human Factors in Computing Systems*. Monterey, California, USA: ACM, pp. 507–515.

SZEKELY, P., SUKAVIRIYA, P., CASTELLS, P., MUTHUKUMARASAMY, J. AND SALCHER, E., 1995. Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach. *Engineering for Human-Computer Interaction*, 10.

TOPI, H., LUCAS, W.T. AND BABAIAN, T., 2005. Identifying Usability Issues with an ERP Implementation. In *Proceedings of the 7th International Conference on Enterprise Information Systems*. Miami, USA: SciTePress, pp. 128–133.

UFLACKER, M. AND BUSSE, D., 2007. Complexity in Enterprise Applications vs. Simplicity in User Experience. In *Proceedings of the 12th International Conference on Human-Computer Interaction: Applications and Services*. Beijing, China: Springer, pp. 778–787.

VANDERDONCKT, J., 2008. Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges. *Proceedings of ROCHI*, 8.

VANDERDONCKT, J. AND BODART, F., 1996. The "Corpus Ergonomicus": A Comprehensive and Unique Source for Human-Machine Interface. In *Proceedings of the 1st International Conference on Applied Ergonomics*. Istanbul, Turkey: USA Publishing, pp. 162–169.

VANDERDONCKT, J.M. AND BODART, F., 1993. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*. Amsterdam, The Netherlands: ACM, pp. 424–429.

WIECHA, C., BENNETT, W., BOIES, S., GOULD, J. AND GREENE, S., 1990. ITS: A Tool for Rapidly Developing Interactive Applications. *ACM Transactions on Information Systems*, 8(3), pp.204–236.