Engineering Adaptive Model-Driven User Interfaces

Journal Item

Engineering Adaptive Model-Driven User Interfaces

Pierre A. Akiki, Arosha K. Bandara, *Member, IEEE,* and Yijun Yu, *Member, IEEE*

Abstract—Software applications that are very large-scale, can encompass hundreds of complex user interfaces (UIs). Such applications are commonly sold as feature-bloated off-the-shelf products to be used by people with variable needs in the required features and layout preferences. Although many UI adaptation approaches were proposed, several gaps and limitations including: extensibility and integration in legacy systems, still need to be addressed in the state-of-the-art adaptive UI development systems. This paper presents Role-Based UI Simplification (RBUIS) as a mechanism for increasing usability through adaptive behavior by providing end-users with a minimal feature-set and an optimal layout, based on the context-of-use. RBUIS uses an interpreted runtime model-driven approach based on the Cedar Architecture, and is supported by the integrated development environment (IDE), Cedar Studio. RBUIS was evaluated by integrating it into OFBiz, an open-source ERP system. The integration method was assessed and measured by establishing and applying technical metrics. Afterwards, a usability study was carried out to evaluate whether UIs simplified with RBUIS show an improvement over their initial counterparts. This study leveraged questionnaires, checking task completion times and output quality, and eye-tracking. The results showed that UIs simplified with RBUIS significantly improve end-user efficiency, effectiveness, and perceived usability.

Index Terms—Design Tools and Techniques, Software Architectures, Support for Adaptation, User Interfaces

1 Introduction

Software applications that are very large-scale, can encompass hundreds of complex user interfaces (UIs). Such applications are commonly sold as feature-bloated off-the-shelf products to be used by people with variable needs of the required UI feature-set and layout preferences. For example, end-users with different job descriptions could require a variable sub-set of a UI's features. Additionally, each end-user's layout preferences in terms of UI factors, e.g. font-size, widget grouping, widget types, etc., can be driven by context-of-use facets that we refer to as *aspects*, e.g. computer literacy, culture, motor-abilities, screen-size, etc. Enterprise applications, e.g. enterprise resource planning (ERP) systems, are a common example of such types of software applications. Hence, it is not surprising that these applications suffer from many usability problems [1], some of which are caused by their complex off-the-shelf UIs. Due to these considerations, enterprise applications are used in this paper to illustrate and evaluate our approach for engineering adaptive UIs.

1.1 Adaptive Model-Driven User Interfaces

There are existing solutions, which attempt to address some usability problems by adapting UIs to the context-of-use.

Adaptive UIs are aware of their context-of-use and are capable of providing a response to changes in this context, by adapting one or more of their characteristics using rules that are either predefined or deduced by learning over time. For example, an adaptive UI can automatically increase the font-size for end-users with a vision-impairment. On the other hand, an adaptable UI can allow end-users to manually adapt desired characteristics, e.g. adding and removing toolbar buttons. Adaptive UIs, i.e. semi/fully-automated adaptation, could be a more feasible approach for complex large-scale systems (e.g., enterprise applications), than adaptable ones, i.e. manual adaptation, mainly due to two reasons. First, the scope of variability, based on which the UI can be adapted, might not be known at design-time. Second, it could be very costly to manually develop several context-dependent UI variations for a large number of UIs.

There is an increased need for user-adaptivity, accompanied by an increase in the feasibility of realizing adaptive systems due to advances in research [2]. Adaptation can provide benefits in certain cases such as improving an end-user's performance [3]. Existing studies have reported certain conditions that affect the benefit of adaptive UIs. Adding adaptation capabilities to a system depends on different variables such as the task and the user [4]. For example, users performing complex tasks could benefit more from adaptation than those performing simple tasks. Also, end-users tend to take more advantage of an accurate adaptive UI, due to the establishment of trust [5]. Another important point to consider when adopting adaptive UIs, is providing the end-users with a preferences area where they can provide feedback to adjust the adaptive behavior [6]. Our approach leverages the results of existing works, in order to provide a better outcome.

The model-driven UI development approach formed the basis for many works researching adaptive UIs. This approach has advantages such as offering technology independence, and providing the ability to support verification of model properties and traceability. It also offers the possibility of applying different types of adapta-

- *Pierre A. Akiki is with the Department of Computer Science, Notre Dame University – Louaize, P.O. Box: 72, Zouk Mikael, Lebanon. E-mail: pakiki@ndu.edu.lb*
- *Arosha K. Bandara and Yijun Yu are with the Computing and Communications Department, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom E-mail: firstname.lastname@open.ac.uk*

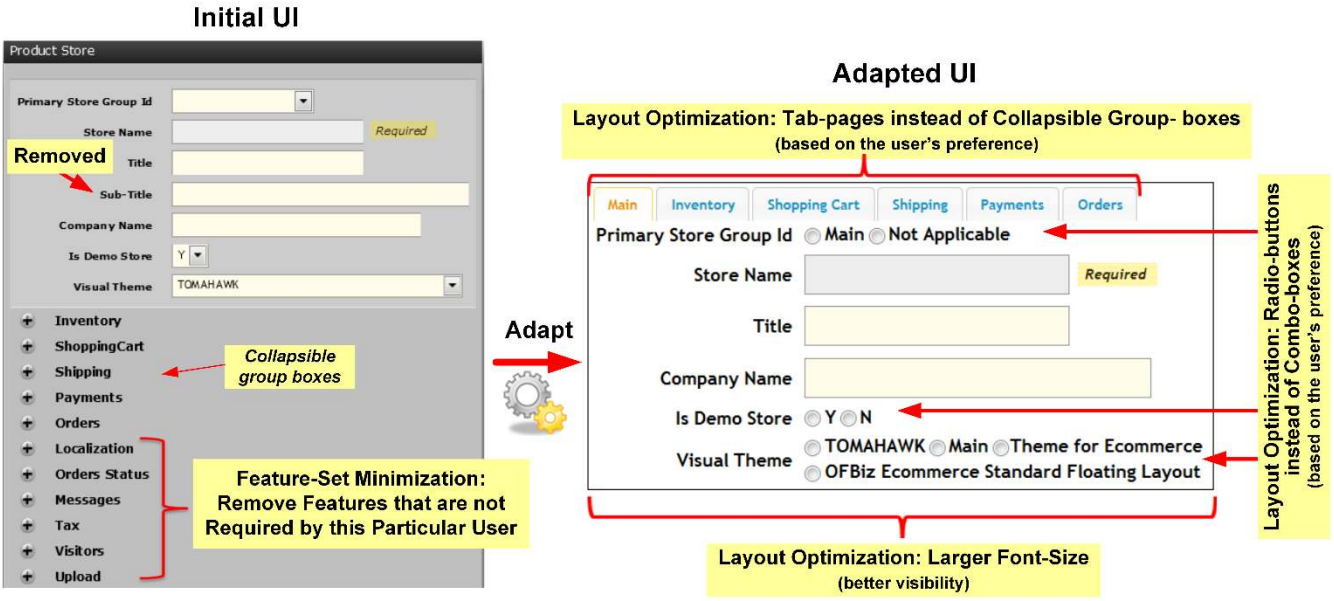


Fig. 1. An Example on Adapting a User Interface with RBUIS by Minimizing its Feature-Set and Optimizing its Layout

tions on the various UI levels of abstraction. More details about the advantages of using this approach for developing adaptive UIs can be found in our survey of the state-of-the-art [7].

1.2 Limitations of Existing Adaptive Model-Driven User Interface Development Systems

We assert that an adaptive model-driven UI development system should comprise three parts: a **reference-architecture** depicting the various characteristics of the proposed approach, a practical **adaptation technique** for achieving the sought after adaptive behavior based on this reference-architecture, and a **support tool** for allowing stakeholders, e.g. developers and IT personnel, to develop UIs and adapt them with the proposed adaptation technique.

The state-of-the-art adaptive model-driven UI development systems provide at least one of the parts mentioned above. However, several gaps and limitations still need to be addressed in these systems to obtain a solution that can be applied to large-scale software systems such as enterprise applications. For example, feature-set adaptation techniques suffer from at least one of the following problems: lack of a practical implementation mechanism, e.g. training wheels UI [8]; lack of generality, e.g. SAP's GuiXT [9]; or restriction to design-time UI adaptation, e.g. (de)composition [10]. There are also gaps and limitations that pertain to adaptive model-driven UI development systems in general, which include: lacking the ability to define adaptive behavior both visually and through code, e.g. Supple [11]; not supporting adaptive behavior for an extensible number of UI adaptation aspects and factors, e.g. MASP [12]; and limited end-user feedback techniques, e.g. MyUI [13], etc. This paper presents a novel adaptive model-driven UI development system that includes a reference-architecture, a UI adaptation technique, and a supporting tool, which address some of the gaps and limitations surveyed in the existing state-of-the-art.

The approach described in this paper can be used to provide newly developed applications with adaptive UI capabilities. Nonetheless, an adaptive UI development system also needs to support mature legacy software applications. The proposed UI adaptation technique has to integrate within existing legacy systems, in order to empower them with adaptive UI capabilities. Additionally, the integration method should work without incurring a high development cost or significantly changing the way legacy systems function. This integration challenge must be overcome to allow legacy software applications to benefit from adaptive UIs at a reasonable cost. However, this challenge has not been addressed by existing state-of-the-art adaptive UI development systems, which were mostly tested by developing new prototype UIs or redeveloping UIs from existing software systems. Therefore, as part of our approach, we present a method for integrating our UI adaptation technique into legacy software systems.

1.3 Our Contributions to Engineering Adaptive UIs

This paper presents a UI adaptation technique (TE) called Role-Based User Interface Simplification (RBUIS), which is based on a reference-architecture (AR) called the Cedar Architecture with tool (TL) support provided by the Cedar Studio integrated development environment (IDE).

The **Cedar Architecture** offers a high-level reference for developing adaptive model-driven UIs. **RBUIS** realizes some of the high level components of the Cedar Architecture to support two types of adaptation, namely feature-set minimization 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, by adapting the properties of concrete UI widgets. With RBUIS, the adaptation starts with feature-set minimization, which eliminates (e.g., hides, disables, etc.) the UI subset that is not required by particular end-users. Afterwards, layout

optimization makes the UI better-suited for the context-of-use aspects, e.g. computer literacy, culture, motor-abilities, screen-size, etc., by adapting UI factors such as: font-size, widget grouping, widget types, etc. For example, novice users might prefer the UI to be displayed as a step-by-step wizard, whereas advanced users might feel more productive if the UI is presented on one page. A study that we conducted showed that UIs adapted with RBUIS improve usability by increasing the perceived usability of end-users and helping them in fulfilling their daily tasks more efficiently and effectively. **Cedar Studio** offers visual-design and code-editing tools for supporting the development of model-driven UIs and adapting them to the context-of-use using RBUIS.

An example of a UI, which was adapted using RBUIS, is illustrated in Fig. 1. It shows the "Product Store" UI of an open-source ERP system called Apache Open For Business (OFBiz). We integrated RBUIS into OFBiz to evaluate our integration method based on several metrics. We showed that it is possible to integrate RBUIS into legacy systems without increasing the development cost or significantly changing the way these systems function. Enterprise applications such as OFBiz generally have box-like Window-Icon-Menu-Pointer (WIMP) UIs. The contributions made in this paper can work for software systems other than enterprise applications given that the same box-like WIMP UI paradigm is used. More examples of RBUIS being applied in practice can be viewed online in demonstration videos [74].

The related work is evaluated in Section 2 in order to identify its possible gaps and limitations. The gaps and limitations are addressed by presenting three novel technical contributions including: the **Cedar Architecture**, the Role-Based UI Simplification (**RBUIS**) mechanism, and their supporting IDE **Cedar Studio** in Sections 3, 4, and 5 respectively. An **integration method** for empowering legacy software applications with adaptive behavior using RBUIS is presented in Section 6 alongside a metric-based **technical evaluation**. In Section 7, we conduct a **human-centered evaluation** through a laboratory study, which tests whether RBUIS can significantly improve usability.

We have previously presented parts of this work including: [7], [14], [15], [16], [17], and [18], over the course of three years. This paper provides the first complete presentation of the technical underpinnings of our system, in addition to a comprehensive evaluation that covers both the technical and human perspectives.

2 Related Work

This section presents the criteria that we used for evaluating the related work. Afterwards, the state-of-the-art adaptive model-driven UI development systems, i.e. reference-architectures, adaptation techniques, and supporting tools, are briefly evaluated based on these criteria. This section covers some of the criteria, which showed major gaps and limitations in the existing state-of-the-art. A more elaborate review, with more evaluation criteria and details, can be found in a separate survey paper [7]. These criteria include: completeness, control over the UI, multiple data sources, etc. We should note that some existing works focus on other important areas such as multi-modality and UI distribution. This section does not cover these works unless they support UI adaptation, in order to stay within the scope of this paper.

2.1 Evaluation Criteria

In order to conduct a sound and objective critical review of the existing systems, we produced the following list of criteria by drawing on direct recommendations from the literature and combining features from multiple existing systems. Some of these criteria are implementation-dependent, thereby can only be used to evaluate practical UI adaptation techniques or tools. Other criteria are also suitable for evaluating reference-architectures. The codes AR, TE, and TL indicate the applicability of each criterion to: architectures, techniques, and/or tools respectively.

— **Extensibility** is important for new UI development approaches [19]. We refined its meaning as follows:

  — **Extensibility of adaptation aspects and factors** indicates that a UI adaptation approach can accommodate an extensible number of adaptation aspects on which basis any UI factor can be adapted. (**TE**, **TL**)

  — **Extensibility of adaptive behavior** indicates the approach's ability to support the definition of new adaptive behavior at runtime as needed. (**AR**, **TE**, **TL**)

Supporting the extensibility of adaptation aspects and factors, and adaptive behavior, allows an approach to provide more coverage in terms of possible UI adaptations.

— **Expressive leverage** "is where the designer can accomplish more by expressing less" [20] (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 adaptive behavior, e.g. visual-parts or scripts. (**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 for mature legacy systems. Providing a new advance while maintaining legacy code is desirable [20]. (**AR**, **TE, TL**)

— Using interpreted runtime models as a **modeling approach** can support more advanced runtime adaptations than code generation. Also, a major drawback of generative approaches is that, over time, models may get out-of-sync with the running code [21]. (**AR**, **TE**)

— **Scalability** is an important criterion to check for every new system [20]. If the scalability of an adaptation technique is not demonstrated with real-life scenarios, its adoption for complex software systems could decrease. (**TE**)

— An ideal tool would have low **threshold and** high **ceiling** [22]. 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**)

— **User feedback** on the adapted UI provides end-users with awareness of automated adaptation decisions and the ability to override them. It can increase the end-users' UI control [23] and feature-awareness [24] affected by adaptive/reduction mechanisms. Creating a representation for end-users and the automation to communicate is a challenge in human-automation interaction [25]. (**AR**, **TE**)

— **Visual and code-based representations** allow different

stakeholders, e.g. developers and IT personnel, to implement adaptive behavior. A textual representation, e.g. CSS in Comet(s) [19], can be advanced but a visual notation can simplify the creation of UI adaptation rules by hiding the complexity of programming languages [26]. (**TE**, **TL**)

2.2 Reference Architectures

Several architectures have been proposed as a reference for applications targeting adaptive UI behavior and other UI related features such as: multimodality, distribution, etc. This section focuses on the adaptive behavior part of existing architectures since it is our paper's main focus.

A ***3-layer architecture*** was presented for devising adaptive smart environment UIs [27]. Its *modeling approach* is based on generative runtime models, which are less flexible than interpreted runtime models for performing advanced adaptations. This architecture does not support *user feedback* but refers to another work [28], which does not offer an architecture but uses user-feedback for refining initial situation models at runtime.

CAMELEON-RT is a reference architecture model for distributed, migratable, and plastic UIs within interactive spaces [29]. It provides a good conceptual representation of the *extensibility of adaptive behavior* through the use of open-adaptive components [30], which allow new adaptive behavior to be added at runtime.

FAME is an architecture that targets adaptive multimodal UIs using a set of context models in combination with user inputs [31]. It only targets modality adaptation, hence it is not meant to be a general-purpose reference for adapting other UI characteristics.

Malai is an architectural model for interactive systems [32] and forms a basis for a technique that uses aspect-oriented modeling (AOM) for adapting UIs [33]. *Extensibility of adaptive behavior* is partially fulfilled in Malai since multiple presentations are defined at design-time by the developer, and later switched at runtime. Its *modeling approach* relies on generating code to represent the UI. Malai allows developers to define *feedback* that helps end-users to understand the state of the interactive system, but the end-users cannot provide feedback on the adaptations (e.g., reverse an unwanted adaptation).

User feedback was not addressed by any of the architectures reviewed in this section. Also, in spite of the importance of *integration* in legacy software systems, these architectures were evaluated by building new prototypes.

2.3 Adaptation Techniques

A variety of UI characteristics can be targeted by adaptation techniques. The characteristics that are the most common in the existing literature are the UI's features and layout. Other characteristics that can be adapted include: modality [31], navigation [34], content [35], etc. Despite the importance of feature and layout adaptation, the existing works that target these characteristics still suffer from limitations and gaps. Therefore, our UI adaptation technique focuses on feature-set minimization and layout optimization. In order to remain within our scope, this section presents the strengths and shortcomings of the state-of-the-art UI adaptation techniques, which target feature-set minimization, layout optimization, or both.

2.3.1 Feature-Set Minimization

Since existing feature-set minimization solutions focus on design-time adaptation rather than runtime adaptive behavior, we did not evaluate them according to the criteria established in Section 2.1. Instead, we categorized them and generally evaluated their strengths and shortcomings.

Several ***theoretical propositions*** were made for reducing a UI's feature-set based on the context-of-use, including: universal usability [36], two UI versions [23], training wheels UI [8], etc. These works present a sound theoretical basis for reducing the bloat of feature-bloated software applications. Yet, in addition to lacking a technical implementation, the given examples: basic text editor [36], Word menu [23], etc., do not match the complexity of large-scale systems such as enterprise applications.

Approaches from ***software product-line*** (SPL) engineering [37] are used to tailor software applications and some particularly address tailoring UIs. MANTRA [38] adapts UIs to multiple platforms by generating code particular to each platform from an abstract UI model. Although SPLs can be dynamic [39], the SPL-based approaches for UI adaptation focus on design-time (product-based) adaptation, whereas runtime (context-based) adaptive behavior is not addressed.

Several ***commercial software applications*** use role-based tailoring of the UI's feature-set. Microsoft Dynamics CRM [40] and SAP's GuiXT [9] 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.

Graceful degradation [41] and (de)composition [10], relied on ***decomposing*** UIs into small fragments that fit the context-of-use better. These approaches depend on design-time activities (e.g., designer annotations with "graceful degradation"). However, runtime adaptation is more desirable for situations that are not known at design-time.

The main limitations in approaches attempting to target feature-set minimization 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.

2.3.2 Layout Optimization

Existing works use different approaches for layout optimization. We provide a brief description of each of these works and argue their strengths and shortcomings based on the criteria we established in Section 2.1.

Comet(s) is a set of widgets, which support UI plasticity [42]. It is claimed that *extensibility of the adaptive behavior* is supported through cascading-style-sheets. A comet can only adjust its shape, whereas other types of adaptation (e.g., feature-set), which are related to the overall UI design, cannot be supported by this architectural-style. We consider the *modeling approach* criterion to be partially fulfilled since it requires a code-based implementation as opposed to the possibility of using interpreted runtime models.

DynaMo-AID [43] is a design-process and runtime-architecture for devising context-aware UIs and is part

of the Dygimes UI framework [44]. The support of interpreted runtime models provides a good *modeling approach*. Adaptive behavior is *extensible* but is restricted to one type of adaptation, namely the UI dialog. DynaMo-AID is particularly criticized for using a "Task Tree Forest" [32], where each tree corresponds to the tasks possible in a given context. The combinatorial explosion affects the approach's *scalability* for complex systems.

Supple supports automatic generation of UIs adapted to each user's motor and vision abilities, devices, tasks, and preferences [11]. Supple *interprets* and renders UI models at runtime, hence making the fulfillment of more advanced adaptations easier. Yet, the inability to have human input at the different levels of abstraction, at least at design-time, makes it difficult to adopt for large-scale systems. Supple has built-in algorithms for adapting the UI and does not provide a means for *extending the adaptive behavior* through either a visual or a code-based representation. 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 the means for *extending adaptation* aspects, and factors. Also, it was criticized [13] for exceeding acceptable performance times. Supple is complemented by a system called Arnauld [45], which is responsible for eliciting user preferences to adapt the UI at runtime. Arnauld could serve as a feedback mechanism, but the sole reliance on runtime elicitation can be time consuming and might not provide sufficient data.

MASP is a system for developing ubiquitous model-driven UIs for smart homes [12]. It supports: multimodality [46], distribution [47], synchronization [48], and adaptation [49]. MASP demonstrates powerful capabilities in UI distribution and multimodality, but we focus on its adaptation capabilities to stay within our scope. The *modeling approach* bases the final UI on generated code or markup [46], thus allowing less advanced adaptations to be performed at runtime as opposed to a fully-dynamic approach. A limited number of UI adaptation factors are supported (e.g., orientation, size, etc.), and no means are provided for extending the adaptation aspects and factors. MASP provides a tool to visually divide the layout into boxes. Yet, it does not support the definition of *visual and code-based adaptation rules* that can cover a variety of layout optimization factors, which go beyond changing the font-size, and layout grouping.

One technique uses aspect-oriented modeling (***AOM***) for adapting UIs [33] based on the ***Malai*** architecture (reviewed in Section 2.2). 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. The meta-model does not support the *extension of adaptation aspects and factors*. Also, no mechanism is provided for adding *adaptive behavior visually*. *Scalability* is demonstrated with complex interactive system adaptations.

MyUI is a UI development infrastructure for improving accessibility through adaptive UIs [13]. 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. 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 in MyUI is defined *visually* using a state-chart model. However, its basic adaptation examples (e.g., changing font-size) do not show whether the state-charts have the potential for defining more advanced usability-related adaptations (e.g., layout grouping, widget types, etc.).

Roam is a framework that allows developers to build applications, which can adapt to different devices [50]. One of the adaptation approaches supported by Roam is having multiple device-dependent implementations, which are created by the developers at design-time and instantiated by the system at runtime. Another approach is the implementation of a single device-independent presentation, which is built using a device-independent GUI toolkit. This device-independent implementation gets transformed at runtime to run on a target device. In both approaches, UIs are developed using a toolkit rather than a *model-driven approach*. Roam focuses on adapting UIs to device properties such a display size and the available input methods but does not support *extensibility of adaptation aspects and factors*. Additionally, it does not provide a way to express the adaptive behavior (transformations) using both *visual and code-based representations*.

Like Roam, ***XMobile*** is another environment that limits its focus to the adaptation of UIs based on variations in device properties [51]. XMobile is model-based, but it relies on generating code at design-time rather than using an interpreted runtime *modeling approach*.

Quill is a web-based development environment that focuses on cross-platform deployment by supporting collaborative model-based design of UIs [52]. Quill is limited to web-applications and cross-platform deployment and does not support *extensibility of adaptation aspects and factors*. Also, it does not support both *visual and code-based representations* of adaptive behavior.

We noticed that several criteria were not addressed by the existing layout optimization techniques. Supple and MyUI support, to different extents, *user feedback* on the adapted UI. The existing techniques were evaluated by developing new prototype UIs instead of showing the ability to *integrate in legacy software systems*. For example, MASP was evaluated by (re)-building home automation applications, and Supple was evaluated by developing a variety of simple UI dialogs. These techniques do not support an *extensible number of adaptation aspects and factors* but merely a limited number them. We also noticed that only a few works conducted *scalability* tests.

2.4 Support Tools

This section provides an overview of the state-of-the-art tools for developing (adaptive) model-driven UIs and evaluates them according to the criteria from Section 2.1.

There are a few commercial tools that partially support MDE in UI development such as ***Leonardi***[1], ***OpenXava***[2] and ***Himalia***[3]. There are several tools, which are the products of academic work. The ConcurTaskTrees Environment (***CTTE***) [53] is a tool for developing and

[1] Leonardi: www.w4.eu
[2] OpenXava: www.openxava.org
[3] Himalia: www.bit.ly/HimaliaDotNet

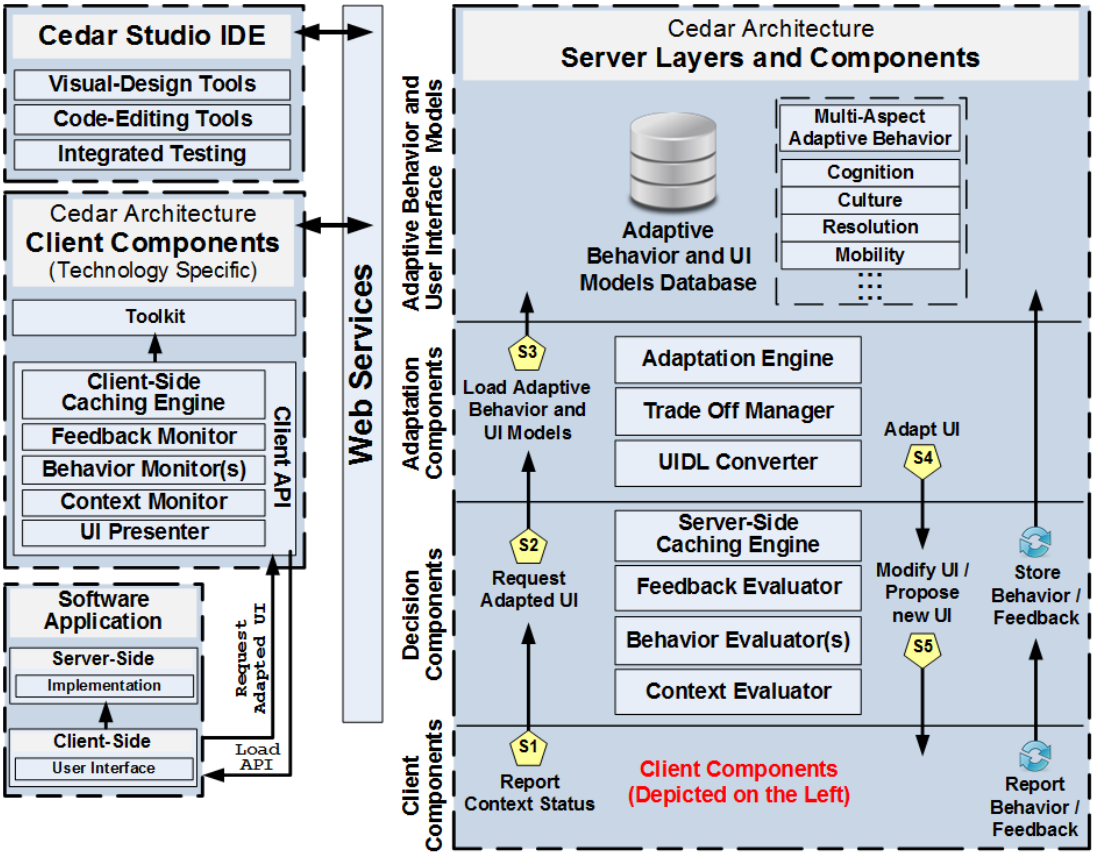


Fig. 2. The Cedar Architecture: A Reference for Developing Adaptive Model-Driven User Interfaces

analyzing task models using the CTT notation. The ***MARIA*** [54] language also has a separate authoring environment. Several tools were presented for supporting the UIDL UsiXML [55], including: ***UsiComp*** [56] and ***Xplain*** [57], ***SketchiXML*** [58], ***IdealXML*** [59] and ***GraphiXML*** [60]. A few supporting tools were also presented as part of ***MASP*** [61]. ***Gummy*** supports multi-platform graphical UI development [62]. ***Damask*** [63] is a tool for prototyping cross-device UIs. ***CIAT-GUI*** supports model-driven UI development and offers visual-design tools for the various levels of abstraction [64].

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 v*isual and code-based adaptive behavior* are only supported partially by MASP, UsiComp, CIAT-GUI, and MARIA. Also, apart from Leonardi and CIAT-GUI, the tools do not provide a mature *IDE-style UI* for easing the development process. 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, adaptive behavior reusability is not demonstrated but could be in principle possible in MARIA, MASP, CIAT-GUI, and UsiComp, which support transformation rules. Hence, we consider that these tools partially fulfill the *expressive leverage* criterion. Achieving a low *threshold and* a high *ceiling* is considered a major criticism regarding model-driven UI development tools. Therefore, building models graphically was suggested to achieve a lower threshold [65]. Damask, Gummy, and Leonardi potentially have a lower threshold than other tools since they promote a development technique that could start with the concrete UI 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. Leonardi is an exception since it is a commercial IDE, but it does not support adaptive behavior. Hence, we considered the surveyed tools to partially fulfill both the threshold and ceiling criteria. Since the UI adaptation techniques do not support the *integration in legacy software systems*, naturally the existing tools do not fulfill this criterion either.

2.5 Summary

Our evaluation of the state-of-the-art systems is aggregated and presented in Table 1. This table shows the gaps and limitations that we determined and aim to address with the work presented in this paper. Additionally, a major gap lies in the need for a feature-set minimization technique, which offers: a practical implementation, generality, and the ability to apply it at runtime. Detailed tables showing how each state-of-the-art architecture, technique, and tool fulfills each of the evaluation criteria can be found in a separate survey paper [7].

Table 1
Aggregated Evaluation

	Arch.	Tech.	Tool
Extensibility of aspects and factors	☒	◖	○
Extensibility of adaptive behavior	●	●	○
Expressive leverage	☒	☒	◖
Integration in legacy systems	○	○	○
Modeling approach	◖	●	☒
Scalability	☒	◖	☒
Threshold	☒	☒	◖
Ceiling	☒	☒	◖
User feedback	○	◖	☒
Visual/code-based representation	☒	◖	◖

● Completely Fulfills Criterion ◖ Partially Fulfills Criterion
○ Does not Fulfill Criterion ☒ Not Applicable

3 The Cedar Architecture

This section gives an overview of the components comprising the Cedar Architecture, which is based on: Cameleon Reference Framework (CRF) [66] (UI abstraction), Three Layer Architecture [67] (adaptive system layering), and MVC (implementation). This architecture comprises three server-side layers (decision components, adaptation components, and adaptive behavior and UI models), which are accessed through web-services from the client-components layer and Cedar Studio.

3.1 Layers Comprising the Cedar Architecture

The section discusses the layers comprising the Cedar Architecture, shown in Fig. 2, and offers an explanation of the components they encompass.

Client Components Layer: The components in this layer are deployed to the client machine, and are accessed through an API by the software system, e.g. enterprise application, which requires adaptation.

In "context-aware" computing, a "context" is composed of a triplet: user, platform, and environment [66]. In the Cedar Architecture, the *context-monitor* component is responsible for monitoring any changes to this triplet such as changes in the end-user's role, device, etc. This component was allocated to the client since it would be able to monitor changes to the environment, end-user's profile, and selected platform.

The *feedback-monitor* allows end-users to report their feedback on the UI adaptations presented by the system. The end-users have the ability to reverse adaptations or choose other possible alternatives.

An important part of any dynamic approach is data caching. The ability to cache data on the client allows dynamically rendered UIs to load more efficiently. The *client-side caching-engine* is responsible for caching UIs on the client machine to give interpreted models the performance of compiled code, thus providing the necessary robustness, without diminishing the ability to perform runtime adaptations.

The *UI-presenter* is responsible for presenting the UI model to the end-user in the form of a running interface using an existing presentation technology. This component also handles: event management, data binding, and validation, by linking the dynamic UI layout to the application's code-behind.

Decision Components Layer: The components in this layer are deployed to the application server to handle decision making in various adaptive UI scenarios.

The *context-evaluator* handles the information submitted by the *context-monitor* in order to evaluate whether the change requires the UI models to be adapted.

The *server-side caching-engine* assumes a role similar to that of its counterpart on the client. Yet, in this case, the caching is not made on the session level for each individual user but on the application level for all the users. The UI models cached at this level would have already been adapted. Hence, in case the same adaptation is required by a different user, the models could be loaded from the cache rather than re-performing the adaptation, which could be more time consuming.

Adaptation Components Layer: The components in this layer are deployed to the application server to perform the adaptation on the models.

The *adaptation-engine* is responsible for taking a UI model as input and adapting it by executing the relevant adaptive behavior.

Since software applications can make use of multiple adaptation aspects, trade-off is needed in certain situations, where conflicting rules make it impossible to fully meet all the constraints. In such situations, the *trade-off manager* assumes the role of balancing the trade-offs between different sets of adaptation constraints to meet each set as much as possible.

The adapted UI must be transmitted to the client in a standard format. This allows the adaptation techniques, which are based on the Cedar Architecture, to be consumed as a generic service through APIs from different applications. The format could be one of the known UI description languages (UIDLs) such as: UsiXML [55], UIML [68], etc. The *UIDL-converter* is responsible for handling the conversion between the UI model (stored as relational data) and the selected UIDL format.

Adaptive Behavior and UI Models Layer: This layer contains the various models, which are stored in a relational database hosted on the database server.

The adaptive-behavior represents a generic set of rules according to which the UI will be adapted. Such rules would be based on various adaptation aspects.

The UI is represented on multiple levels of abstraction, which were suggested by the Cameleon Reference Framework (CRF), including: task model, domain model, abstract UI (AUI), and concrete UI (CUI). The adaptive behavior could be applied on any of the UI abstraction levels. Afterwards, the UI models are transferred to the client to be presented to the end-user as a running UI.

Adaptive behavior data is needed for the adaptation procedure to make decisions on how to adapt the UI. Sections 3.2 and 3.3 discuss this data and procedure respectively.

3.2 Adaptive Behavior Data

Making decisions about the ways of adapting the UI can be obtained from a variety of sources such as *adaptive behavior* (adaptation rules) that are based on empirical studies or expert knowledge, *end-user feedback* that is obtained through the feedback-monitor, and *monitoring behavior*

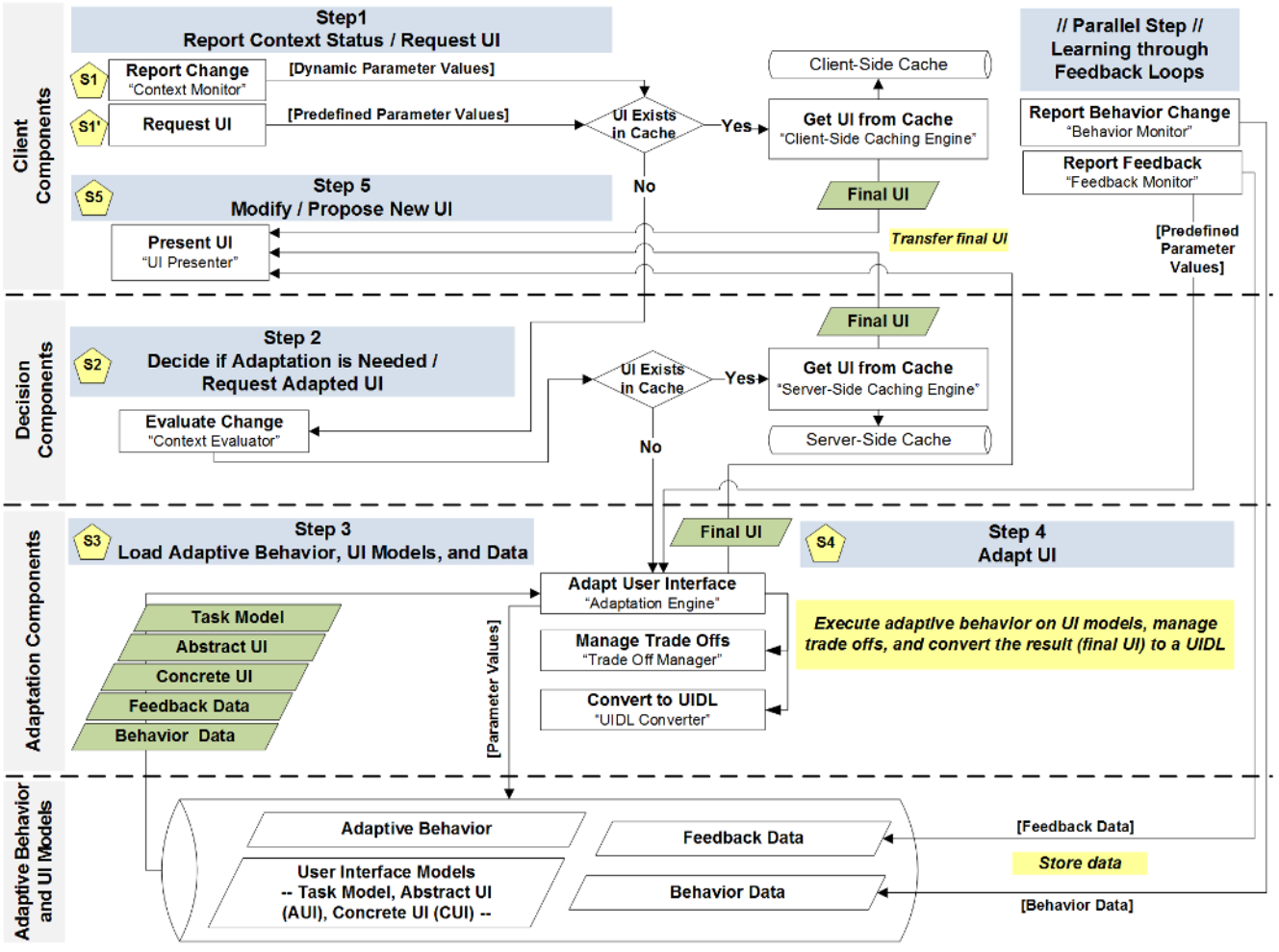


Fig. 3. User Interface Adaptation Procedure

change that is done by the behavior-monitor(s).

The *adaptive behavior* is stored in a relational database, and represents different adaptation aspects pertaining to any context-of-use dimension. Empirical studies could be conducted within an enterprise to identify how the UI should be adapted for particular adaptation aspects such as computer literacy. On the other hand, expert knowledge could be enough to define these models for other adaptation aspects such as different screen-sizes.

The client-side *monitor* components serve as feedback loops that provide the server-side layers with data for refining the predefined *adaptive behavior*. On the other hand, the server-side *evaluator* components check whether the information sent by the monitors requires a new adaptation to be performed, and whether the sent data should be used to update the *adaptive behavior*.

The *Behavior Monitor(s)* detect new situations and behavior changes on the Client Components Layer and report them to the Decision Components Layer. Some example changes that could be monitored include: an end-user's usage rate of input fields, new updates installed on the platform, information collected about the environment using sensors, etc. For example, an end-user could be initially given access to part of a UI's features. However, a behavior monitor may detect that even in this part of the UI, there are still some unused fields. In this particular example, when the end-users conduct an activity, the *Behavior Monitor* could send the identifiers of the fields and a value indicating whether the field was used to the server. Hence, the behavior monitor can trigger an update to the behavior data to indicate that these fields should be removed as well. The *Behavior Evaluator(s)* could, in this case, check whether the usage rate of certain UI elements, e.g. input fields, is low enough to exclude these elements from the UI.

The *Feedback Monitor* can collect end-user feedback on the Client Components Layer, and report it to the Decision Components Layer. One example of end-user feedback could be choosing to show features that were removed from the UI by an adaptation. In this case, the *Feedback Monitor* would send a list of identifiers representing the features that were enabled by the end-user to the server. The *Feedback Evaluator* could check whether showing one feature requires other features to be shown as well due to interdependency.

3.3 Adaptation Procedure

Some systems such as MASP [69] and MyUI [13], directly adapt UIs while the end-user is working (direct adaptation) due to the ubiquitous nature of their target applications. On the other hand, McGrenere et al. [23] promote offering the adapted UI as an alternative version (indirect

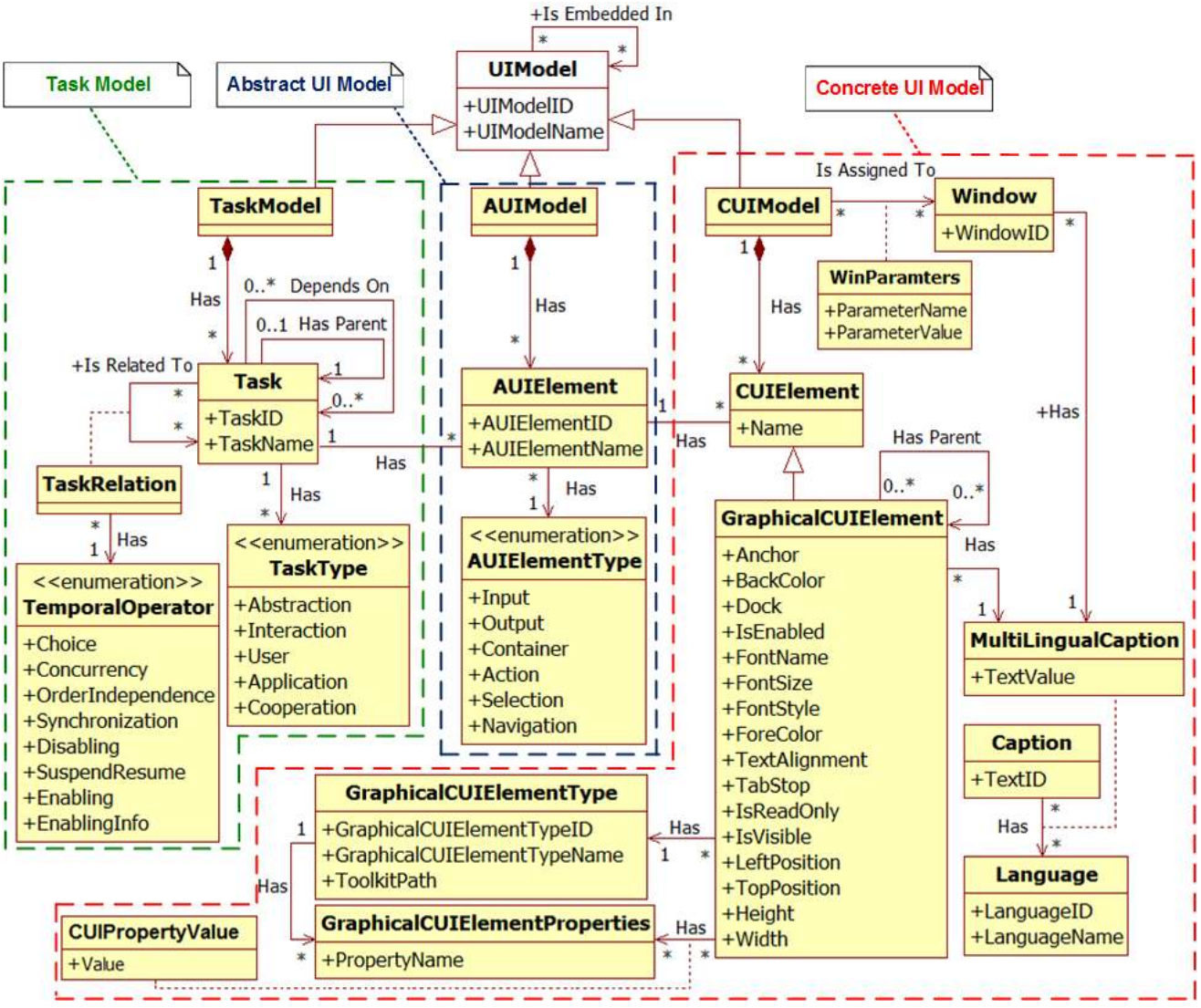


Fig. 4. Meta-Model Representing the Levels of Abstraction of a Model-Driven UI

adaptation) because direct adaptation could confuse the end-users. We think that both approaches are necessary to cater for a wider variety of adaptations. For example, if an end-user is working on a mobile phone while sitting down, and then suddenly stands up and begins to walk at a fast pace, the UI could be adapted directly to cater for this change in the context-of-use. On the other hand, if the UI requires adaptation to each end-user's computer literacy level, this level could be known and stored in the database in advance. Hence, when the end-users log into the application, they will be given access to an adapted version of the UI that meets their particular profile.

With WIMP-style UIs that are used in an office environment, e.g. with enterprise applications, proposing the adapted UI version as an alternative could be a better adaptation choice. However, our architecture supports both direct and indirect adaptation to also cater for UIs, such as the ones running on mobile phones, which have to directly adapt to an evolving context-of-use.

The adaptation procedure is shown as a part of the architecture in Fig. 2 by steps **S1** to **S5**. These steps could be mapped to the flow chart in Fig. 3, which illustrates them in more detail. A direct UI adaptation could occur once a context-change is detected by the *context-monitor* (**S1**) and reported to the *context-evaluator* in case the client-side cache does not have the necessary UI. A decision is made on whether the UI should be adapted. The server-side cache is checked for an existing version. If the required UI version does not exist in the cache, the adaptation engine is called (**S2**) for obtaining the new UI. The adaptive-behavior, data, and UI models including: Task Model, Abstract UI (AUI), and Concrete UI (CUI) are loaded (**S3**) from the database server. The adaptation procedure is then performed by applying the adaptive behavior to the UI models (**S4**). Finally, the adapted UI is sent back to the client-side in order to be presented to the end-user (**S5**). We can see that only the Final UI (FUI) is transmitted to the client-side, whereas the other UI models are used on the server-side when performing the adaptation. On the other hand, with an indirect adaptation, the UI is not adapted while the end-user is working but rather when he or she launches a UI (**S1'**). However, the adaptation procedure goes through

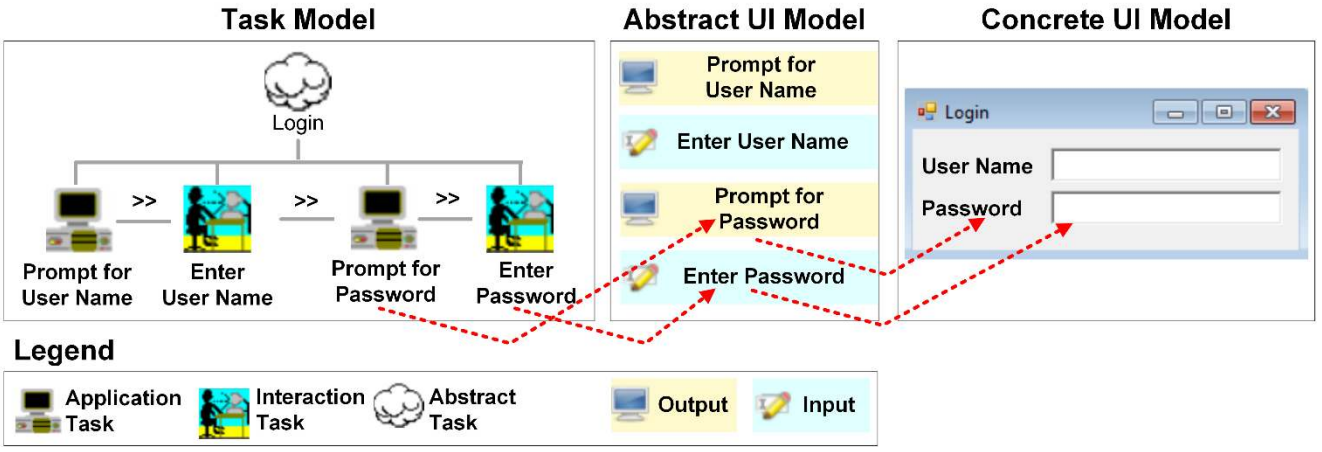


Fig. 5. An Example "Login" UI Represented on Multiple Levels of Abstraction: Task Model (Left), AUI (Middle), and CUI (Right)

the same steps, but the UI is adapted based on predefined parameters such as the end-user's computer literacy level rather than dynamic parameters that are detected by the *context-monitor* like data collected from sensors.

The adapted UIs are cached based on the parameters that prompted the adaptation. Hence, for example, if a request is issued to adapt a UI for a certain screen-size, the *client-side cache* is checked first to see if the concerned end-user had previously requested the same UI to be adapted for the same screen-size. If not, then the *server-side cache* is checked to see if another end-user had already requested the same adaptation. If both cache repositories are empty, then the adaptation operation is performed by loading the UI models from the database and executing the appropriate adaptive behavior.

The *behavior-change* and *feedback* monitor(s) constantly run in parallel with the other functionality. Once a behavior change is detected, the new data is stored on the database-server. This data is collected over time and processed in order to refine the adaptations. Feedback submitted by the end-users is also stored on the database server in order to refine the adaptations. Since the end-users are manually submitting the feedback, the adaptation engine is called after storing the feedback data in order to directly reflect the change.

3.4 Representing Model-Driven UIs

The meta-model in Fig. 4 shows our representation of the UI levels of abstraction suggested by the Cameleon Reference Framework (CRF) [66], namely the task, abstract UI (AUI), and concrete UI (CUI) models. We should note that we are not claiming that this meta-model is the most comprehensive, but that it contains the concepts required for supporting our UI adaptation technique. The example in Fig. 5 depicts a "Login" UI represented on three levels of abstraction, and shows how the elements on one level can be mapped to their counterparts on another level. The following subsections explain the meta-model shown in Fig. 4.

3.4.1 Task Models

We adopted the ConcurTaskTrees (CTT) [70] notation for depicting the *Task Models*, which represent the activities that the UI is required to support.

Following the CTT notation, a *Task Model* is composed of several *Tasks* each having a type from those enumerated by *Task Type*. The *Tasks* are connected in a hierarchical manner to indicate parent/child relationships. The *Tasks* are also connected to each other with temporal relationships (*Task Relation*) indicating interdependency between them. For example, a calculated field can depend on values from other fields. Each relation can be assigned one of the temporal operators, which were specified by CTT [71] such as: choice, concurrency, etc.

3.4.2 Abstract User Interface (AUI) Models

The *AUI Model* is a modality-independent representation of the user interface. It is composed of *AUI Elements*, each of which has a type from those enumerated by the *AUI Element Type*. These elements could be grouped inside the model using container elements. Each *AUI Element* could be mapped to many *Tasks* in the *Task Model* and vice-versa.

3.4.3 Concrete User Interface (CUI) Models

The *CUI Model* is a modality-dependent representation of the UI. The *CUI Elements* can be of different types, each representing a certain modality such as: graphical, character, voice, etc. However, since most applications use graphical UIs (GUIs), we only define a *Graphical CUI Element* (widget) class as a specialization of the *CUI Element* class. Nevertheless, the meta-model can be extended in the future for supporting other modalities.

A *Graphical CUI Element* has standard properties (e.g., height, width, etc.), which are common for all graphical UI widgets. These elements also define *Graphical CUI Element Properties*, which depend on each element's type. For example, a data-grid widget could have a property called "alternating row color", which is not defined in other widgets. The names of such type-dependent properties are stored as a string in the *PropertyName* attribute of the *Graphical CUI Element Properties* class. Developers using Cedar Studio can specify values for these properties. These values are stored as a string in the *CUI Property Value* association class. By default, the value of each of these properties is "null", and is ignored by the *UIPresenter*. In case the value is not "null", the *UIPresenter* is responsible of casting it to the appropriate type and assigning it to the property of the widget it is presenting from the selected toolkit.

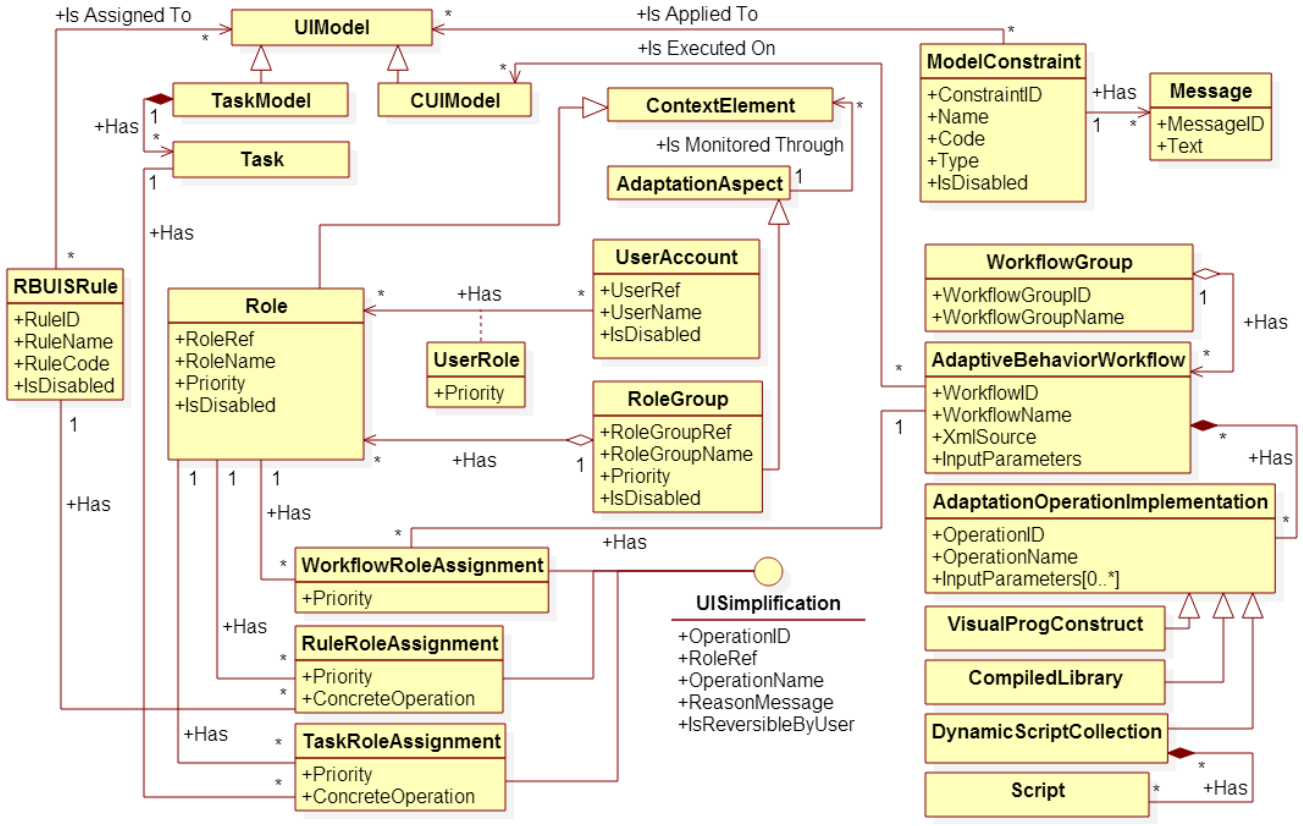


Fig. 6. Class Diagram Depicting the Concepts of RBUIS Including both Feature-Set Minimization and Layout Optimization

We adopted a relative positioning approach for the UI layout, whereby *Graphical CUI Elements* can be embedded inside one another, e.g. text-boxes inside a group-box, and positioned using the top and left position properties. This approach is supported by many presentation technologies such as: HTML, Java Swing, and Windows Forms. To support multi-lingual UIs, *Graphical CUI Elements* are assigned a *Multilingual Caption*.

Each *Graphical CUI Element* has a *Graphical CUI Element Type* such as: button, text-box, etc. These types define a *Toolkit Path* property, which indicates where the UI Presenter (Section 3.1) component could locate the relevant widget inside the toolkit. For example, if Windows Forms is adopted, the Toolkit Path property can store the widget's assembly path (e.g., "System.Windows.Forms.Button"). End-users can access the UI by activating *Windows* through links in a navigation structure such as a menu.

3.4.4 Mapping the UI Levels of Abstractions

It is important to connect the different UI levels of abstraction. This connection is done by mapping the elements on each level to the corresponding elements on another. It is important to allow developers to define and adjust these mappings [72], [73].

As depicted by the meta-model in Fig. 4, there are one-to-many relationships connecting the *Task* to the *AUI Element* and the latter to the *CUI Element*. Since we are storing these models in a relational database, these one-to-many relationships are translated by copying the primary keys from the entities on the one side as foreign keys in the entities on the many side. These keys provide the necessary traceability between the elements of the different levels of abstraction. Cedar Studio allows developers to define and modify these mappings, as will be explained later in the paper. Although in many cases, such as the example shown in Fig. 5, the relationships between the model elements is one-to-one, there are many cases where they are one-to-many, hence the latter was our design choice. Consider the example shown in Fig. 5, the "prompt for user name" and "enter user name" tasks could be kept as one abstract task, which is mapped to the same two AUI input and output elements. Similarly, for example, an output element on the AUI model could be mapped in certain cases to two graphical widgets on the CUI model. One example can be that of a read-only sales invoice total, which could be represented as one output element on the AUI model and mapped on the CUI model to both a label displaying the name of the field and a read-only textbox containing the value.

4 Role-Based User Interface Simplification

This section presents Role-Based UI Simplification (RBUIS), a mechanism for improving usability by providing end-users with a minimal feature-set and an optimal layout based on the context-of-use. RBUIS is a UI adaptation technique, which realizes the components of the Cedar Architecture shown in Fig. 2.

We define a feature as a functionality of the software system and a minimal feature-set as the UI sub-set with the least features required by an end-user to perform a job. An optimal layout is the one that maximizes the

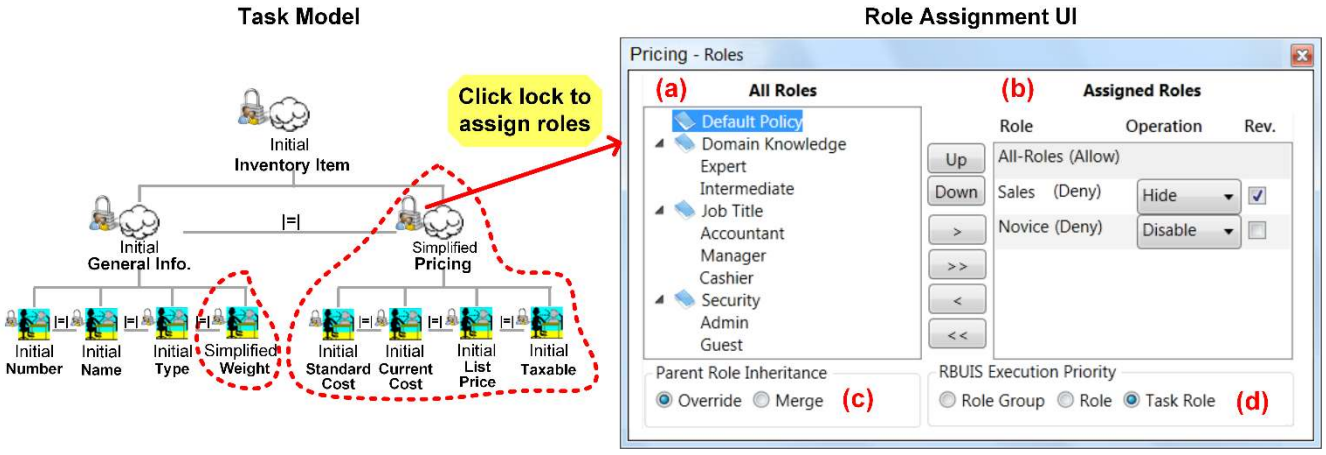


Fig. 7. Assigning Roles to Tasks in Order to Achieve Feature-Set Minimization (Inventory Item UI – Example)

(a) All the role groups and roles except those that are already assigned **(b)** The roles assigned to the task at hand **(c)** Specifying whether the task overrides or merges the inherited role assignments from its parent task **(d)** Specifying the source of the role priority

satisfaction of the constraints imposed by a set of aspects. An optimal layout is achieved by adapting concrete widget properties such as: type, grouping, size, location, etc. RBUIS is based on the Cedar Architecture, which we presented in Section 3, and implements several of its high-level components.

4.1 Utilization of Roles in RBUIS

RBUIS utilizes the concept of *roles* from the standard for role-based access control (RBAC) [74]. In RBAC, *resources* can be made accessible only for certain roles. Hence, upon assigning a role to a *user*, he or she will gain access to all the resources that are accessible by this role.

In RBUIS, UIs and their related adaptations are treated as resources, which are only made accessible to end-users holding *User Accounts* that were allocated particular roles. However, the concept of a *Role* in RBUIS is not ultimately related to security but to usability. The class diagram illustrated in Fig. 6 depicts the utilization of roles in RBUIS. A *Role* is used to indicate a member of a *Role Group*, which represents an *Adaptation Aspect* on which basis the UI can be simplified. Some example adaptations aspects could be the end-user's computer literacy level, the device's screen-size, and the environment's brightness. A *Role* is neither only related to the job that the end-user performs, nor is it only related to one dimension of the context-of-use, but it can represent *Context Elements* related to any of the three context-of-use dimensions: "user", "platform" and "environment". For example, if one would like to adapt a UI based on each end-user's computer literacy, a possible *Role Group* could be "computer literacy level" with the following *Roles*: novice, intermediate, and expert. An example *Role Group* that is related to the "platform" is device screen-size, with the following *Roles*: 5.5 inch mobile phone, 10 inch tablet, 17 inch laptop, etc. An example *Role Group* that is related to the "environment" dimension of the context-of-use could be brightness level with the following *Roles*: high, medium, and low. Nevertheless, it is possible for a *Role Group* used for representing UI *Adaptation Aspects* to contain roles traditionally used for security purposes. For example, a job title *Role Group* including the *Roles* secretary and accountant, can be used to offer end-users holding different jobs an adaptation of the UI, which allows them to accomplish their daily tasks with a higher efficiency and effectiveness.

The allocation of *Roles* to *User Accounts* can be done in two different ways depending on the *Role* at hand. For example, *Roles* belonging to *Role Groups* such as computer literacy and job title can be allocated to a *User Account* when it is created. These role allocations can be manually modified at a later stage if, for example, the end-user's computer literacy level or job title change. When an end-user logs into the system with a *User Account* that has one or more *Roles*, the adaptive behavior relevant to these *Roles* is executed on the UI models in order to adapt the UI to the context-of-use. On the other hand, *Roles* belonging to *Role Groups* such as device's screen-size and environment's brightness are usually only known at runtime. Hence, these *Roles* are allocated dynamically to a *User Account* for an individual session based on data reported by the *Context Monitor* component of the Cedar Architecture (Fig. 2). For example, the *Context Monitor* can obtain the device's screen-size from the operating system, and it can detect changes to the brightness in the environment by reading data from a sensor. These changes to the context-of-use are then reported to the server-side layers of the Cedar Architecture to dynamically perform *Roles* associations, and to execute the adaptive behavior pertaining to these *Roles*, on the UI models.

4.2 UI Simplification Operations in RBUIS

RBUIS realizes several of the Cedar Architecture components previously illustrated in Fig. 2 and explained in Section 3. The class diagram illustrated in Fig. 6 presents the main concepts of RBUIS. These concepts cover the two supported UI adaptation types namely, feature-set minimization that is applied on task models, and layout optimization that is applied to the concrete UI models.

There are three types of operations, which realize the *UISimplification* interface:

1. ***Task-Role-Assignments*** are used to achieve context-dependent feature-set minimization. *Roles* can be allocated to *Tasks* in *Task Models* to indicate that these

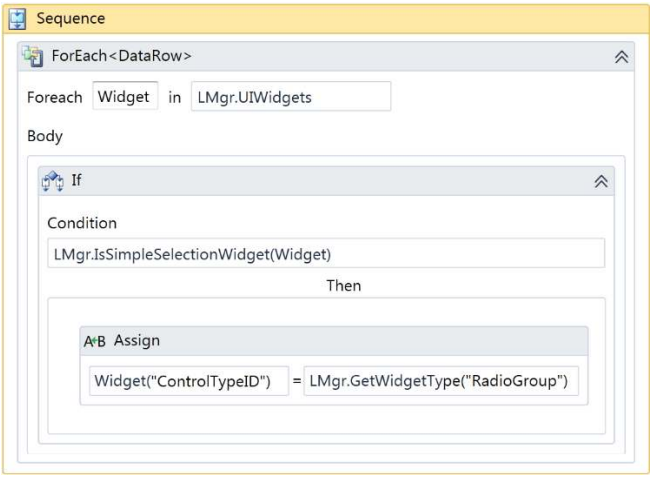


Fig. 8. Example of an Adaptive Behavior Workflow that Changes the Simple Selection Widgets to Radio-Button-Groups

tasks should be blocked for *User Accounts* that are assigned these roles.

2. ***Workflow-Role-Assignments*** are used to achieve context-dependent layout optimization. The notion of a workflow is used in this paper for referring to adaptive behavior represented using programming constructs (e.g., control structures). For example, a workflow for switching combo-boxes with list-boxes could contain: (1) a "for" loop that iterates on the CUI widgets, (2) an "if" statement nested in the "for" loop to check if a widget is a combo-box, and (3) an assignment nested in the "if" statement to change the widget-type from "combo-box" to "list-box". Roles are allocated to *Adaptive Behavior Workflows*, which are executed on *CUI Models* to adapt their characteristics for *User Accounts* that are assigned these roles. An *Adaptive Behavior Workflow* is composed of *Adaptation Operation Implementations*, which can be: *Visual Programming Constructs*, *Compiled Libraries* or *Dynamic Scripts*. *Adaptive Behavior Workflows* are grouped under *Workflow Groups*, each representing a particular adaptation factor. For example, a *Workflow Group* called "UI Grouping" could contain three *Adaptive Behavior Workflows*, which adapt the UI's widget grouping to: tab-pages, group-boxes, or sub-windows.
3. ***Rule-Role-Assignments*** are used to achieve context-dependent feature-set minimization and layout optimization. An *RBUIS Rule* is represented by code instructions that dynamically select *Tasks* to be blocked or *CUI Models* to be adapted based on certain conditions. RBUIS rules are assigned *Roles* to indicate the *User Accounts* to which they apply.

Defining the adaptive behavior and configuring the role-assignments is done after deploying the software in the enterprise, and can be a joint effort between personnel from the software company and the enterprise's IT department.

Model Constraints are supported by RBUIS, primarily for verifying possible human errors in the role assignments. For example, someone might mistakenly perform a task-role assignment, which undesirably blocks a task from all the users. By executing these constraints against the models and the role-assignments, our supporting tool is able to issue appropriate warnings and errors.

We adopted the Structured Query Language (SQL) for defining RBUIS Rules and Model Constraints since many developers and IT personnel are familiar with its syntax. Also, it is easy to query UI the models and adaptive behavior using SQL since they are stored in a relational database.

4.3 Feature-Set Minimization

Following the concept of multilayer UI design [36], RBUIS creates separate layers containing various features, and end-users can gain access to these layers at different stages. Before RBUIS is applied, the UI is initially designed based on the least constrained context-of-use. From the feature-set perspective, the initial UI design can be considered as a layer containing all the features. Additional layers are created with RBUIS when access is revoked by allocating roles to tasks. For example, a novice user can be initially given access to a layer containing a primitive subset of the tasks, and later promoted to a more advanced layer by changing his or her role from novice to expert.

When a task is blocked based on roles, a property called *concrete operation* (Fig. 6 - *TaskRoleAssignment* class) is used to specify how the blocking is translated on the CUI level. A task can become: invisible, disabled, faded until first use, etc. The task model is mapped to the AUI and then to the CUI to get the UI widgets relevant to the task(s) to which the role(s) were allocated.

The task model example illustrated in Fig. 7 (left) represents part of a UI for managing inventory items. With RBUIS, each task is presented with a lock-shaped button next to it. Upon clicking any of these buttons, the window illustrated in Fig. 7 (right) is launched to allow the assignment of roles to the task. The UI shows a list of all the role-groups and their roles (Fig. 7 – a). Upon assigning one or more roles to a task (Fig. 7 – b) its status changes from "Initial" to "Simplified". The tasks shown in Fig. 7 (left) encircled with a dashed line were allocated roles. In the case of the "Weight" task, the roles were directly assigned to it, whereas tasks under "Pricing" inherit their role allocation from their parent task. Since sub-tasks inherit the access rights of the parent tasks, the person performing the task-role-allocation can specify whether a child task overrides or merges its own access rights with those inherited from its parent (Fig. 7 – c).

Since each user can be simultaneously allocated multiple roles, e.g. novice and sales officer, a certain order should be followed at runtime to perform the elimination. The class diagram in Fig. 6 shows priorities on different levels. The person in charge of the role allocation can specify the source, e.g. "RoleGroup", "Role", "Task-Role", or "UserRole", from which the priority is read (Fig. 7 – d). By default, task-based assignments have a higher priority than rule-based ones unless specified otherwise. We consider this to be the case because a rule can be applied to a large set of tasks at once, whereas manually assigning a role to a task is more explicit.

In comparison to several other approaches, e.g. Roam [50], RBUIS is more general in the sense that it does not focus one adaptation aspect, e.g. display size, but supports the extensibility of adaptation aspects and factors. Furthermore, RBUIS supports the definition of adaptive behavior using both a visual and a code-based represen-

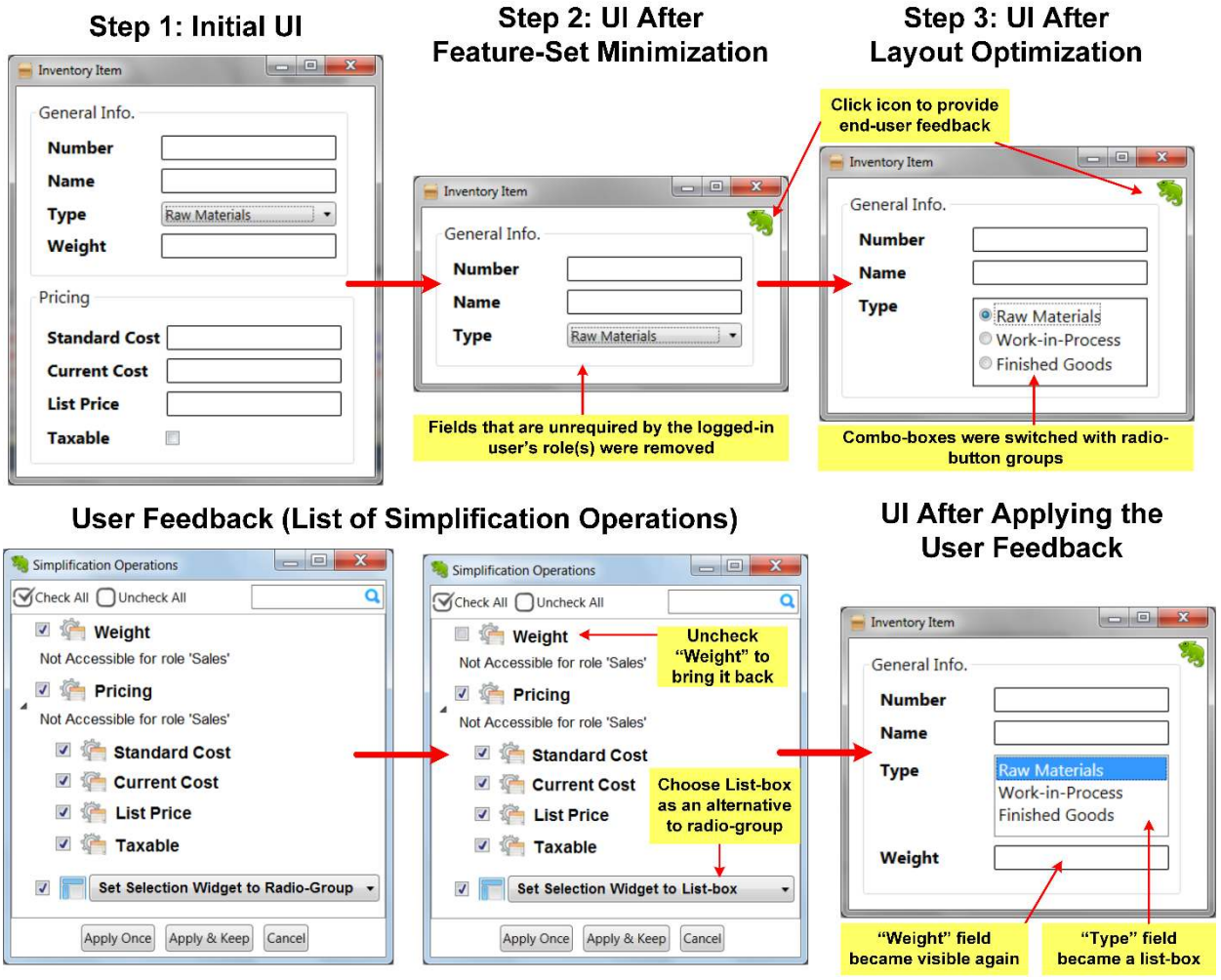


Fig. 9. Feature-Set Minimization, Layout Optimization, and User Feedback (Inventory Item UI – Example)

tation, e.g. using workflows.

4.4 Layout Optimization

As discussed in Section 4.3 on feature-set minimization, the UI is initially designed based on the least constrained context-of-use. From the layout perspective, the properties of the CUI model elements (e.g., height, width, type, etc.) are chosen to fit the context-of-use that requires the least adaptation. Since it is possible to use multiple adaptation aspects, the initial UI design can be the least constraint for all aspects in consideration. For example, in terms of the aspects related to the platform part of the context-of-use, a platform can be considered more constraint if it has a lower resolution, supports a limited number of widgets, becomes less usable with certain widgets due to the lack of a keyboard, etc. [41]. Hence, for example, a UI can be initially designed based on the screen-size of a desktop computer and later optimized for a mobile phone screen. The choice of which context-of-use is considered the least constraint is based on the designer's perspective.

Layout optimization is achieved by executing adaptive behavior workflows on the CUI models to adapt the properties of their widgets. As depicted by the class diagram in Fig. 6, adaptive behavior workflows can encompass a mixture of visual and code-based constructs. We implemented adaptive behavior workflows using the Windows Workflow Foundation (WF), which is part of the .NET Framework. An example of an adaptive behavior workflow is illustrated in Fig. 8. It shows visual programming constructs including a "for-loop" iterating around a UI's widgets. Inside the "for-loop", there is an "if-condition", which checks whether or not the type of the widget is a simple selection widget (e.g., combo-box, list-box, etc.), in order to change it to a radio-group. The checking is done using a helper class called "LMgr" (layout manager), which has a method called "IsSimpleSelectionWidget". This helper class was defined in a C# library and used to extend the functionality of the adaptive behavior workflows. We should note that since the adaptive behavior workflows use WF, they can support more advanced examples than the one shown in Fig. 8.

The visual programming constructs supported by WF can be extended using external compiled class libraries developed in a .NET language such as C# or VB.NET. One extension that we created is a construct that calls a dynamic script. Currently, this construct supports Iron Python code, but it can be extended in the future to support other languages.

The adaptive behavior workflows are executed on the CUI model by the *Adaptation Engine* depicted as part of

Listing 1. Feature-Set Minimization Algorithm (Excerpt)

```
1.Simplify-Task(TaskID,UserRoles[],TaskRoles[],UIModel)
2. //Determine the primary role
3. foreach ur in UserRoles
4.   tr ← TaskRoles.GetRole(ur.RoleRef)
5.   if tr = null then tr ← TaskRoles.GetRole(All-Roles)
6.   ur.Priority ← tr.Priority;
7. UserRoles.OrderBy(Priority)
8. PrimaryRole ← UserRoles.First()
9. if PrimaryRole.RoleRef ≠ All-Roles
10. //Apply the concrete operation to the CUI model
11. blockedAUI←GetBlockedAUI(TaskID,UIModel.TMToAUIMap)
12. blockedCUI←GetBlockedCUI(blockedAUI,UIModel.UIToCU
    IMap, UIModel.CUI)
13. foreach element in blockedCUI
14.   switch PrimaryRole.ConcreteOperation
15.     case Hide: element.Visible ← false; break;
16.     case Disable: element.ReadOnly ← true; break;
17.     case Protect: element.ReadOnly ← true;
18.                   element.MaskChar ← '*'; break;
19.     case Fade: element.Opacity ← '30%'; break;
```

the Cedar Architecture in Fig. 2. Afterwards, the Final UI (FUI) is transferred to the client as an XML file and presented dynamically to the end-user by the *UI Presenter*, as shown by the adaptation procedure in Fig. 3. As is the case of feature-set minimization, selecting which workflows to apply for a particular end-user with multiple roles depends on priorities.

4.5 Implementation and UI Simplification Examples

This section demonstrates the algorithms behind RBUIS with their running times and a few examples.

The following example demonstrates the feature-set minimization process assuming the priorities were set at the "TaskRole" level.

- *UserA*: Novice, Sales Officer
- *TaskX*: **1.** All-Roles (Allow)
  2. Cashier (Deny- Hide)
  3. Novice (Deny-Disable)

An excerpt of our feature-set minimization algorithm is shown in Listing 1. Following this algorithm "UserA" has access to "TaskX" since the role "Sales Officer" has the highest priority. In contrast, if the role "Novice" had a higher priority than "All-Roles", then "UserA" would have been denied access to "TaskX" hence disabling its related CUI elements as indicated by the concrete operation next to the role "Novice".

The running time of our algorithm is polynomial: $O\ (m \times (n \times l \times p \times (2\ j \log j + k) + n))$, where **m** is the number of task models, **n** is the number of tasks in a task model, **j** is the number of user roles, **k** is the number of blocked CUI model elements for a task, **p** is the number of parent tasks for a task, and **l** is the number of task roles. The full algorithm and complexity analysis can be found in a technical report [75]. The part of this algorithm, which uses CTT temporal constraints to check for conflicts that could occur when features are removed, also has a polynomial running time of $O\ (m)$, and can be found in a separate paper [17].

An excerpt of our layout optimization algorithm is shown in Listing 2. This algorithm is responsible for selecting the relevant workflows that are to be executed on a UI model for a particular end-user. The running time

Listing 2. Layout Optimization Algorithm (Excerpt)

```
1.Optimize-Layout (UserRoles[],Roles[],UIModel,LayoutID)
2. //Determine the primary role
3. foreach ur in UserRoles
4.   tr ← Roles.GetRole(ur.RoleRef)
5.   if tr = null then tr ← Roles.GetRole(All-Roles)
6.   ur.Priority ← tr.Priority
7. UserRoles.OrderBy(Priority)
8. PrimaryRole ← UserRoles.First()
9. WorkflowsToExecute[] ← GetWorkflows(
   PrimaryRole,LayoutID)
10.WorkflowsToExecute.OrderBy(ExecutionOrder)
11.//Execute workflows
12.foreach workflow in WorkflowsToExecute
13. //Execution time depends on the workflow's content
14. workflow.Execute(UIModel)
```

Listing 3. Iron Python Script for Switching Combo-Boxes with Radio-Groups if the Combo-Box Contains less than Three Items

```
1.  import sys
2.  def AdaptUI(UIModelMgr):
3.     for widget in UIModelMgr.UIDataSet.UIWidgets.Rows:
4.        if str(widget["ControlType"])== "ComboBox":
5.           if str(widget["DataSource"])!= "":
6.              if UIModelMgr.GetListItems(int(
                widget["DataSource"])).Rows.Count< 3:
7.                 widget["ControlType"] = "RadioGroup"
```

of this algorithm is polynomial: $O\ (2\ m \log m + 2\ n \log n)$, where **m** is the number of user roles, **n** is the number of workflows to be executed.

As we indicated in Section 4.4, dynamic Iron Python scripts can be embedded in adaptive behavior workflows. The algorithm presented in Listing 3 is an example of an Iron Python script, which switches combo-boxes with radio-button groups, in case the combo-box contains less than three items.

Continuing with the inventory item UI example previously shown as a task model in Fig. 7 (left), an initial non-adapted version of this UI is illustrated in Fig. 9 – Step 1.

After applying the feature-set minimization, the UI's features are reduced in this example by hiding the unrequired widgets as shown in Fig. 9 – Step 2. Then, an adaptive behavior workflow like the one presented in Fig. 8 is executed on the UI with reduced features in order to change the simple selection widgets (e.g., combo-boxes) to radio-button-groups. After the execution of this workflow, the "Type" combo-box is replaced with a radio-button-group replacing all the combo-box-items with radio-buttons as illustrated in Fig. 9 – Step 3.

4.6 User Feedback on the Adapted UI

As mentioned in Section 2.1, supporting user feedback in adaptive software systems provides end-users with awareness of automated adaptation decisions, and the ability to override them when needed. It can also increase the end-users' feature-awareness and control over the UI. Meta-UIs have been suggested as a set of functions, which provide end-users with the ability to control and evaluate the state of a UI [76]. For example, meta-UIs have been used for allowing end-users to understand ubiquitous UIs, and control operations such as adaptation [77]. In a similar sense, RBUIS provides end-users with the ability

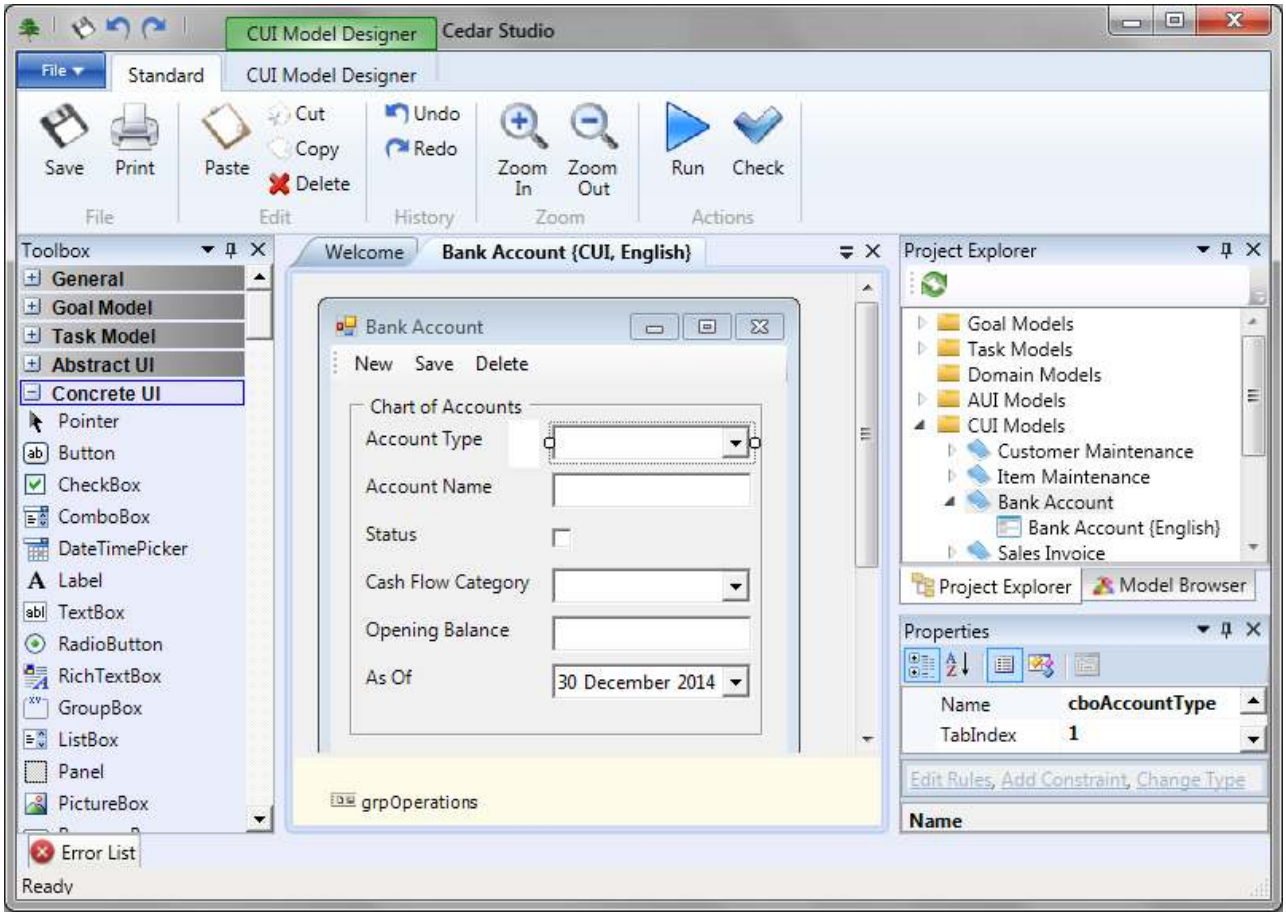


Fig. 10. Cedar Studio CUI Model Design Tool Showing a 'Bank Account' UI

to list all the simplification operations that have been performed on a UI, and the ability to reverse them or choose alternatives when possible.

The class diagram in Fig. 6 shows a UML interface called *UISimplification*, which is realized by all adaptation operations in RBUIS. This interface defines attributes called *ReasonMessage* and *IsReversibleByUser*, which are used by RBUIS's user feedback mechanism to indicate the reason behind the simplification and whether end-users can reverse it.

Each UI adapted using RBUIS is extended with a chameleon icon in its top right corner (e.g., Fig. 9). By clicking this icon, the end-users gain access to a list of the simplification operations that have been applied to the UI. An example list is shown in Fig. 9 based on the inventory item UI example. End-users can uncheck any reversible feature-set minimization or layout optimization operation. They can also choose from multiple possible layout optimization alternatives. These alternatives are the adaptive behavior workflows, which have been assigned to the same workflow group (refer to class diagram in Fig. 6). For example, a workflow group can contain workflows for changing the type of the simple selection widget (e.g., combo-box, list-box, etc.).

End-users can apply the feedback changes for one time only or store them for future use. After an end-user applies the changes, a request will be sent to the server to re-simplify the UI and exclude unchecked simplification operations. If an operation is set to be irreversible by end-users, for example due to security reasons, the check-box pertaining to this simplification becomes disabled, and the end-user gets notified of the reason.

In case end-users try to enable features that depend on disabled features, they are informed that this dependency dictates that those features should be enabled as well. The dependency among features is determined based on the CTT temporal operators. The recursive relationship "*Depends On*" on the *Task* class in the meta-model shown in Fig. 4 indicates the dependencies among tasks.

5 Cedar Studio

Cedar Studio is an Integrated Development Environment (IDE), which allows technical stakeholders such as developers and IT personnel, to develop adaptive model-driven UIs, which leverage RBUIS (Section 4) and are based on the Cedar Architecture (Section 3).

As presented earlier in Section 2, there are several gaps in the existing adaptive model-driven UI development tools. Cedar Studio is an initiative for filling these gaps. It offers technical stakeholders, e.g. developers and IT personnel, the ability to devise model-driven UIs using easy-to-use tools similar to those provided by existing IDEs such as: Visual Studio.NET, Eclipse, etc. With the existence of tool support, the adoption rate of model-driven UIs could increase. This paper merely provides a brief overview of Cedar Studio. More infor-

mation about this IDE can be found in a separate paper [16]. It is also possible to observe Cedar Studio in operation by watching its demonstration videos online [78].

5.1 Overview of Cedar Studio's Features

Cedar Studio offers several visual-design and code-editing tools that allow stakeholders to create and manage all the artifacts associated with RBUIS and the Cedar Architecture. This IDE can be used during development (design-time), deployment and post-deployment phases. The deployment phase is the stage where the software that requires adaptation is being installed for the first time and configured. The post-deployment phase is the stage of maintenance and further configuration after the software has been deployed. The UI models are created at development-time, and the adaptive UI behavior can be added and maintained during the deployment and post-deployment phases.

Visual-design tools are provided for creating and managing the UI models (refer to the meta-model in Fig. 4), which the Cedar Architecture (Fig. 2) adopts from the Cameleon Reference Framework (CRF). These models include: (1) task models, (2) domain models, (3) abstract UI (AUI) models, and (4) concrete UI (CUI) models. Cedar Studio's CUI model visual-design tool is illustrated as an example in Fig. 10.

In order to save time and effort, stakeholder using Cedar Studio can automatically generate one level of abstraction from another. For example, in a top-down design approach, the AUI model can be generated from the task model, and the CUI model can be generated from the AUI model. The inverse can be done in a bottom-up approach. One step has to be conducted manually, and that is the specification of the mappings between the elements of one level of abstraction and another. For example, we can specify that an AUI input element should be mapped to a text-box on the CUI model. Cedar Studio has default mappings between the levels of abstraction. It also provides a UI through which the default mappings can be overridden. Manual changes that occur after the automatic generation on any level of abstraction can be automatically synchronized to the other levels.

Cedar Studio supports a combination of visual-design and code-editing tools for implementing adaptive UI behavior using RBUIS. These tools include: (1) *visual adaptive behavior workflows* (Fig. 8) and (2) *dynamic scripts* for optimizing a UI's layout, (3) *visual role-assignments* and (4) *code-based rules* for minimizing a UI's feature-set to a particular context-of-use, and (5) *SQL-based model constraints* used for model verification.

5.2 Testing Cedar Studio

Cedar Studio was assessed by constructing UIs based on examples from existing enterprise applications, and applying adaptive behavior to them. The UIs we constructed are data entry interfaces for managing different types of records including: bank accounts, customers, inventory items, sales invoices, etc. The adaptations that we applied to these UIs represent practical scenarios, which can potentially occur in real-life enterprise systems.

One of the main observed strengths of using Cedar Studio in practice is in its AUI, CUI, and Workflow design-tools, which are based on existing mature Visual Studio components. The task model design-tool can be developed further to reach the same level of maturity, and the code editors can be enhanced by providing additional functionality such as intelligent-sense.

Sections 6 and 7 present an evaluation of the contributions made in this paper, from the technical and the human perspectives respectively.

6 Technical Evaluation

This section presents a method for integrating RBUIS's adaptive UI capabilities in legacy software applications without a major integration effort. We evaluated this method by establishing and applying technical metrics to measure several of its properties using scenarios from the open-source enterprise application Apache Open for Business (OFBiz).

6.1 Integrating RBUIS in Legacy Software Systems

The server-side components presented by the Cedar Architecture (Fig. 2) can be accessed through web-services by any software system that needs to leverage RBUIS's adaptive UI capabilities. However, software systems have to integrate the client-components of the Cedar Architecture within them to be able to communicate with the server-side components and benefit from RBUIS. In case a software system is being newly developed, it can be designed to make use of RBUIS. However, a technique is needed to allow mature legacy systems to perform this integration without a major effort. Furthermore, it is likely that legacy applications adopt a code-based UI representation rather than a model-driven UI one. Hence, the integration technique has to provide means for reverse-engineering the code-based UIs into a model-driven representation based on the meta-model illustrated in Fig. 4.

Since OFBiz represents its UIs using HTML, we developed a JavaScript version of the Cedar Architecture's client components that would work with HTML-based UIs. We also devised a technique for reverse-engineering HTML forms into a model-driven representation. This technique adopts a bottom-up approach, whereby it generates an XML document representing the HTML UI. This document is imported into Cedar Studio to create the CUI model and generate the upper levels of abstraction from it. Although the UIs are reverse-engineered at design-time, our technique launches the HTML UI and generates the XML document using JavaScript to include any HTML elements that might have been generated by server-side code (e.g. ASP.NET, JSP, etc.).

By using our integration method, legacy applications, e.g. OFBiz, only need to add a few lines-of-code globally to leverage RBUIS's adaptive UI capabilities. These few lines-of-code will load the client-side API of the Cedar Architecture, use it to call the web-service and access the server-side components implemented by RBUIS, and apply the obtained result on the running HTML page.

The steps presented in Fig. 11 explain the process of reverse-engineering legacy UIs into a model-driven representation, and the empowerment of these systems with adaptive UI capabilities by integrating RBUIS into them. The process is demonstrated using excerpts of our algorithms and a simple login UI as an example.

Step1: Generate XML from HTML and Use XML to Generate UI Levels of Abstraction

(a) Example HTML UI Login Form (Excerpt)
This simple example assumes that the UI is from a legacy system. Similar to this example, OFBiz uses HTML tables to represent its data entry UIs (forms).

User Name
Password

```
<table id="frmLogin">
  <tr>
    <td>User Name</td>
    <td><input type="text" id="txtUserName"/></td>
  </tr>
  <tr>
     <td>Password</td>
     <td><input type="password" id="txtPassword"/></td>
  </tr>
</table>
```

(b) Algorithm for Generating XML from HTML Table (Excerpt)

```
 1. function ConvertHTMLTableToXml(TableID) {
 2.   var xml = "";
 3.   $("#" + TableID + " tr").each(function () {
 4.   var cells = $("td", this);
 5.   /*Parse Cells*/
 6.   for(var cellCtr=0;cellCtr<cells.length;++cellCtr){
 7.      var inputs = $("input", cells.eq(cellCtr));
 8.      /*Parse Input Fields*/
 9.      for(var inpCtr=0;inpCtr<inputs.length;++inpCtr){
10.        var fieldType=inputs.eq(inpCtr).attr('type'),
11.        fieldID = GetFieldID(inputs.eq(inpCounter)),
12.        element = GetElement(fieldID);
13.        /*Generate XML for Element*/
14.        var xmlInput = GetInputFieldXml(element,
           fieldType, fieldID) + "\n";
15.        xml += xmlInput; } } }
16.   return xml;
17. }
```

The JavaScript algorithm above parses an HTML UI and generates an XML representation similar to the one below.

(c) XML Generated from HTML UI (Excerpt)

```
<CUI Name="frmLogin">
  <GraphicalCUIElement Name="lblUserName" Text="User Name"
  Visible="True" Type="Label" Top="0" Left="0" FontSize="12"/>
  <GraphicalCUIElement Name="txtUserName" Visible="True"
  Type="TextBox" Top="0" Left="100" FontSize="12"/>
  <GraphicalCUIElement Name="lblPassword" Text="Password"
  Visible="True" Type="Label" Top="40" Left="0" FontSize="12"/>
  <GraphicalCUIElement Name="txtPassword" Visible="True"
  Type="Password" Top="40" Left="100" FontSize="12"/>
</CUI>
```

The XML is imported into Cedar Studio, and the UI models are generated using a bottom-up approach, CUI to AUI and AUI to task model. The resulting UI models for this example will be the ones previously presented in Fig. 5.

Step 2: Integrate RBUIS in Legacy System and Use RBUIS's Adaptive UI Capabilities

(d) Code for Enabling Adaptive UI Capabilities in Legacy Systems

```
   //Load the API Scripts
1: <script type="text/javascript"
   src="http://[Service Address]/CedarScripts.js" />
2: <script type="text/javascript">
3: $(document).ready(function() {
4: Initialize('[ServiceAddress]'); //Setup the API
   //Call the API to adapt the UI and
   //pass the logged-in user id as a parameter)
5: LoadAdaptedUI(getUserID()); }); </script>
```

The "LoadAdaptedUI" method calls the server-side implementation of RBUIS, which will return an XML representation of the adapted UI. The following XML document represents an adapted "Login" UI example with a larger font-size and a different layouting of the widgets.

(e) XML Representing the Adapted UI (Excerpt)

```
<CUI Name="frmLogin">
  <GraphicalCUIElement Name="lblUserName" Text="User Name"
  Visible="True" Type="Label" Top="0" Left="0" FontSize="18"/>
  <GraphicalCUIElement Name="txtUserName" Visible="True"
  Type="TextBox" Top="50" Left="0" FontSize="18"/>
  <GraphicalCUIElement Name="lblPassword" Text="Password"
  Visible="True" Type="Label" Top="100" Left="0" FontSize="18"/>
  <GraphicalCUIElement Name="txtPassword" Visible="True"
  Type="Password" Top="150" Left="0"  FontSize="18"/>
</CUI>
```

(f) API Algorithm for Applying the Adapted UI (Excerpt)

```
 1: function ApplyAdaptedUI(UIXML){
     //Iterate around the UI widgets
 2:  $(UIXML).find("GraphicalCUIElement").each(
      function () {
        //Get each widget's properties
 3:     var elementName=$(this).attr('Name');
 4:     var isVisible = $(this).attr('Visible');
 5:     var fontSize = $(this).attr('FontSize');
 6:     var top = $(this).attr('TopPosition');
 7:     var left = $(this).attr('LeftPosition');
 8:     var element = GetElement(elementName);
 9:     if (typeof (element)!= 'undefined') {
10:       if(isVisible == 'false') {
11:         element.style.visibility = 'collapse'; }
12:         element.style.fontSize = fontSize;
13:         element.style.top = top;
14:         element.style.left = left;
16:      } }); }
```

The algorithm above adapts the running HTML UI by changing its properties based on the adapted XML. In this example, the adapted UI would look as follows:

User Name
Password

Fig. 11. Method for Integrating RBUIS into Legacy Software Systems Demonstrated with a Simple Example and Excerpts of Its Algorithm

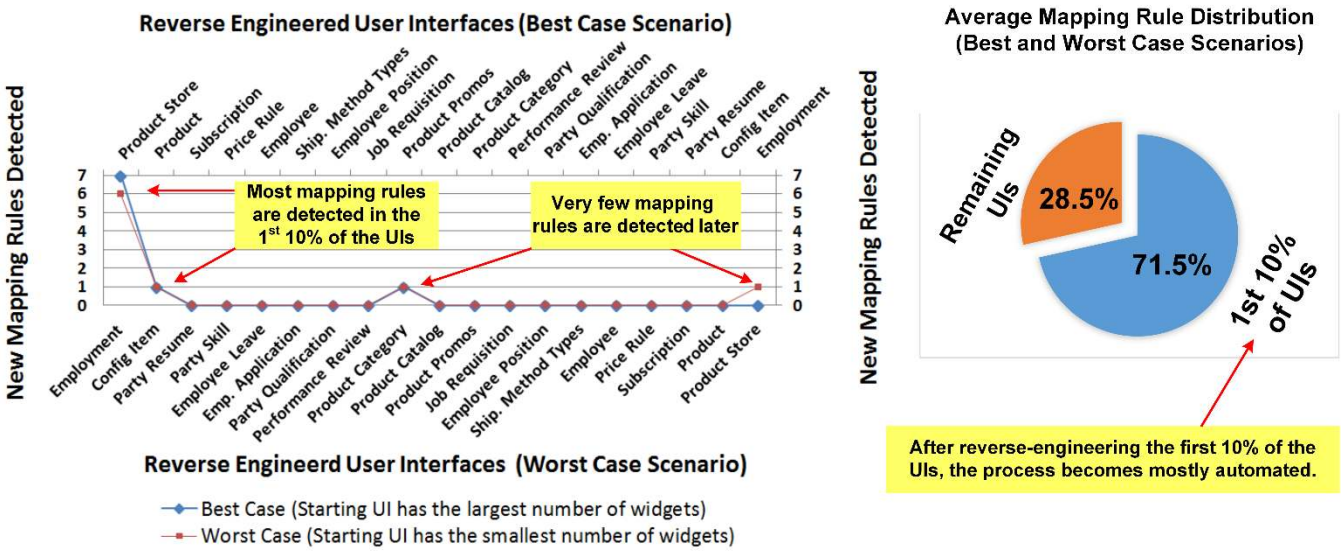


Fig. 12. Results of Applying the Approximate Mapping Rules Detection Saturation Point Metric to UIs from OFBiz

6.2 Evaluating the Integration Method

After establishing the technique for integrating RBUIS into legacy software systems, we established and applied technical metrics for evaluating our integration method in its different stages including: reverse-engineering, integration process, and runtime performance. Scenarios from OFBiz were used as examples for applying the metrics. Since this work only aims at adapting UIs, the integration method does not change an application's functional core (business logic), but extends the application with functionality that allows its UIs to become adaptive.

6.2.1 Degree of Automation in Reverse-Engineering of UIs

The process we devised for reverse-engineering code-based (e.g., HTML) UIs to a model-driven representation is automated. However, before this process can run automatically, some mapping rules have to be defined to indicate how certain elements on one level of abstraction map to their respective counterparts on another level. For example, an HTML *div* can map to a *label* at the CUI level, and the *label* can map to an *output* at the AUI level.

We defined a metric for estimating the number of consecutive UIs that have to be manually examined during the reverse-engineering phase before the process can become mostly automated. We should note that systems that have more diverse UIs could yield mapping rules, which are more uniformly distributed. We called this metric: **approximate mapping rule detection saturation point** (*SP*). The saturation point is the sequence number of a UI in the sequence of reverse-engineered UIs, after which the number of newly detected mapping rules stabilizes. We consider the metric's result to be positive if the Pareto principle (70-30 rule) holds. In practice, the applicability of the 70-30 rule indicates that 70% of the mapping rules are detected in the first 30% of the UIs that require reverse-engineering. Therefore, some manual work will be needed to reverse-engineer the first 30% of the UIs, thereafter most of the process becomes automated. We should note that our choice of the 70-30 rule does not indicate that this ratio is going to always be the same for all cases. Other rules can also apply, e.g. 80-20, 90-10, etc. As long as the majority of the mapping rules can be detected early on in a small percentage of the UIs, we can consider that the principle holds. A higher number of mapping rules detected early on in a smaller number of UIs, helps in automating a larger part of the process.

The following equation checks whether the Pareto principle (70-30 rule) holds:

$$P = \frac{|\{R\}|}{|\{MR\}|} : SP(\{MR\}) \leq 0.3 \qquad (1)$$

- $\{R\}$ is the set of rules detected in the UIs before SP
- $\{MR\}$ is the set of all the detected mapping rules

The saturation point SP is defined as follows:

$$SP(\{MR\}) = \frac{UI_a}{T} \mid \forall\, UI_{b \mid b>a} : C_b = C_{b+1} \pm \varepsilon \qquad (2)$$

- UI is a user interface being reverse engineered
- C is the number of new mapping rules detected in UI
- b of C is the sequence number of the next UI in the sequence of UIs that should be reverse-engineered
- T is the total number of UIs to be reverse-engineered

We applied the *SP* metric on a sample of the 19 main input UIs from the *Catalog* and *Human-Resources* modules of OFBiz. We deduced the rules necessary for reverse-engineering these UIs to a model-driven representation. We were able to deduce two types of mapping rules. The most common type of rule maps HTML elements to CUI elements, which are in turn mapped to AUI elements then tasks in the task model. The second type of rule is related to defining logical groups containing widget pairs composed of a label and an input widget, and reflecting these groups in the AUI and task models. We should note that these types of mapping rules are specific to OFBiz, which we selected as an example for this study. Other systems could have similar types of rules and different ones. The distribution of these rules across the sequence of reverse-engineered UIs is illustrated in Fig. 12 (left).

Table 2
User Interface Adaptations Used as Examples

	Adaptation
A1	Reduce features (e.g., hide or disable widgets)
A2	Switch widget type (e.g., combo-boxes to radio-buttons)
A3	Change layout grouping (e.g., group-boxes to tab-pages)
A4	Change font-size (e.g., large fonts for visually impaired)

* A1 is a feature-set min. A2, A3, and A4 are layout optimizations

Table 3
Change Impact of Applying an Adaptation that Switches Combo-boxes with Radio-buttons, List-boxes, and Lookups to a Sample of 19 UIs from OFBiz

		Change-Impact	
Approach	v (Eq. 4)	Mean	Total
Widget Toolkits	1	6.94	132
Generative Design-Time Model-Driven	3	106.73	2028
AOM & Design-Time Manual Adaptation (e.g., [33])	3	106.73	2028
Interpreted Runtime Model-Driven (*our adopted approach*)	0	0	0

* From Equation 4, $n = 1$ since we are only adapting combo-boxes. Each combo-box is represented by a single HTML tag hence and $l = 1$.

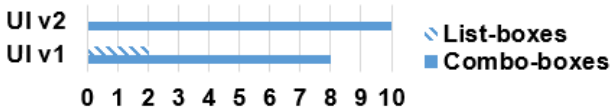
Most of the mapping rules were detected in the first 2 UIs (10% of the total 19 UIs). Hence, SP=2/19=0.10 indicating that after the second UI, the mapping rules become minimal as shown in Fig. 12 (left). Using SP=0.1, P is: 7/9=0.77 (77%) in the best case scenario and 6/9=0.66 (66%) in the worst case one. The average mapping rule distribution is 71.5% in the first 10% of the UIs (Fig. 12 - right). This indicates a high similarity in the UIs of OFBiz and a possibility of obtaining similar results for other enterprise applications using the same UI paradigm.

The completeness at the saturation point can be expressed using the following equation by Virzi [79]: $1 - (1 - p)^n$, where p is the probability of detecting a rule and n is the number of UIs (sample size). The value for p is 7.5/9, the average number of rules between the best and worst case scenarios at the saturation point (UI #2), and the value for n is 2, the number of the saturation point UI. The value would then be: $1 - (1 - 7.5 / 9)^2 = 0.9722$.

6.2.2 Change Impact of Integrating Different UI Adaptation Approaches in Legacy Software Applications

To measure the level of change that adaptive UI capability integration will have on legacy software systems, we defined the *lines-of-code* and *change-impact* metrics.

The **lines-of-code** (LOC) metric measures the code, which should be added locally in each UI or globally in the software application for making a certain UI adaptation operational. A line-of-code that can be added once per application, e.g. in a global class or web-page, and still achieve the desired integration functionality is considered global. In terms of integration, global lines-of-code are more desirable because they have a much lower change impact than local lines-of-code, which must be repeated in every UI. The API code is excluded from this metric since it is reusable with any software application.



- Two versions of the same UI have the same 10 fields that are data selection widgets (e.g., marital status)
- In v1 the 8 fields are combo-boxes and 2 are list-boxes
- In v2 the 10 fields are combo-boxes

Fig. 13. Backward Compatibility Example

This metric is only applicable for runtime approaches like ours since it represents the code that empowers an application with runtime adaptive UI capabilities. The lines-of-code metric is defined as follows:

$$LLOC(A, UI) \in \mathrm{N},\ GLOC(A, SA) \in \mathrm{N} \qquad (3)$$

- *LLOC* is the number of local lines-of-code added in a UI
- *GLOC* is the number lines-of-code added globally for the whole software application *SA*
- *A* is the adaptation to be applied to a user interface *UI*

The LOC metric is simple enough to use but cannot be applied to approaches that adapt the UI at design-time. Since we aim to compare our choice of an interpreted runtime model-driven approach to other approaches in the literature, we established the **change-impact** (CI) metric, which can be applied to different approaches by calculating UI change in terms of widgets as follows:

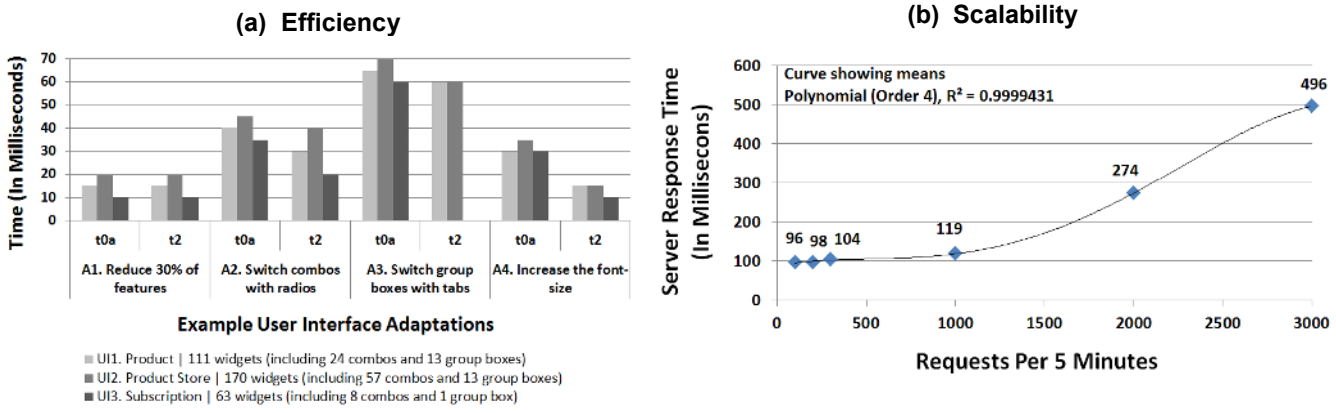
$$CI(A, UI) = \sum_{k=1}^{n} l_k \times |\Delta\{W\}_k| \times v \qquad (4)$$

- $\boldsymbol{A}$ is the adaptation being applied to a user interface $\boldsymbol{UI}$
- k is a type of widget (e.g., text box, combo box, etc.)
- n is the number of widget types in the *UI*
- l_k is the number of lines required for representing each widget type (e.g., number of HTML tags)
- $|\Delta\{W\}_k|$ is the number of widgets of a certain type that have been changed by the adaptation
- v is the number of generated UI versions and is > 1 for approaches that cannot adapt the same UI copy

A higher change-impact indicates that more time and effort could be needed to integrate a UI adaptation technique such as RBUIS in a legacy software application.

We applied the **LOC** metric to OFBiz UIs using the adaptations listed in Table 2 as a set of examples {AE}. The lines-of-code needed to make these adaptations work in OFBiz using our method are: $\forall$ x | x $\in$ {AE}, *GLOC* (x, OFBiz) = 5 lines-of-code (shown in Fig. 11 - d), and *LLOC* (x, AnyUI) = 0 since our approach does not modify the local UI code.

We should note that our adaptation technique is not only limited to the examples shown in Table 2. Furthermore, our intention is not to establish a taxonomy of adaptations, because RBUIS and Cedar Studio are general purpose and can support a wide variety of adaptations depending on the application being adapted and the context-of-use. We only selected a few adaptations for the purpose of performing this technical evaluation. Other example adaptations that can be supported include, but are not limited to: multi-document interface (window, page, or tab), multi-record visualization (grid, cards, or



- t_1 is 15ms over a 1Gbps corporate network, based on the average XML file size for the selected UIs is 20kb
- t_{0b} was calculated to be 30ms
- t_{0a} and t_2 are shown in the graph for each UI
- The fitting curve above shows the mean response times and is polynomial of the 4th order with $R^2 = 0.9999431$
- < 500 ms response time at 3000 requests per 5 minutes

Fig. 14. RBUIS Runtime Efficiency and Scalability (Load-Testing) Test Results

detailed form), accessibility of functions (high, medium, or low), information density (high, medium, or low), text versus graphics (high, medium, or low), etc.

We used the **CI** metric to compare the different types of UI adaptation approaches, which we encountered in the literature. We applied adaptation A2 (Table 2) to the 19 main input UIs of the *Catalog* and *Human Resources* modules of OFBiz. As an example, we assumed that adaptation A2 should switch combo-boxes in these UIs with one of the following alternatives: radio-buttons, list-boxes, and lookups. We calculated CI for the different possible adaptation approaches, based on our application of adaptation A2 to the selected samples of UIs. The results are presented in Table 3.

We can notice that v (Equation 4) is equal to 3 for the generative design-time model-driven and AOM approaches, because these approaches generate a different code version for each adapted UI. In this case, these approaches would generate a different version for each of the three combo-box alternatives. The high value for v for these two approaches resulted in a high *CI* (Mean = 106.73). The CI was less for widget toolkits (Mean = 6.94), because this approach only replaces the combo-boxes once with a generic adaptive widget from a custom toolkit. This widget would in principle be capable of adapting itself to any of the three alternatives at runtime. The integration time of the interpreted runtime model-driven approach, which we adopted, has $v = 0$ resulting in $CI = 0$. The yielded result is due to the fact that this approach does not cause any changes to the UIs at design-time but merely adapts them at runtime by executing adaptive behavior (e.g., workflows in RBUIS). Therefore, we can say that the UI adaptation approach we adopted allows developers to maintain their standard work procedure on the legacy application without major disruptions, while empowering these applications with RBUIS's adaptive UI capabilities.

6.2.3 Backward Compatibility of Different UI Adaptation Approaches with Legacy Software Applications

An adaptation can be considered backward compatible if it works with previous UI versions *successfully* and *without reintegration effort*.

We defined the **backward compatibility** (BC) metric as a ratio that determines the success of an adaptation:

$$BC(A) = \frac{|\{AW(UI_{vn})\} \cap \{AW(UI_{vn-k})\}|}{|\{AW(UI_{vn})\} \cap \{W(UI_{vn-k})\}|} \quad (5)$$

- UI_{vn} is a UI from the software application version into which the adaptation A was integrated for the first time
- UI_{vn-k} is a UI from one of the previous application versions
- $\{W\}$ is the set of widgets that a UI contains
- $\{AW\}$ is the set of widgets affected by an adaptation A

Consider the example presented in Fig. 13. If a UI adaptation was defined in UI-v2 for switching the data selection widgets with radio-buttons, it might ignore list boxes. In this example, $BC = 8/10 = 0.8$; hence there is an 80% success rate when considering UI-v1. With approaches that adapt the UI at design-time, two adapted UIs have to be generated and integrated into each respective UI version in order for the adaptation to fully work. Widget toolkit approaches can provide backward compatibility as long as a new version of the toolkit can be loaded at runtime. On the other hand, with dynamic approaches such as ours, only the adaptive behavior, e.g. workflows in RBUIS, have to be adjusted to take into consideration list-boxes as well as combo-boxes to achieve a full backward compatibility.

6.2.4 RBUIS's Runtime Efficiency and Scalability

We benefited from the highly dynamic nature of RBUIS in terms of providing a low change impact and a good backward compatibility as explained in Sections 6.2.2 and 6.2.3 respectively. However, we had to test whether the dynamism might degrade RBUIS's runtime *efficiency* and *scalability*, especially since end-users expect UIs to load in real-time.

We defined our **efficiency** metric as a function of an adaptation A and a user interface UI:

$$E(A, UI) = t_{0a} + t_{0b} + t_1 + t_2 \quad (6)$$

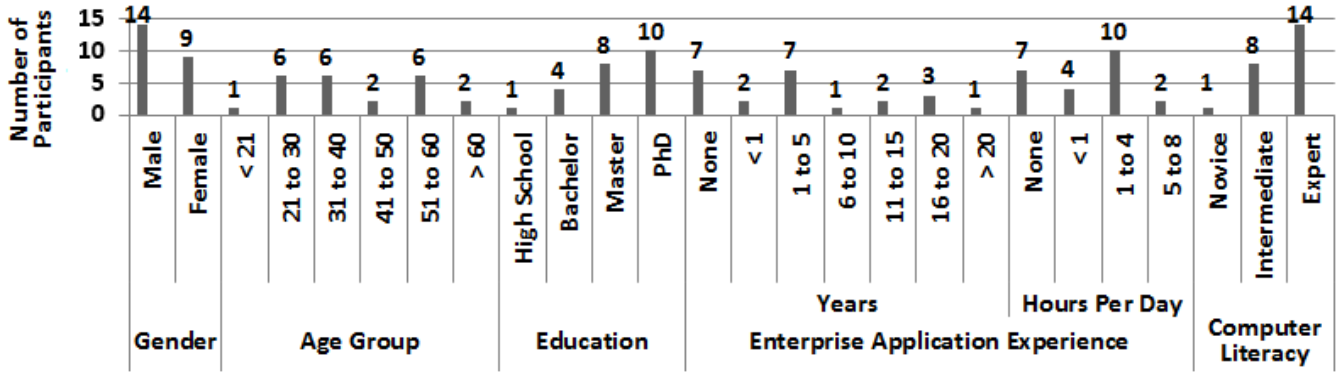


Fig. 15. Participant Demographic Information

- t_{0a} is the server-side time required to apply *A*
- t_{0b} is the common server-side time for any number of adaptations (e.g., time for loading common data)
- t_1 is the time it takes to transmit the adapted UI as XML to the client-side
- t_2 is the time it takes the API to adapt a running *UI* using the XML returned from the sever-side after applying *A*

If the caching components of the Cedar Architecture are implemented, the metric becomes as follows:

- With Client-Side Caching: $E(A, UI) = t_2$
- With Server-Side Caching: $E(A, UI) = t_1 + t_2$

We measured RBUIS's efficiency using this metric after executing the example adaptations listed in Table 2 on three UIs from the *Catalog* module of OFBiz. These UIs were selected, because they contain the largest number of widgets. Hence, if RBUIS can adapt them efficiently, it is bound to be at least as efficient with smaller UIs. The efficiency test was carried out on one machine with an Intel Core 2 Duo 2.93 GHz CPU and 4 GB of RAM running Windows 7 (32 bit). The Firefox web-browser was used to run OFBiz during the test.

The results of the efficiency test are illustrated in Fig. 14 – a. Based on this data, we determined the average efficiency for each adaptation to be as follows: $E(A1)$=75ms, $E(A2)$=115ms, $E(A3)$=150ms, and $E(A4)$=90ms. The general average is (75+115+150+90)/4=107.5ms. By subtracting the fixed values t_{0b} (30ms) and t_1 (15ms), this average is reduced to 62.5ms. Based on this number, we can say that RBUIS can apply around 15 different adaptations on the same UI, transmit the adapted UI's XML representation to the client-side, and adapt the running UI using the XML in less than 1 second (62.5×15+30+15=982.5ms).

The complexity analysis presented in Section 4 showed that RBUIS's algorithms are theoretically scalable. In this section, we demonstrate RBUIS's **scalability** by load-testing its web-service. We applied the four adaptation operations shown in Table 2 to the Product Store UI, which encompasses 170 widgets and is the largest among the three UIs we used for the efficiency test. The load-testing results are illustrated in Fig. 14 – b and are calculated as: $t_{0a} + t_{0b}$ based on Equation 6.

The web-service was hosted on an Amazon cloud machine running Windows Server 2012 (64-bit edition) and the IIS 7 web-server. The machine has a single Intel Xeon CPU with 2 cores (2.40 GHz, 2.15 GHz) and 3.75 GB of RAM. This configuration is considered moderate in comparison to that of large-scale enterprise servers with multiple CPUs and much more RAM. We developed our own application to perform the load-testing by calling the server. We ran three different instances of this application on three client machines.

7 Human-Centered (Usability) Evaluation

It is important to empirically assess UIs that are adapted to a particular context-of-use. For example, multi-target UIs that are generated using a model-driven approach can be evaluated by collecting both quantitative and qualitative data [80], [81]. We conducted a study to determine if RBUIS can significantly improve usability when applied to real-life enterprise application UIs.

Adaptive behavior can have benefits in terms of improving usability, but there could also be costs associated with it [4]. For example, if the adaptation is not accurate, the end-users might not trust it enough to use it and benefit from it [5]. One could expect an improvement, especially when simplifying a UI's feature-set, since the end-users are presented with fewer fields. Yet, the main question in this study is on the significance of this improvement. Since implementing adaptive UIs can add some overhead cost in comparison to using one generic off-the-shelf UI, the significance of the improvement in usability is important. A marginal improvement might not justify the investment in adaptive UIs.

In total, the study involved 23 participants × 8 UIs (4 initial and 4 simplified) × 3 criteria (efficiency, effectiveness, and perceived usability) = 552 samples. Eye-tracking was also performed with the same participants using 4 UIs (2 initial and 2 simplified): 23 × 4 × 2 criteria (fixation duration and fixation count) = 184 samples.

7.1 Study Design

This study has two parts, one on feature-set minimization and another on layout optimization.

In the feature-set minimization part, the participants were presented with two pairs of UIs on a desktop computer connected to a Tobii[4] eye-tracker. The UIs were selected from the SAP ERP system. The first pair included an initial and a simplified version of the Material UI. The second pair represented the Vendor maintenance UI. The initial version of these UIs contains several tab-pages and a lot of fields, and can be simplified for roles that require more basic functionality. An existing docu-

[4] Tobii eye-tracker: www.tobii.com

Table 4
Results of Wilcoxon Signed Ranks Test for
Perceived Usability and Efficiency

UI	Perc. Usability (SUS Score)	Efficiency (Time)
Material	Z = -4.200, p = 0.000027	Z = -4.197, p = 0.000027
Vendor	Z = -4.199, p = 0.000027	Z = -4.198, p = 0.000027
Sales Trx.	Z = -4.109, p = 0.000040	Z = -4.167, p = 0.000031
Contacts	Z = -2.617, p = 0.008877	*Not Measured (very short)*

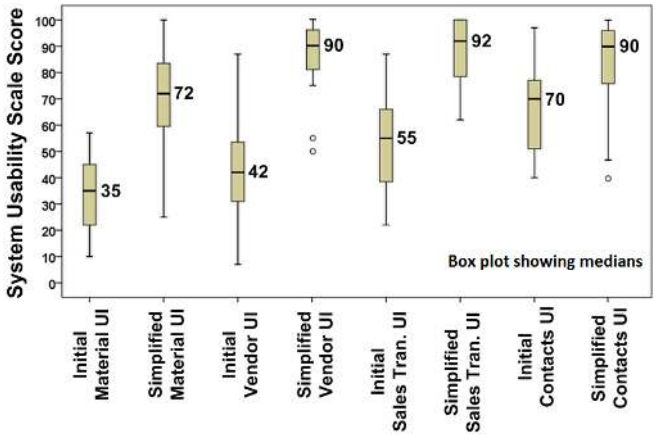


Fig. 16. End-User Perceived Usability Results (SUS Scores)

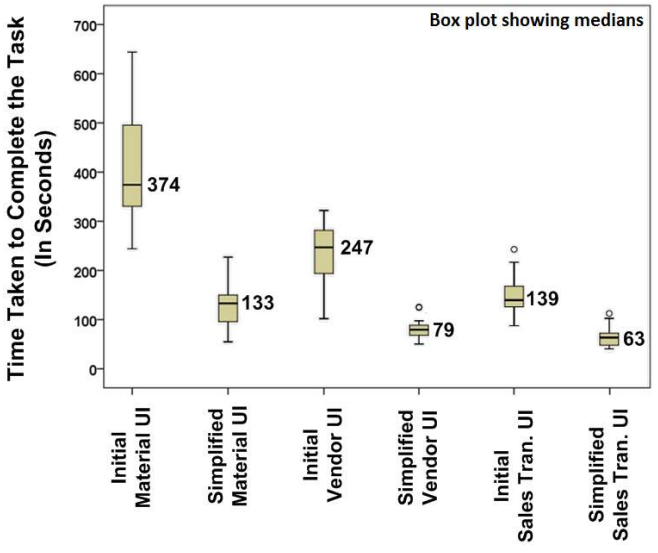


Fig. 17. End-User Efficiency Results (Time Taken to Complete Tasks)

ment provides information and examples on the variation in SAP's end-user needs, and helped us in determining what features to remove from the UIs [9]. The fields unrequired by a role were hidden in the simplified UI version, and the widgets were regrouped accordingly.

The initial UIs we used in the study had six tab-pages. As provided by SAP, these UIs originally have a larger number of tab-pages. However, we only used six because we wanted to keep the time required to complete the study reasonable for the participants. Nevertheless, a UI with six tab-pages offers enough complexity for evaluating the significance of usability improvement provided by feature-set minimization. Hence, if the simplified UI provides a significant improvement in comparison to an initial UI with six tab-pages, it is likely to yield similar or even better results when compared to an initial UI with more tab-pages.

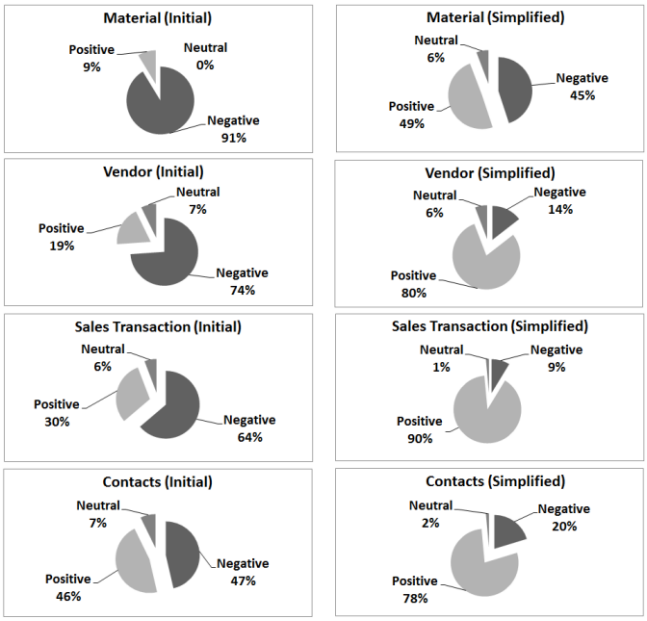


Fig. 18. Aggregated Product Reaction Card Results

In the layout optimization part, the participants were also given two pairs of UIs. The first pair was presented to them on an iPad tablet and consisted of an initial and a simplified version of the Sales Transaction UI, which was selected from Microsoft's Dynamics ERP system. Its initial version has a desktop-style input grid and a lookup list for selecting items. Many enterprise systems maintain this UI style on tablets. The simplified UI had a point-of-sale style. It offered a panel containing buttons for selecting items and a grid with up/down arrows for changing an item's quantity. The second pair was presented on an HTC Desire mobile phone and consisted of an initial and a simplified version of a Contacts (business partners) UI from the SAP ERP system.

Several measurements were made to compare the usability of both the initial and simplified versions of the UIs presented in this study. For both parts of the study, the participants were asked to answer the System Usability Scale (SUS) [82] questions to determine their perceived usability. In addition to SUS, the participants were asked to select three terms from the Microsoft Product Reaction Cards [83] to describe the UI. The SUS questionnaire was adopted because it has been validated and widely used in the literature. Additionally, SUS can be used to quantitatively measure the end-users' perceived usability, allowing a statistical evaluation of the results' significance. We used the Microsoft Product Reaction Cards, because this technique allows end-users to qualitatively express their opinion about a UI by selecting descriptive terms, and is hence complementary to SUS. The time taken to complete each task was also recorded in both parts to measure the end-users' efficiency with a UI.

In the feature-set minimization part of the study, eye-tracking was also used to determine how lost the partic-

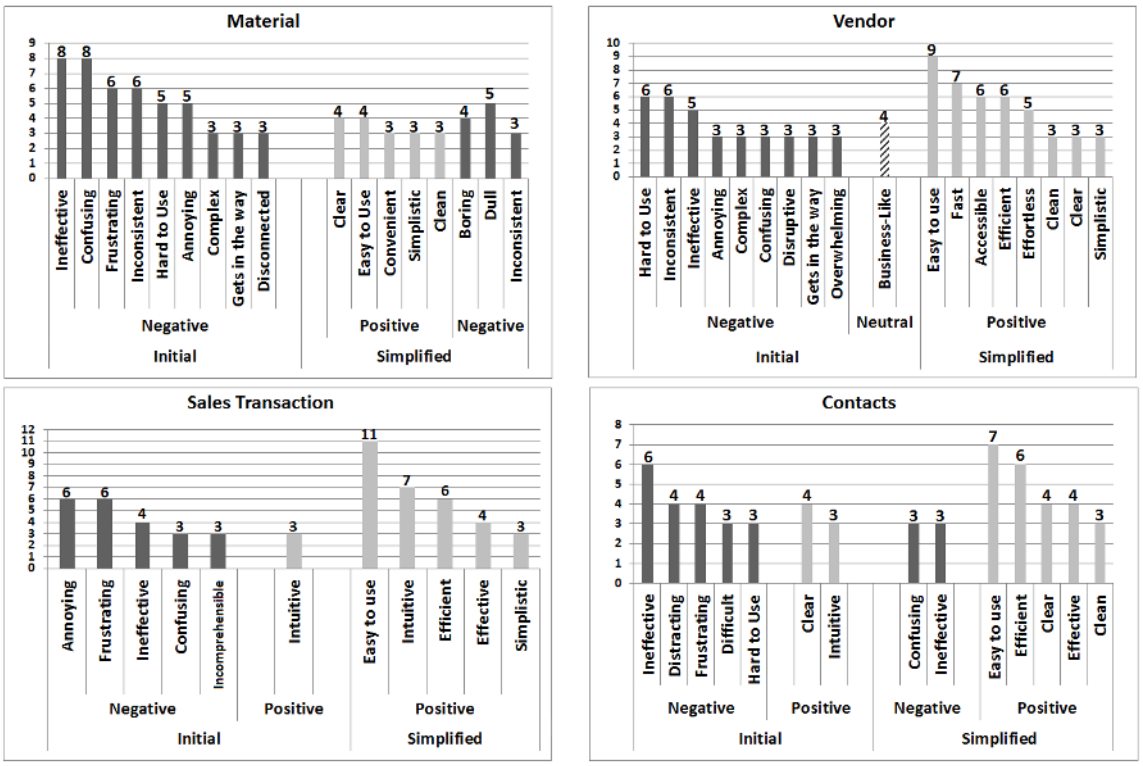


Fig. 19. Product Reaction Cards Selected More than Two Times by the Participants

ipants were in the initial UI versus the simplified one.

The sessions were video-recorded, and the participants were asked to express their thoughts out loud to give us insights on their experience. The video recordings, especially in the feature-set minimization part, helped us in identifying if a participant missed a field because of the UI's complexity or due to a simple human error.

7.2 Participant Demographics and Given Tasks

We recruited 23 participants, who volunteered to take part in this study without receiving any financial compensation. The study took an average of 45 minutes. The participants were recruited at random and had diverse demographics in terms of: gender, age group, education, computer literacy, and enterprise application expertise. Details about the participants' demographics are presented in Fig. 15. The study passed ethics approval and was advertised by email to Open University staff and students. The recruited participants included both academic and non-academic members of staff, PhD students, and one visiting high school student. They attended to the university's eye-tracking lab to participate in the study.

In the feature-set minimization part, the participants were asked to fill a new record in each of the initial and simplified Material and Vendor UIs.

In the layout optimization part, the participants were asked to perform a simple task with each UI. For the first pair of UIs, they were asked to select a customer, add four items, and change the quantities for two of the added items. As for the second pair, the participants were asked to call a contact assuming that the UI is being used while running down the street. They were presented with the ability to shake the phone, hence prompting it to change from one UI version to another.

Since some users have a tendency to like the first UI that they see, we presented half of the participants with the initial UI first, and the other half with the adapted one first. This technique, referred to as counterbalancing, helps in avoiding potential bias in experiments with same participant (within subjects) design. "*Counterbalancing neutralizes possible unfair effects of learning from the first task, known as the order effect*" [84]. Furthermore, once a participant saw the UI, performed the task, and answered the SUS questionnaire, he or she was not allowed to go back and change their answers after seeing the second UI version. As shown by the results presented in the coming sections, in all cases, when either the adapted UI or non-adapted one was presented first, the participants gave the adapted UI a better SUS score. This result is confirmed by the time it took the participants to complete the tasks, which does not depend on their perception, but is automatically measured by the system.

7.3 Perceived Usability and Efficiency Results

We used a Wilcoxon signed-ranks test to check if there are statistically significant differences between the initial UIs and the simplified ones. Upon observing the Quantile-Quantile plots, we found that the data we collected did not have a normal distribution. Therefore, we used the Wilcoxon signed-rank test, nonparametric equivalent to the dependent t-test, because it does not assume normality in the data. Additionally, this test can be applied to continuous variables, which is our case with the task completion times and SUS scores. Since we are only comparing two UIs, the Wilcoxon signed-rank test works accurately with a sample of $n \geq 20$, versus the need of $n \geq 30$

Table 5
Improvement in End-User Perc. Usability after UI Simplification

	Mean SUS Score		
UI	**Initial**	**Simplified**	**Improvement**
Material	34.09 (se=2.785)	68.78 (se=4.273)	**↑ 101.76% (2.01×)**
Vendor	41.65 (se=3.942)	86.13 (se=2.788)	**↑ 106.79% (2.06×)**
Sales Trx.	54.09 (se=4.488)	88.48 (se=2.830)	**↑ 63.58% (1. 63×)**
Contacts	66.70 (se=3.703)	83.74 (se=3.727)	**↑ 25.55% (1. 25×)**

Table 6
Improvement in End-User Efficiency after UI Simplification

	Mean Task Completion Time (In Seconds)		
UI	**Initial**	**Simplified**	**Improvement**
Material	406.39 (se=23.005)	129.96 (se=9.010)	**↓ 68.02% (3.12×)**
Vendor	236.30 (se=12.043)	84.57 (se=6.943)	**↓ 64.21% (2.79×)**
Sales Trx.	148.87 (se=8.035)	63.83 (se=4.331)	**↓ 57.12% (2.33×)**

Table 7
Improvement in End-User Effectiveness after UI Simplification

	Mean Missing Required Fields per Participant		
UI	**Initial**	**Simplified**	**Improvement**
Material	1.36 fields (6.33 %) (se=0.381)	0.14 fields (0.93 %) (se=0.100)	**↓ 89.92 % (9.92×)**
Vendor	0.95 fields (7.91 %) (se=0.259)	0.27 fields (2.25 %) (se=0.117)	**↓ 71.57 % (3.51×)**

in ANOVA when comparing more than two UIs.

The results presented in Table 4 show that simplifying the UI based on roles elicited a statistically significant improvement in both perceived usability based on the SUS score and efficiency based on the task completion time. The asymptotic significance (2-tailed) is the p value for the test, and the Wilcoxon signed-ranks test is reported using the Z statistic. The p value is < 0.01 for all cases, indicating that the simplified UI versions show a very strong improvement over the initial ones in terms of perceived usability and efficiency. We should note that the time taken by the participants to complete the task was not measured for the Contacts UI because the task is trivial and only takes a few seconds.

The results of the SUS scores are illustrated by the box-plots in Fig. 16. We can observe three outliers, but these are not extreme cases. The results of the time taken to complete the tasks are illustrated by the box-plots in Fig. 17. We can observe three outliers, but these are very close to the boundary of the box-plot.

The reaction cards selected by the participants to describe each UI confirm the improvement of the end-users' perceived usability achieved by the simplified UIs over the initial ones. The participants were asked to select three reaction cards to describe each UI. The pie-charts illustrated in Fig. 18 show that they selected a majority of positive terms when describing the simplified UIs, whereas they described the initial UIs with a majority of negative terms.

In addition to the reaction cards, the participants also mostly expressed dissatisfaction with the initial UIs in their verbal and written comments. For example, one participant said: "*I do not want a job where I have to use this UI* (initial version), *whatever the job may be*". On the other hand, the simplified UIs mostly got positive comments. For example, although the task was exactly the same for both versions of the same UI, one participant described a simplified UI as follows: "*The interface is simple, and the task is easier and more familiar*". The terms that were selected more than two times for each UI by the participants are shown by the bar-charts in Fig. 19.

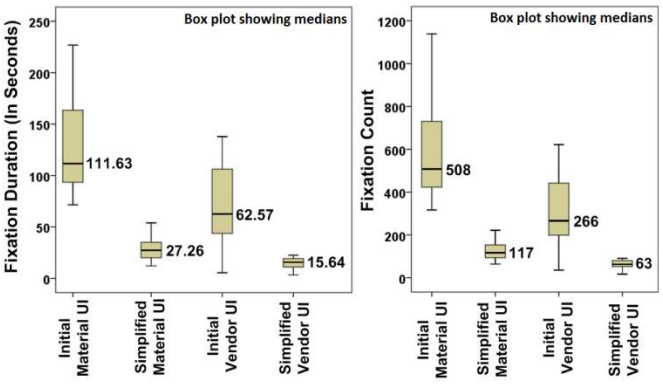


Fig. 20. Eye-Tracking Results of Fixation Duration and Fixation Count

Table 8
Improvement in Fixation Data after Simplification

	Mean Fixation Duration (In Seconds)		
UI	**Initial**	**Simplified**	**Improvement**
Material	137.68 (se=15.03)	29.77(se=2.748)	**↓ 78.37% (4.62×)**
Vendor	71.98 (se=8.693)	14.97 (se=1.365)	**↓ 79.20% (4.80×)**

	Mean Fixation Count		
UI	**Initial**	**Simplified**	**Improvement**
Material	599 (se=48.336)	128 (se=9.552)	**↓ 78.63% (4.67×)**
Vendor	312 (se=34.755)	63 (se=5.217)	**↓ 79.80% (4.95×)**

The mean SUS scores and task completion times are presented in Table 5 and Table 6 respectively. The improvement percentage, presented in the tables, for each of the UIs used in the study show the advantage of the simplified UI versions over the initial ones for all cases.

7.4 Effectiveness Results

The effectiveness is related to the "*accuracy and completeness with which specified users can achieve specified goals in particular environments*" [85]. To measure and compare the effectiveness between the initial and simplified UIs, we checked the number of fields that were left blank by the participants. We were able to determine the reason behind the effectiveness results because the sessions were video-recorded.

In the layout optimization part, the participants were able to complete the tasks successfully in most cases. The task given in the contacts UI (calling one of the contacts) was quite simple; hence all the participants were able to perform it. In the Sales Transaction UI, very few participants missed entering one of the items or increasing a quantity. As pointed out by the participants themselves, this mistake was not due to the UIs but to a simple human error when reading the instructions.

The case was not the same for the feature-set minimi-

Fig. 21. Heat Maps Showing Aggregation of the Participants' Gazing

Fig. 22. Gaze Plots of One Participant with Data Close to the Mean

zation part where the participants left more blank fields with the initial UI than with the simplified one. A Wilcoxon signed-ranks test showed that the simplified UIs elicited a strong statistically significant improvement ($p < 0.01$) in both the Material ($Z = -2.728$, $p = 0.0063$) and the Vendor ($Z = -2.655$, $p = 0.0079$) UIs. The average percentages of the missing fields per participant are presented in Table 7.

Very few fields were missed with the simplified UIs constituting an average of 1.59% per participant. This percentage indicates that there is on average 1 missing field for every 5 participants. As we observed in the video-recordings, the main reason for missing a field in the simplified UI is a simple human error when reading the instructions (e.g., skipping a field by mistake).

The percentage of missing fields was higher with the initial UI versions having an average of 7.12%. This percentage indicates that there is on average 1 missing field for each participant. By observing the video-recordings, we noticed that there were two main reasons behind the missing fields in the initial UI. The first reason is that in some cases the participants tried to fill a field whenever they spotted it, thereby causing them to forget some fields because of not working sequentially. The second reason was knowingly skipping a field after getting frustrated from searching thoroughly and not finding it. Some participants tried using the search feature available in the web-browser to find certain fields. Yet, as we observed, this was not helpful in all the cases since the participants still had to go through the tabs and apply the search on each one separately.

7.5 Eye Tracking Results

The eye-tracking that we conducted in the feature-set minimization part of the study on the Material and Vendor UIs, helped to determine and compare how lost the participants were when using each of the UI versions. We discarded the eye-tracking data for 4 of our 23 participants because their eye-glasses prevented the eye-tracking device from recording sufficient data. We used two metrics, namely the fixation duration and fixation count, from the eye-tracking data to determine how lost the participants were when using the different UI versions. The fixation duration is the measurement of how much time a participant spent focusing directly on certain points in the UI, while the fixation count is the measurement of the number of times that each participant directly focused on certain points.

The difference between the initial and simplified UIs, in terms of fixation duration and fixation count, is illustrated by the box-plots in Fig. 20. By observing the box-plots, we can notice the improvement between the initial and simplified UI versions for all cases. These numbers represent the eye-tracking data of the part of the screen that shows the data entry form. The task instructions that were also displayed on the screen were excluded by defining an area-of-interest around each UI using the software tool provided by the Tobii eye-tracker. The areas-of-interest allow us to get accurate data about how much gazing each participant did on the UI itself.

The mean values of the fixation duration and fixation count are presented in Table 8 alongside the improvement percentages, which show that the simplified UIs have a significant advantage over the initial ones.

The heat maps in Fig. 21 show the aggregated fixation data. The improvement provided by the simplified UI versions can be observed visually. We can notice that the highest amount of gazing is done on the left-hand-side of the input fields in the initial UI, where the labels

are presented. This indicates that the participants were carefully checking the labels because it was difficult for them to find the fields in which they were required to enter data. On the other hand, with the simplified versions, the overall gazing was much less intense. The gaze plots in Fig. 22 illustrate an example of one participant whose data came close to the means presented in Table 8. We can observe that even in an average case the significance of the improvement is visually noticeable.

8 Threats to Validity

In terms of the technical evaluation (Section 6), the data presented is based on applying our UI adaptation approach to scenarios from OFBiz. The figures we obtained by applying the saturation point (SP) metric (Section 6.2.1) give us an indication about the nature of enterprise application UIs, and potentially other systems that use WIMP-style UIs, without claiming generalizability to all software applications. When we compared our approach to others from the literature using the change-impact (CI) and backward compatibility (BC) metrics, we aimed at giving a general conceptual idea about the differences, while acknowledging that there could be some variations between the low-level adaptation techniques using the same approach. The load-testing curve, presented in Fig. 14 – b, is intended to show that our UI adaptation mechanism is scalable. Determining an accurate regression equation is not the purpose of this test and requires a larger sample of mean execution times.

Concerning the usability evaluation (Section 7), one might ask about the effect of learning over time on the results, which we obtained from the usability study. Would learning eventually make the end-users more effective, efficient, and satisfied with the initial UIs? We can say that as the end-users learn, their efficiency and effectiveness are likely to improve for both the initial and the simplified UIs. Learning however is unlikely to improve their perceived usability with the initial UIs. Subjecting the end-users to the complexity of the initial UIs could drive them to reject the software application, e.g. ERP system, in the early stages of the training, hence causing an implementation failure. An additional rationale for using the simplified UIs is that training the end-users on the initial UIs requires more time and money, whereas the simplified ones are likely to be learned more quickly.

9 Conclusions

This paper contributes an approach for engineering adaptive model-driven UIs. Our approach is not intended to replace any of the stakeholders involved in the process of designing and developing UIs. It is merely meant to help them in producing UIs that fit the context-of-use better, thereby providing end-users with an improved usability. We evaluated the state-of-the-art after classifying it under: architectures, techniques, and tools. The evaluation identified gaps, which we filled by presenting three novel technical contributions: the Cedar Architecture, the Role-Based UI Simplification (RBUIS) mechanism, and their supporting IDE, Cedar Studio.

We presented the **Cedar Architecture** in Section 3, as a reference for the stakeholders interested in developing adaptive model-driven UIs. This architecture is based on existing works including: the Three Layer Architecture [67] and the Cameleon Reference Framework (CRF) [66]. The Cedar Architecture has three server-side technology-independent layers including: *decision components*, *adaptation components*, and *adaptive behavior and UI models*. It also has a technology-specific *client components* layer. This layer's components are part of an API, which integrates in a software application's code and allows it to connect to the server-side layers in order to adapt its user interfaces. The Cedar Architecture adopts interpreted runtime models (Section 2.1) as a *modeling approach*, in order to support more advanced runtime adaptations.

In Section 4, we presented **Role-Based User Interface Simplification** (RBUIS), a mechanism for improving usability through adaptive behavior by providing end-users with a minimal feature-set and an optimal layout based on the context-of-use. RBUIS merges role-based access control (RBAC) with adaptive behavior for simplifying UIs. In RBUIS, roles are divided into groups representing the aspects (e.g., computer literacy, job title, etc.) based on which the UI will be simplified. RBUIS supports *feature-set minimization* by assigning roles to task models for providing end-users with a minimal feature-set based on the context-of-use. The assignment could be done by IT personnel, but there is also a potential for engaging end-users in the process. *Layout optimization* is supported by assigning roles to workflows that represent adaptive UI behavior visually and through code before being applied to CUI models. RBUIS fills a major gap in the existing state-of-the-art feature-set minimization techniques, which lack: a practical implementation, generality, and/or the ability to be applied at runtime. Furthermore, RBUIS fulfills several of the criteria presented in Section 2.1. It promotes *user-feedback* for refining the adaptation operations. Hence, end-users are allowed to reverse feature-set minimizations and layout optimizations, and to choose possible alternative layout optimizations. The *extensibility of aspects and factors* and *extensibility of adaptive behavior* is achieved through the use of role-based workflows and scripts and role-assignments to task models. *Visual/code-based representation* of adaptive behavior can be done using visual workflows and visual role-assignments, in addition to scripts and RBUIS rules.

A brief overview of **Cedar Studio** was presented in Section 5. This IDE supports the development of adaptive model-driven UIs based on the Cedar Architecture and using RBUIS. Cedar Studio provides stakeholders such as developers and IT personnel, with *visual-design and code-editing tools* for defining and managing artifacts such as UI models and adaptive behavior. Using these tools, the *adaptive behavior* and the *adaptation aspect and factors* can be extended as needed. More information about this tool can be found in a separate paper [16]. It can be also seen in operation through several demonstration videos [78]. Cedar Studio provides *expressive leverage* (Section 2.1) by supporting the use of reusable visual-components and scripts as part of the adaptive behavior workflows. It also achieves a balance between *threshold and ceiling* (Section 2.1). For example, it sup-

ports automated generation and synchronization between models (low threshold), alongside the possibility of conducting manual adjustments (high ceiling). Also, if developers understand the semantics of the meta-model, then they can use the visual-design tools to produce an advanced outcome (medium threshold/high ceiling). In the cases where coding could be used, a visual-design tool alternative was provided.

In Sections 6 and 7, the **contributions were evaluated from technical and human perspectives**. In the technical evaluation, RBUIS was integrated into an existing open-source enterprise application called OFBiz. Several metrics were defined and applied to measure technical characteristics related to: reverse-engineering, integration, and runtime execution. We showed that RBUIS could be integrated into existing legacy software applications without causing major changes to the way they function or incurring a high integration cost. We also showed that it can run efficiently in real-time and that it is scalable. The technical evaluation showed that our approach fulfills two criteria from Section 2.1: *integrating in legacy systems* and *scalability*.

Concerning the human perspective, we evaluated whether our UI adaptation approach can significantly improve the usability of complex UIs, by conducting a usability study with real-life UIs. This study showed that UIs with a minimized feature-set and an optimized layout elicited a very strong statistically significant improvement over their initial versions in terms of end-user efficiency, effectiveness, and perceived usability. Eye-tracking was also conducted, and it showed that minimizing the feature-set of complex UIs significantly decreases the extent to which end-users are lost when searching for input fields.

By presenting our approach for adapting UIs to the context-of-use, we aim to support the design of interfaces that can provide better usability than a single static off-the-shelf UI design. As a future outlook, we think that collaboration between academics researching adaptive model-driven UI development systems, and industrial partners who are really interested in adopting these systems, could help in improving the research outcome in this area. Furthermore, such collaboration could help in producing tools, which rival traditional commercial IDEs, and are more widely adopted in industry. The work presented in this paper can be considered as a strong starting point towards this objective.

ACKNOWLEDGMENT

This research was funded by the Computing and Communications Department at The Open University and ERC Advanced Grant 291652. We would like to thank Shanghai Banff Technology Ltd and all the participants of our study for their valuable feedback.

REFERENCES

[1] H. Topi, W. T. Lucas, and T. Babaian, "Identifying Usability Issues with an ERP Implementation.," in *Proceedings of the 7th International Conference on Enterprise Information Systems*, Miami, USA, 2005, pp. 128–133.

[2] A. Jameson, "Adaptive Interfaces and Agents," J. A. Jacko and A. Sears, Eds. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 2003, pp. 305–330.

[3] B. Rosman, S. Ramamoorthy, M. M. H. Mahmud, and P. Kohli, "On User Behaviour Adaptation Under Interface Change," in *Proceedings of the 19th International Conference on Intelligent User Interfaces*, New York, NY, USA, 2014, pp. 273–278.

[4] T. Lavie and J. Meyer, "Benefits and costs of adaptive user interfaces," *Int. J. Hum.-Comput. Stud.*, vol. 68, no. 8, pp. 508 – 524, 2010.

[5] K. Z. Gajos, M. Czerwinski, D. S. Tan, and D. S. Weld, "Exploring the Design Space for Adaptive Graphical User Interfaces," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, New York, NY, USA, 2006, pp. 201–208.

[6] R. Kniewel, C. Evers, L. Schmidt, and K. Geihs, "Designing Usable Adaptations," in *Socio-technical Design of Ubiquitous Computing Systems*, K. David, K. Geihs, J. M. Leimeister, A. Roßnagel, L. Schmidt, G. Stumme, and A. Wacker, Eds. Springer International Publishing, 2014, pp. 211–232.

[7] P. A. Akiki, A. K. Bandara, and Y. Yu, "Adaptive Model-Driven User Interface Development Systems," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 64:1–64:33, 2014.

[8] J. M. Carroll and C. Carrithers, "Training Wheels in a User Interface," *Commun. ACM*, vol. 27, no. 8, pp. 800–806, Aug. 1984.

[9] Synactive GmbH, "GuiXT - Simplify and Optimize the SAP ERP User Interface," 2010. [Online]. Available: http://bit.ly/SAPGuiXTSimplifyUI. [Accessed: 04-Sep-2012].

[10] S. Lepreux, J. Vanderdonckt, and B. Michotte, "Visual Design of User Interfaces by (De)composition," in *Interactive Systems. Design, Specification, and Verification*, vol. 4323, G. Doherty and A. Blandford, Eds. Springer Berlin Heidelberg, 2007, pp. 157–170.

[11] K. Z. Gajos, D. S. Weld, and J. O. Wobbrock, "Automatically Generating Personalized User Interfaces with Supple," *Artif. Intell.*, vol. 174, no. 12, pp. 910–950, Aug. 2010.

[12] S. Feuerstack, M. Blumendorf, and S. Albayrak, "Bridging the Gap between Model and Design of User Interfaces," in *GI Jahrestagung (2)*, Dresden, Germany, 2006, vol. P-94, pp. 131–137.

[13] M. Peissner, D. Häbe, D. Janssen, and T. Sellner, "MyUI: Generating Accessible User Interfaces from Multimodal Design Patterns," in *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, Copenhagen, Denmark, 2012, pp. 81–90.

[14] P. A. Akiki, A. K. Bandara, and Y. Yu, "Integrating Adaptive User Interface Capabilities in Enterprise Applications," in *Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, 2014, pp. 712–723.

[15] P. A. Akiki, A. K. Bandara, and Y. Yu, "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, 2013, pp. 3–12.

[16] P. A. Akiki, A. K. Bandara, and Y. Yu, "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, 2013, pp. 139–144.

[17] P. A. Akiki, A. K. Bandara, and Y. Yu, "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, 2013, pp. 121–126.

[18] P. A. Akiki, A. K. Bandara, and Y. Yu, "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, 2012, vol. Volume 3, pp. 72–77.

[19] A. Demeure, G. Calvary, and K. Coninx, "COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces," in *Proceedings of the 15th International Workshop on Interactive Systems Design Specification and Verification*, Kingston, Canada, 2008, pp. 225 – 237.

[20] D. R. Olsen,Jr., "Evaluating User Interface Systems Research," in *Proceedings of the 20th ACM SIGCHI Symposium on User Interface Software and Technology*, Newport, Rhode Island, USA, 2007, pp. 251–258.

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

[22] B. Myers, S. E. Hudson, and R. Pausch, "Past, Present, and Future of User Interface Software Tools," *ACM Trans. Comput.-Hum. Interact.*, vol. 7, no. 1, pp. 3–28, Mar. 2000.

[23] J. McGrenere, R. M. Baecker, and K. S. Booth, "An Evaluation of a Multiple Interface Design Solution for Bloated Software," in *Proceedings of the 20th SIGCHI Conference on Human Factors in Computing Systems*, Minneapolis, Minnesota, USA, 2002, pp. 164–170.

[24] L. Findlater and J. McGrenere, "Evaluating Reduced-Functionality Interfaces According to Feature Findability and Awareness," in *Proceedings of the 13th International Conference on Human-Computer Interaction*, 2007, pp. 592–605.

[25] C. Miller, H. Funk, P. Wu, R. Goldman, J. Meisner, and M. Chapman, "The Playbook™ Approach to Adaptive Automation," in *Proceedings of the 49th Human Factors and Ergonomics Society Annual Meeting*, Florida, U.S.A., 2005, vol. 49, pp. 15–19.

[26] V. López-Jaquero, F. Montero, and F. Real, "Designing User Interface Adaptation Rules with T:XML," in *Proceedings of the 14th International Conference on Intelligent User Interfaces*, Sanibel Island, Florida, USA, 2009, pp. 383–388.

[27] G. Lehmann, A. Rieger, M. Blumendorf, and S. Albayrak, "A 3-Layer Architecture for Smart Environment Models," in *Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications*, Mannheim, Germany, 2010, pp. 636 –641.

[28] O. Brdiczka, J. L. Crowley, and P. Reignier, "Learning Situation Models for Providing Context-Aware Services," in *Universal Access in HCI*, C. Stephanidis, Ed. Springer-Verlag, 2007, pp. 23–32.

[29] L. Balme, R. Demeure, N. Barralon, J. Coutaz, and G. Calvary, "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, 2004, pp. 291–302.

[30] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *Intell. Syst. Their Appl. IEEE*, vol. 14, no. 3, pp. 54–62, 1999.

[31] C. Duarte and L. Carriço, "A Conceptual Framework for Developing Adaptive Multimodal Applications," in *Proceedings of the 11th International Conference on Intelligent User Interfaces*, Sydney, Australia, 2006, pp. 132–139.

[32] A. Blouin and O. Beaudoux, "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, 2010, pp. 115–124.

[33] A. Blouin, B. Morin, O. Beaudoux, G. Nain, P. Albers, and J.-M. Jézéquel, "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, 2011, pp. 85–94.

[34] G. Rossi, D. Schwabe, and R. Guimarães, "Designing Personalized Web Applications," in *Proceedings of the 10th International Conference on World Wide Web*, Hong Kong, 2001, pp. 275–284.

[35] M. Pazzani and D. Billsus, "Content-Based Recommendation Systems," in *The Adaptive Web*, vol. 4321, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Springer Berlin Heidelberg, 2007, pp. 325–341.

[36] B. Shneiderman, "Promoting Universal Usability with Multi-Layer Interface Design," in *Proceedings of the Conference on Universal Usability*, Vancouver, Canada, 2003, pp. 1–8.

[37] A. Pleuss, G. Botterweck, and D. Dhungana, "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, 2010, pp. 69–76.

[38] G. Botterweck, "Multi Front-End Engineering," in *Model-Driven Development of Advanced User Interfaces*, vol. 340, H. Hussmann, G. Meixner, and D. Zuehlke, Eds. Springer, 2011, pp. 27–42.

[39] N. Bencomo, P. Sawyer, G. S. Blair, and P. Grace, "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, 2008, vol. 2, pp. 23–32.

[40] Microsoft, "Role based UI - Dynamics CRM 2011," 2011. [Online]. Available: http://bit.ly/DynamicsRoleBasedUI. [Accessed: 31-Aug-2012].

[41] M. Florins and J. Vanderdonckt, "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, 2004, pp. 140–147.

[42] G. Calvary, J. Coutaz, O. Dâassi, L. Balme, and A. Demeure, "Towards a New Generation of Widgets for Supporting Software Plasticity: The 'Comet,'" in *Engineering Human Computer Interaction and Interactive Systems*, vol. 3425, R. Bastide, P. Palanque, and J. Roth, Eds. Springer, 2005, pp. 306–324.

[43] T. Clerckx, K. Luyten, and K. Coninx, "DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development," in *Engineering Human Computer Interaction and Interactive Systems*, R. Bastide, P. A. Palanque, and J. Roth, Eds. Springer, 2005, pp. 77–95.

[44] K. Coninx, K. Luyten, C. Vandervelpen, J. Van den Bergh , and B. Creemers, "Dygimes: Dynamically Generating Interfaces for

Mobile Computing Devices and Embedded Systems," in *Proceedings of the 5th International Symposium on Human-Computer Interaction with Mobile Devices and Services*, Udine, Italy, 2003, pp. 256–270.

[45] K. Gajos and D. S. Weld, "Preference Elicitation for Interface Optimization," in *Proceedings of the 18th ACM Symposium on User Interface Software and Technology*, Seattle, USA, 2005, pp. 173–182.

[46] M. Blumendorf, S. Feuerstack, and S. Albayrak, "Multimodal Smart Home User Interfaces," in *Proceedings of the Workshop on Intelligent User Interfaces for Ambient Assisted Living*, Gran Canaria, Spain, 2008.

[47] M. Blumendorf, S. Feuerstack, and S. Albayrak, "Multimodal User Interaction in Smart Environments: Delivering Distributed User Interfaces," in *Constructing Ambient Intelligence*, M. Mühlhäuser, A. Ferscha, and E. Aitenbichler, Eds. Berlin: Springer-Verlag, 2007, pp. 113–120.

[48] M. Blumendorf, S. Feuerstack, and S. Albayrak, "Event-based Synchronization of Model-Based Multimodal User Interfaces," in *Proceedings of the 2nd International Workshop on Model Driven Development of Advanced User Interfaces*, Genova, Italy, 2006, pp. 1–5.

[49] V. Schwartze, S. Feuerstack, and S. Albayrak, "Behavior-Sensitive User Interfaces for Smart Environments," in *Proceedings of the 2nd International Conference on Digital Human Modeling*, San Diego, USA, 2009, pp. 305–314.

[50] H. Chu, H. Song, C. Wong, S. Kurakake, and M. Katagiri, "Roam, a seamless application framework," *J. Syst. Softw.*, vol. 69, no. 3, pp. 209 – 226, 2004.

[51] W. Viana and R. M. C. Andrade, "XMobile: A MB-UID environment for semi-automatic generation of adaptive applications for mobile devices," *J. Syst. Softw.*, vol. 81, no. 3, pp. 382 – 394, 2008.

[52] V. Genaro Motti, D. Raggett, S. Van Cauwelaert, and J. Vanderdonckt, "Simplifying the Development of Cross-platform Web User Interfaces by Collaborative Model-based Design," in *Proceedings of the 31st ACM International Conference on Design of Communication*, New York, NY, USA, 2013, pp. 55–64.

[53] G. Mori, F. Paternò, and C. Santoro, "CTTE: Support for Developing and Analyzing Task Models for Interactive System Design," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 797–813, 2002.

[54] F. Paterno', C. Santoro, and L. D. Spano, "MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments," *ACM Trans. Comput.-Hum. Interact.*, vol. 16, no. 4, pp. 19:1–19:30, Nov. 2009.

[55] Q. Limbourg and J. Vanderdonckt, "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, 2004, pp. 325–338.

[56] A. García Frey, E. Céret, S. Dupuy-Chessa, G. Calvary, and Y. Gabillon, "UsiComp: An Extensible Model-Driven Composer," in *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, Copenhagen, Denmark, 2012, pp. 263–268.

[57] A. García Frey, G. Calvary, and S. Dupuy-Chessa, "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, 2010, pp. 41–46.

[58] A. Coyette and J. Vanderdonckt, "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, 2005, pp. 12–16.

[59] F. Montero and V. López-Jaquero, "IdealXML: An Interaction Design Tool," in *Computer-Aided Design of User Interfaces V*, G. Calvary, C. Pribeanu, G. Santucci, and J. Vanderdonckt, Eds. Springer, 2007, pp. 245–252.

[60] B. Michotte and J. Vanderdonckt, "GrafiXML, a Multi-target User Interface Builder Based on UsiXML," in *Proceedings of the 4th International Conference on Autonomic and Autonomous Systems*, Cancun, Mexico, 2008, pp. 15 –22.

[61] S. Feuerstack, M. Blumendorf, V. Schwartze, and S. Albayrak, "Model-based Layout Generation," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, Napoli, Italy, 2008, pp. 217–224.

[62] J. Meskens, J. Vermeulen, K. Luyten, and K. Coninx, "Gummy for Multi-Platform User Interface Designs: Shape me, Multiply me, Fix me, Use me," in *Proceedings of the 8th Working Conference on Advanced Visual Interfaces*, Napoli, Italy, 2008, pp. 233–240.

[63] J. Lin and J. A. Landay, "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, 2008, pp. 1313–1322.

[64] A. I. Molina, W. J. Giraldo, J. Gallardo, M. A. Redondo, M. Ortega, and G. García, "CIAT-GUI: A MDE-compliant environment for developing Graphical User Interfaces of information systems," *Adv. Eng. Softw.*, vol. 52, pp. 10–29, 2012.

[65] J. Vanderdonckt, "Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges," *Romanian J. Hum. - Comput. Interact.*, vol. 1, no. 1, pp. 1–10, 2008.

[66] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A Unifying Reference Framework for Multi-Target User Interfaces," *Interact. Comput.*, vol. 15, no. 3, pp. 289–308, 2003.

[67] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," in *Proceedings of the Workshop on the Future of Software Engineering*, Minneapolis, USA, 2007, pp. 259–268.

[68] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, "UIML: An Appliance-Independent XML User Interface Language," *Comput. Netw.*, vol. 31, no. 11, pp. 1695–1708, May 1999.

[69] M. Blumendorf, G. Lehmann, and S. Albayrak, "Bridging Models and Systems at Runtime to Build Adaptive User Interfaces," in *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, Berlin, Germany, 2010, pp. 9–18.

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

[71] F. Paternò, C. Mancini, and S. Meniconi, "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models," in *Proceedings of the 6th International Conference on Human-Computer Interaction*, Sydney, Australia, 1997, vol. 96, pp. 362–369.

[72] F. Montero, V. López-Jaquero, J. Vanderdonckt, P. González, M. Lozano, and Q. Limbourg, "Solving the Mapping Problem in

User Interface Design by Seamless Integration in IdealXML," in *Interactive Systems. Design, Specification, and Verification*, vol. 3941, S. Gilroy and M. Harrison, Eds. Springer Berlin Heidelberg, 2006, pp. 161–172.

[73] A. Puerta and J. Eisenstein, "XIML: A Multiple User Interface Representation Framework for Industry," in *Multiple User Interfaces*, John Wiley & Sons, Ltd, 2005, pp. 119–148.

[74] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, Aug. 2001.

[75] P. A. Akiki, A. K. Bandara, and Y. Yu, "Cedar: Engineering Role-Based Adaptive User Interfaces for Enterprise Applications," Technical Report, 2012.

[76] J. Coutaz, "Meta-User Interfaces for Ambient Spaces," in *Task Models and Diagrams for Users Interface Design*, vol. 4385, K. Coninx, K. Luyten, and K. Schneider, Eds. Springer Berlin Heidelberg, 2007, pp. 1–15.

[77] D. Roscher, G. Lehmann, M. Blumendorf, and S. Albayrak, "Design and Implementation of Meta User Interfaces for Interaction in Smart Environments," presented at the Supportive User Interfaces (SUI 2011), Pisa, Italy, 2011.

[78] "Cedar Studio - Demo Video." [Online]. Available: http://adaptiveui.pierreakiki.com.

[79] R. A. Virzi, "Refining the Test Phase of Usability Evaluation: How Many Subjects Is Enough?," *Hum. Factors J. Hum. Factors Ergon. Soc.*, vol. 34, no. 4, pp. 457–468, 1992.

[80] N. Aquino, J. Vanderdonckt, N. Condori-Fernández, Ó. Dieste, and Ó. Pastor, "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, 2010, pp. 30:1–30:10.

[81] S. Abrahão, E. Iborra, and J. Vanderdonckt, "Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool," in *Maturing Usability*, E.-C. Law, E. Hvannberg, and G. Cockton, Eds. Springer London, 2008, pp. 3–32.

[82] J. Brooke, "SUS: A Quick and Dirty Usability Scale," in *Usability Evaluation in Industry*, vol. 189, P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, Eds. London, UK: Taylor and Francis, 1996.

[83] J. Benedek and T. Miner, "Measuring Desirability: New methods for Evaluating Desirability in a Usability Lab Setting," *Proc. Usability Prof. Assoc.*, vol. 2003, pp. 8–12, 2002.

[84] J. Preece, H. Sharp, and Y. Rogers, "Experimental Design," in *Interaction Design: Beyond Human - Computer Interaction*, 4th ed., John Wiley & Sons, p. 486.

[85] ISO 9241, "ISO 9241-12:1998 - Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) -- Part 12: Presentation of information," 2008. [Online]. Available: http://bit.ly/ISO9214. [Accessed: 14-May-2012].

Pierre A. Akiki received a BSc. and a MSc. in Computer Science from Notre Dame University – Louaize, Lebanon in 2004 and 2007 respectively. He also received a MSc. in International Business from Bordeaux Business School (now KEDGE), France and a M.B.A. from Notre Dame University – Louaize, Lebanon through a joint program in 2011. Pierre received a Ph.D. in Computing from The Open University, U.K. in 2014. Since then, he has been an assistant professor of computer science at Notre Dame University – Louaize, Lebanon and a visiting research fellow in computing at The Open University, U.K. Previously, he had worked part-time in academia as a computer science instructor and full-time in industry as an application architect focusing on enterprise systems. Pierre is mainly interested in researching adaptive model-driven interactive software systems. His work on adaptive model-driven user interfaces was published in several venues including ICSE and ACM Computing Surveys, and it received the best paper award at ACM SIGCHI EICS'13. Further information about Pierre's research work and publications can be found at: http://www.pierreakiki.com.

Arosha K. Bandara received a MEng and PhD from Imperial College London in 1998 and 2005 respectively. He is now a senior lecturer in computing at The Open University, where he is a member of the Software Engineering and Design research group and the Security and Privacy Laboratory. Arosha's research focusses on building and maintaining self-managing (adaptive) systems by combining rigorous formal techniques with concrete implementations and applications of those techniques. He is a co-investigator on projects funded by the European Research Council (Adaptive Security and Privacy), the UK Engineering and Physical Sciences Research Council (Privacy Dynamics, Monetize Me), and the Qatar National Research Foundation (Adaptive Information Security).

Yijun Yu graduated from the Department of Computer Science at Fudan University (B.Sc. 1992, M.Sc. 1995, Ph.D. 1998). He was a postdoc. Research Fellow at the Department of Electrical Engineering in Ghent University, Belgium (1999-2002), then a Lecturer at the Department of Computer Science in University of Toronto, Canada (2003-2006). Since October 2006, he has become a Senior Lecturer in Computing at The Open University, UK. He is interested in developing automated and efficient software techniques and tools to better support human activities in software engineering. He was interviewed by BBC Radio 4 Today and BBC World Services on his vision of applying Cloud Computing to improve Aviation Security. His research won an ACM SIGCHI Best Paper Award at EICS'13, an IEEE Best Paper Award at TrustCom'14, an IEEE Distinguished Paper Award at RE'11, a BCS Distinguished Paper award at BCS'08, and an ACM SigSoft Distinguished Paper Award at ASE'07. He serves as an Associate Editor of the Software Quality Journal, Secretary of BCS Specialist Group on Requirements Engineering, and PC of international conferences on Requirements Engineering (RE'10-16, CAiSE'10-16), Software Maintenance (SEW'13, CSMR'12, ICSM'10) and IoT (WF-IoT'15-16). He is a member of the IEEE and the British Computer Society. He is a PI for successful knowledge transfer projects at Huawei, IBM and a co-investigator on projects of Adaptive Security And Privacy (ERC Advanced Grant, 2012-2017), Relating Requirements and Designs of Adaptive Information Systems (QNRF, 2012-2015), Lifelong Security Engineering for Evolving Systems" (EU FP7, 2009-2011), Usable Privacy for Mobile Apps (Microsoft SEIF, 2012).