USING INTERPRETED RUNTIME MODELS FOR DEVISING ADAPTIVE USER INTERFACES OF ENTERPRISE APPLICATIONS

Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu
Computing Department, The Open University, Walton Hall, Milton Keynes, United Kingdom
{pierre.akiki,a.k.bandara, y.yu}@open.ac.uk

Keywords: User Interfaces: Model Driven Engineering: Runtime Modelling: Enterprise Applications: Design Tools and Techniques: Domain-Specific Architectures: Software Architectures

Abstract: Although proposed to accommodate new technologies and the continuous evolution of business processes and business rules, current model-driven approaches do not meet the flexibility and dynamic needs of feature-rich enterprise applications. This paper illustrates the use of *interpreted runtime models* instead of static models or generative runtime models, i.e. those that depend on code generation. The benefit of interpreting runtime models is illustrated in two enterprise user interface (UI) scenarios requiring adaptive capabilities. Concerned with devising a tool-supported methodology to accommodate such advanced adaptive user interface scenarios, we propose an adaptive UI architecture and the meta-model for such UIs. We called our architecture **C**ustom **E**nterprise **D**evelopment **A**daptive A**r**chitecture (CEDAR). The applicability and performance of the proposed approach are evaluated by a case study.

1 INTRODUCTION

Modern businesses rely heavily on enterprise software applications for automating their business processes. The dependency on these applications drives business owners to request even more features from the software suppliers. It places a heavy pressure on suppliers to provide the best possible software quality, without increasing the cost. The orientation towards generic enterprise applications (ERP, CRM, etc.) is also being challenged by the variation of demands amongst businesses and users.

Among various components of an enterprise system, the user interface (UI) layer is considered highly important since it interfaces users to the software system. Some software companies chose to build multiple UIs for the same functionality due to variable user needs. Yet in certain situations the scope of variability is unknown at design time or it is costly to develop multiple UI versions manually.

User interface simplicity is an important requirement for enterprise application users. Some novice users prefer the UI to be displayed in a step-by-step wizard whereas advanced users might feel more productive if the UI is displayed on one page. Generally, different users require a variable part of the software's feature set, which could scatter across multiple user interfaces. Displaying a significant UI subset in one place would help users fulfil their repetitive tasks more efficiently.

One method to achieve UI simplification is for enterprise applications to be adaptive/adaptable, respectively referring to the ability of tailoring software applications automatically/manually.

A more detailed explanation on the adaptive UI simplification is given in Section 2 through two practical scenarios. We should emphasize that the objective of this paper is not to solve both scenarios. Instead, we intend to propose a general-purpose solution for creating enterprise applications targeting such adaptive UI scenarios. One of the scenarios will be partially addressed as a case study in Section 7.

We adopt a model-driven approach for devising adaptive/adaptable UI. Hence we differentiate between the following model-driven approaches:

Static modelling is an approach that relies on models for UI design and eventually ends in a phase before code generation. By definition **static models** cannot change at runtime, hence are not suitable to be used beyond the development phase.

Most adaptive model-driven UI approaches in the literature depend on **generative runtime models** of application artefacts that reuse the code already implemented as a generic UI.

Runtime models are usually more opted for adaptive features. However, in certain scenarios such as those discussed in Section 2, using runtime models while maintaining the generated code-based artefacts is insufficient. Features required in such adaptive scenarios include runtime support for actions such as eliminating widgets; replacing a widget with another; adding new widgets; or composing a new UI from existing user interfaces.

In contrast, our approach uses **interpreted runtime models** such that there is no need to generate code for creating the UI. Instead, the models are interpreted at runtime to render the UI.

2 ADAPTIVE SCENARIOS FOR ENTERPRISE APPLICATIONS

Adapting UI functionality through automatic simplification could make complex applications easier to use on mobile devices and by people with cognitive impairments (Gajos et al. 2010). Tailored UIs could enhance user satisfaction (McGrenere et al. 2002) but the manual development cost is high.

The following scenarios are examples for clarifying the importance of our approach.

Scenario 1: Simplifying Individual User Interfaces could be based on: "Elimination", "Substitution", and "Realignment" of UI widgets.

We could adjust the UI per user by eliminating unused features and also consider user level layout adaptation. The following is one possible example:

1. **Beginner**: Present UI in wizard form
2. **Intermediate**: Divide UI among several tabs
3. **Expert**: Display UI widgets on one page

Scenario 2: Composing New Functionality from Existing User Interfaces is related to dynamic "Composition" of new UIs based on existing ones (defined at design time) and end user behaviour.

One possible application would be on scattered UIs, which is the case of entering information for an inventory item in Microsoft Dynamics GP. The main information entry is done through one UI form. Yet various sets of item related information (Prices, Options, etc.) are entered in separate UI forms.

UI composition and decomposition has been addressed in some research works (Lepreux et al. 2010). Yet the researchers focused on performing those actions at design time.

3 RELATED WORK

This section briefly summarizes the existing work that could be classified into reference architectures and state of the art with possible gaps.

3.1 Architectures

Architectures, which could serve for the purpose of designing UIs and adaptive systems in general, could be classified into the following categories:

1. ***User Interface Abstraction*** is concerned with the representation of UIs on multiple levels of abstraction. The CAMELEON reference framework is one example.
2. ***Adaptive System Layering*** provides a reference model for adaptive systems in general. Existing work includes the Three Layer Architecture and IBM MAPE-K loop.
3. ***Implementation*** architectures deal with the distribution of components in a development scenario. Common architectural patterns of this sort include: MVC, MVP, and MVVM.

We will base our proposed architecture on the Three Layer Architecture (Kramer & Magee 2007), CAMELEON (Calvary et al. 2003), and MVC.

3.2 State of the Art

Runtime models constitute an important area of research in MDE (France & Rumpe 2007). Existing research works target adaptive UI differently.

The Multi-Access Service Platform (MASP) targets ubiquitous UI in smart environments and promotes runtime modelling but still relies on code for defining the initial UI (Blumendorf et al. 2010).

Supple is introduced as a system mainly capable of generating interfaces adapted to each user's motor abilities (Gajos et al. 2010). Although the adopted technique generates the UI from an abstract model, it does not support the various possible levels of abstraction and designer input on the concrete UI.

The COntext Mouldable widgeT (Comet(s)) was introduced to support UI plasticity (Calvary et al. 2005). Comets tend to target adaptation of individual widgets while our target is the entire layout.

DYNAmic MOdel-bAsed user Interface Development (DynaMo-AID) is presented as part of the Dygimes UI framework (Clerckx et al. 2004). This system is mostly concerned with simple mobile applications. Furthermore, the adopted approach for generating task trees could lead to a combinatorial explosion making it hard to use for large scale enterprise applications.

4 PROPOSED ARCHITECTURE

Our proposed architecture for enterprise applications with adaptive UI capabilities (CEDAR) is illustrated in Figure 1. The proposed artefacts column illustrates the distribution of the adaptive components according to each of the reference architectures (Three Layer Architecture, CAMELEON, and MVC) discussed in Section 3.1.

4.1 Adaptive Components

This section will elaborate on the function of each of the adaptive components under the four layers.

L1 - Client Components Layer: The components in this layer will be deployed to the client machine.

The "Context Monitor" will be responsible for monitoring any changes in the current context. This component was allocated to the client since it would be able to monitor changes to the environment in addition to any changes in the user's behavior.

The ability to cache data on the client will provide dynamically generated systems with much better performance. The "Caching Engine" will be responsible for caching any part of the model.

The "UI Renderer" will be responsible for rendering the UI model using one of the existing presentation technologies. Additionally, this component will be responsible for managing events, data binding, and validation by linking the dynamic UI layout to the application code behind.

L2 - Decision Components Layer: These components will be deployed to the application server and will handle decision making in the adaptive scenario.

The "Context Evaluator" will handle the information submitted by the "Context Monitor" in order to evaluate whether the change requires the models to be adapted.

The "Caching Engine" on the application server will assume a role similar to that of its counterpart on the client. Yet in this case the caching will not be made on the session level for each individual user but on the application level for all the users.

L3 - Adaptation Components Layer: These components will be deployed to the application server and will be responsible for performing the actual adaptation on the models.

The "Adaptive Engine" will be responsible for taking a UI model as input and conducting the adaptation according to one of the adaptive models.

The "Trade off Manager" assumes the role of balancing the trade-offs between the different adaptation constraints in order to meet each set of constraints as much as possible.

The "UIDL Converter" will be responsible for handling the conversion between the user interface model (*stored as relational data*) and the necessary User Interface Description Language (UIDL).

L4 - Adaptive & User Interface Models Layer: The adaptive and UI models will be stored on the database server. A relational database will be used for managing the various required models.

The adaptive models will represent a generic rule set according to which the UI models will be adapted. Such rules will be based on the various adaptive factors relevant to the changing contexts.

4.2 Adaptive Procedure & Advantages

Two main approaches could be considered for adapting the UIs of enterprise applications. The following paragraphs explain the procedure, which could be mapped to steps S1 to S5 on Figure 1.

The first approach is a direct adaptation. A change in the context gets reported (**S1**) to the "Context Evaluator". A decision is made on whether the UI should be adapted. The adaptive engine is called (**S2**) for obtaining the new UI. The adaptive engine will send the adapted UI back for caching (**S4**). Then it will be transferred to the client and modified on the fly (**S5**).

The second approach differs from the first by the method through which the adapted UI is handed to the user. Instead of modifying the UI while the user is working, the adapted version (**S2**) is stored (**S3**) and the UI is proposed as a new option (**S5**). This could be more convenient in many enterprise scenarios such as those described in Section 2. The convenience lies in preventing the user from being confused by a UI that is constantly changing.

An advantage of the proposed architecture is the separation of concerns allowing the adaptive functionality to be consumed as a generic service. Additionally, the layering conceptually allows the integration of various adaptive models, which in turn allow the UI to adapt according to different factors. Previous research works (Section 3.2) focus on adapting the UI according to specific adaptive factors (Screen size, physical impairments, distance from display devices, etc.). A general architecture could be considered as a more extensible method in terms of accommodating various types of adaptive factors within a generic middleware.

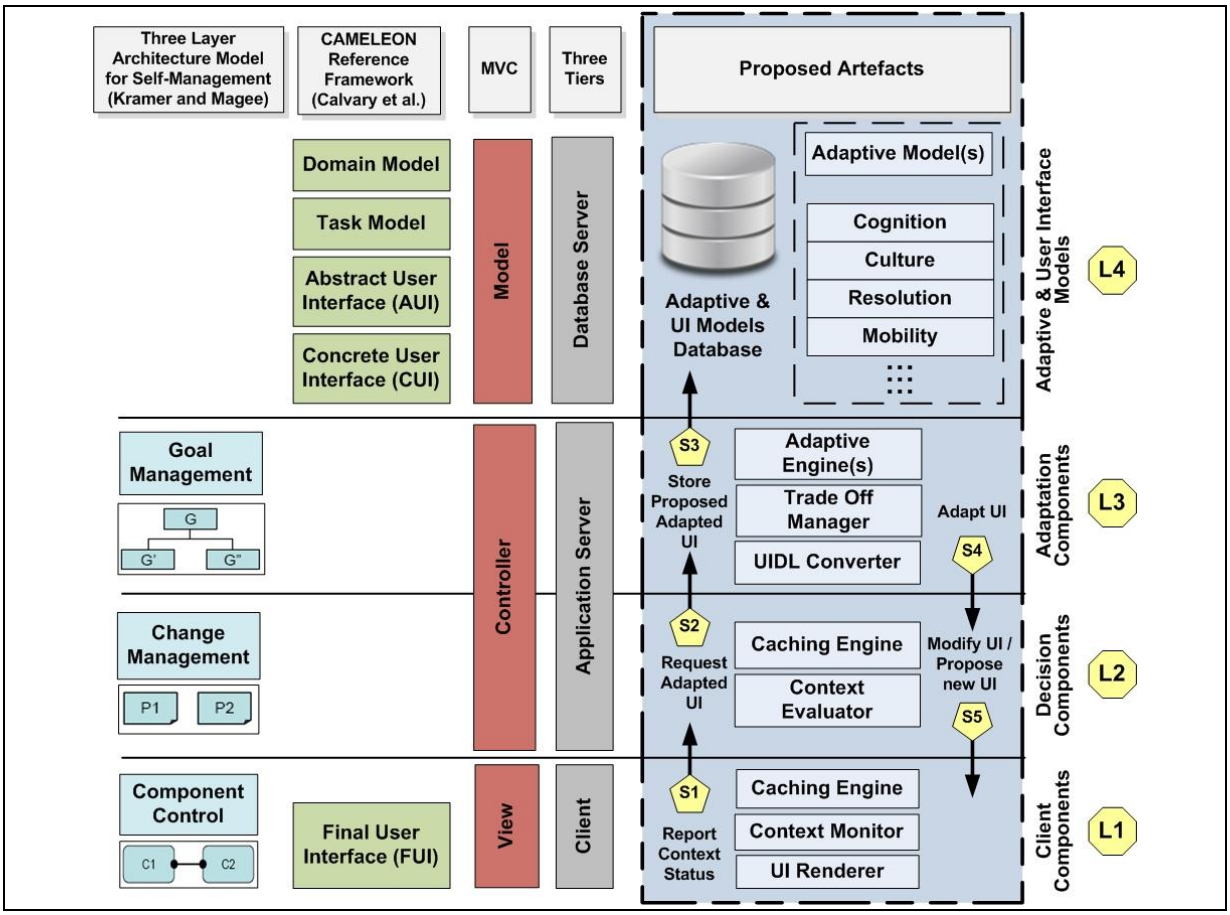

Figure 1: Proposed Architecture for Adaptive User Interfaces

5 UI META-MODEL

User Interface Description Languages (UIDLs) are used to define technology and modality independent UI. Several UIDLs (UsiXml, UIML, XIML, etc.) currently exist. Yet UsiXml is considered to have the most comprehensive meta-model complying with the CAMELEON reference framework. Additionally, it is possible to define mappings and transformations between the various levels of abstraction (Tasks & Domain Model, AUI, CUI, and FUI). Hence we chose to rely on UsiXml's meta-models (Guerrero-Garcia et al. 2008) for UI persistence and transfer. Currently we are only working with the CUI and the domain model. UML class diagrams are used to represent domain models whereas UsiXML's meta-model is used for the CUI.

As indicated in its definition (*www.usixml.org*), UsiXml is not intended to handle all attributes and events of all widgets in all toolkits but merely a subset. Yet our dynamic approach would not allow the UI layout to be defined through code. Hence we required a level of abstraction capable of making the model extensible to support a vast subset of features from different technologies. To achieve that, we define a UI widget in terms of its "Properties" and "Events" and allow the designer to extend those according to different technology profiles. Binding the UI to the data model is also considered in the meta-model by defining a "Data Binding" capable of linking a "Component Property" to a class diagram "Property". To validate the input values, "Validation Rules" could be defined on the data-bindings for checking a value before committing it to the data source. We should note that setting the property values in addition to tying up the events, and bindings will be fully conducted at runtime through the "UI Renderer" depicted in Figure 1.

In order to link the layout to the code behind, the developer will have to attach a "Code Behind Method" to a widget event in a similar manner to how it is done under a regular IDE.

6 TOOL SUPPORT

The CUI designer of the IDE we devised for creating enterprise UIs with runtime adaptive abilities is illustrated in Figure 2. Although the architecture is intended to encompass the various abstraction levels of the CAMELEON framework the tool support at this stage is limited to the CUI and Domain Model.

Developers' productivity and their understanding of the methodology are critical for maintaining a reasonable software development cost. Since many developers tend not to understand modelling very well, we adopted a familiar development approach. Our tool encompasses a visual designer for UI development, which is quite similar to those present in widely adopted IDE's such as Visual Studio.NET, NetBeans, Eclipse, etc. This type of tool will allow developers to create the user interface in a traditional manner by dragging and dropping widgets onto a canvas. Additionally, developers could click on each widget in order to adjust its properties or to tie up its events to a code behind method.

This tool was developed with C# using both Windows Forms and the Windows Presentation Foundation (WPF) for the UI. Currently the model related data is being stored in an SQL Server 2008 database but other database management systems could be also used. The adaptive middleware was developed using the Windows Communication Foundation (WCF) in order to make it accessible from anywhere (*web or intranet*) as a service. To test out our approach we had to develop a rendering engine for at least one presentation technology. WPF was the technology of choice but with the existence of the meta-models the UI rendering engine could be easily adapted for other technologies as well.

As previously noted this tool is not fully developed since we still need to incorporate visual designers for the abstract UI and task trees. Adding those will provide full tool support for the proposed architecture and the ability to adapt the UI at the different levels of abstraction. This will be done by keeping in mind the need to maintain a familiar development approach. In spite of that, at this stage developers could use the tool to create fully functional UIs with the existing designers.

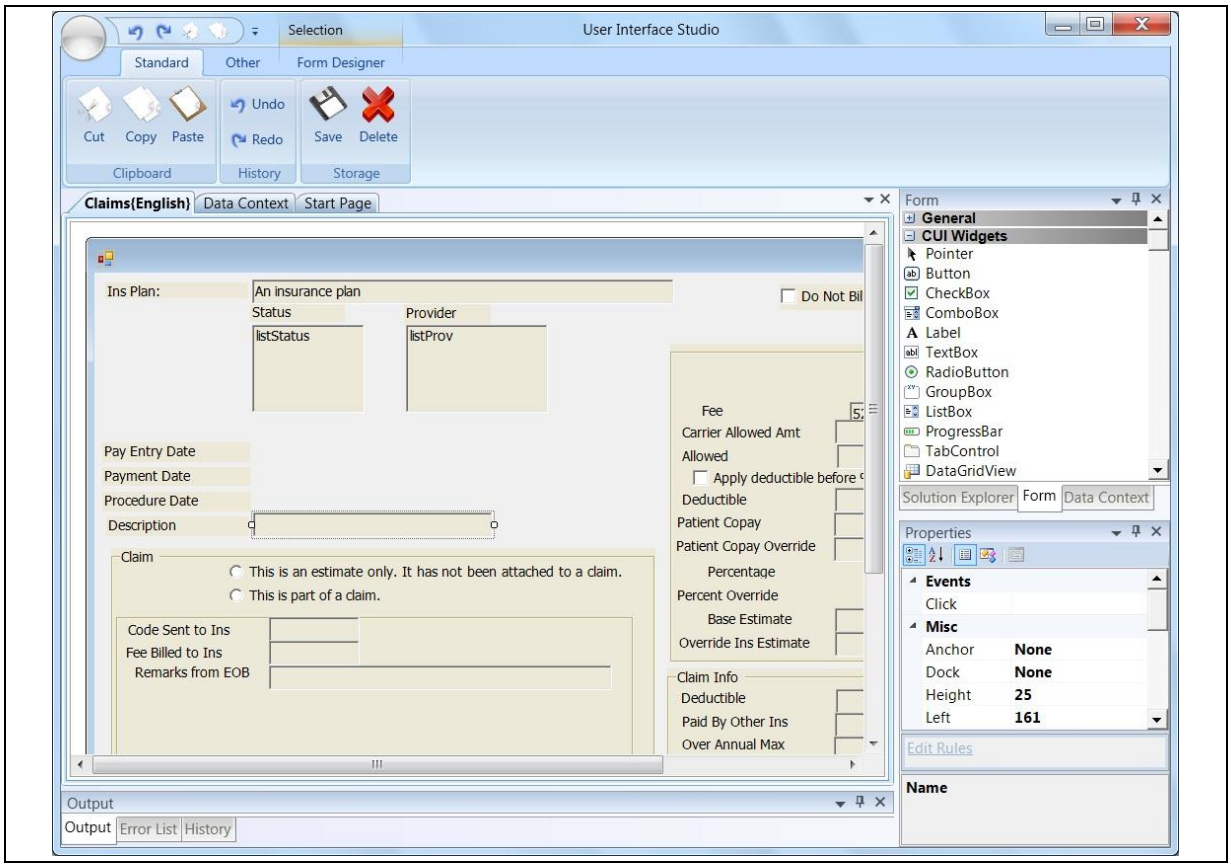

Figure 2: Our Concrete User Interface Designer

7 EVALUATION CASE STUDY

To assess our proposal, we conducted a case study based partially on **Scenario 1** discussed in Section 2.

The standard for role based access control (RBAC) could be utilized by enterprises for protecting digital resources (Ferraiolo et al. 2001). In RBAC, "Users" are assigned "Roles", which are in turn assigned permissions on "Resources". In our case, the UI is the resource we need to secure.

Table 1: CRUD to UI Property Mappings

CRUD Permission	UI Property	Value
Allow / Deny (Create)	isEnabled	True / False
Allow / Deny (Delete)	isEnabled	True / False
Allow / Deny (Read)	isVisible	True / False
Allow / Deny (Update)	isEnabled	True / False

Table 1 lists the mapping between the CRUD permissions and UI-specific properties. The "Create" and "Delete" permissions are applied on the domain model UML classes whereas "Read" and "Update" are applied on UML class properties.

To demonstrate that the proposed method is not only meant for newly developed applications we chose an existing open source dental practice software called OpenDental (*www.opendental.com*). We selected the "Claims" form, illustrated in the UI studio in Figure 2. It has 87 widgets of 9 different types, and was reverse engineered from code into relational data based on our proposed meta-model.

We tested the performance of the dynamic UI, which loads all the widgets at runtime from a database, versus the code based compiled UI.

Both versions of the "Claims" form were loaded and closed 1000 times. The time was plotted on the graph illustrated in Figure 3. The dynamic UI took slightly more time when it was loaded the first time then the caching allowed a significant drop in the time. Overall we could say that our approach will not incur negative impact on performance.

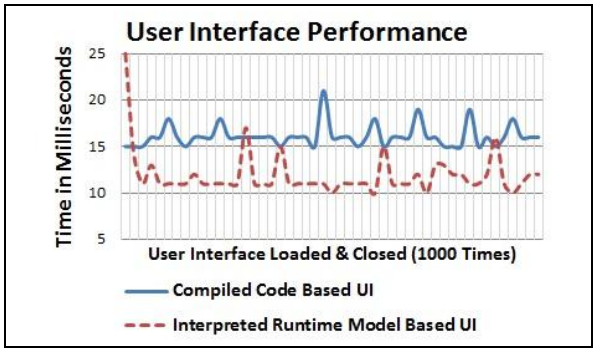

Figure 3: User Interface Performance

8 CONCLUSIONS

Adaptive user interfaces could be considered as a means for addressing variations in the needs of enterprise application users without incurring a high increase in the cost of developing such applications.

In this paper, we have presented an approach that uses interpreted runtime models for creating enterprise applications, which makes it easier to realize both adaptive and adaptable user interfaces. Additionally, the dynamic model-driven nature of the proposed method could make enterprise applications more resilient to change in both technology and business requirements.

In the future we will adopt the proposed approach as a basis for devising an adaptive solution for the scenarios discussed in Section 2.

REFERENCES

Blumendorf, Marco, Lehmann, Grzegorz and Albayrak, Sahin (2010) 'Bridging Models and Systems at Runtime to Build Adaptive User Interfaces'

Calvary, Gaëlle, Coutaz, Joëlle, Dâassi, Olfa, Balme, Lionel and Demeure, Alexandre (2005) 'Towards a New Generation of Widgets for Supporting Software Plasticity: The "Comet"'

Calvary, Gaëlle, Coutaz, Joëlle, Thevenin, David, Limbourg, Quentin, et al. (2003) 'A Unifying Reference Framework for Multi-Target User Interfaces'

Clerckx, Tim, Luyten, Kris and Coninx, Karin (2004) 'DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development'

Ferraiolo, David F., Sandhu, Ravi, Gavrila, Serban, Kuhn, D. Richard and Chandramouli, Ramaswamy (2001) 'Proposed NIST standard for role-based access control'

France, Robert and Rumpe, Bernhard (2007) 'Model-Driven Development of Complex Software: A Research Roadmap'

Gajos, Krzysztof Z., Weld, Daniel S. and Wobbrock, Jacob O. (2010) 'Automatically Generating Personalized User Interfaces with Supple'

Guerrero-Garcia, Josefina, Vanderdonckt, Jean and Gonzalez-Calleros, Juan Manuel (2008) 'Towards a Multi-User Interaction Meta-Model'

Kramer, Jeff and Magee, Jeff (2007) 'Self-Managed Systems: an Architectural Challenge'

Lepreux, Sophie, Vanderdonckt, Jean and Kolski, Christophe (2010) 'User Interface Composition with UsiXML'

McGrenere, Joanna, Baecker, Ronald M. and Booth, Kellogg S. (2002) 'An Evaluation of a Multiple Interface Design Solution for Bloated Software'