Integrating adaptive user interface capabilities in enterprise applications

Conference or Workshop Item

Integrating Adaptive User Interface Capabilities in Enterprise Applications

Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu
Computing and Communications Department
The Open University
Milton Keynes, United Kingdom
{pierre.akiki, arosha.bandara, yijun.yu}@open.ac.uk

ABSTRACT
Many existing enterprise applications are at a mature stage in their development and are unable to easily benefit from the usability gains offered by adaptive user interfaces (UIs). Therefore, a method is needed for integrating adaptive UI capabilities into these systems without incurring a high cost or significantly disrupting the way they function. This paper presents a method for integrating adaptive UI behavior in enterprise applications based on CEDAR, a model-driven, service-oriented, and tool-supported architecture for devising adaptive enterprise application UIs. The proposed integration method is evaluated with a case study, which includes establishing and applying technical metrics to measure several of the method's properties using the open-source enterprise application OFBiz as a test-case. The generality and flexibility of the integration method are also evaluated based on an interview and discussions with practitioners about their real-life projects.

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

General Terms
Design; Human Factors

Keywords
Adaptive user interfaces; enterprise systems; software architectures; model-driven engineering; integration; software metrics

1. INTRODUCTION
Existing research shows that adaptive user interfaces (UIs) can help enterprise applications to overcome some of their usability problems by tailoring their off-the-shelf UIs to each end-user's needs [2]. Yet, many enterprise applications incorporate hundreds or even thousands of UIs and are already at a mature stage in their development. A method is needed for integrating adaptive UI capabilities into these systems, without incurring a high development cost or significantly changing the way they function.

In his paper on criteria for evaluating UI research, Olsen [29] gives an example about the objections that were made in the late 1970s towards new UI architectures due to the large amount of legacy code written for command-line or text UIs. He notes that legacy code can be a barrier to progress hence, if rewriting applications is necessary, it could be the price of progress. Yet, Olsen also states that providing a new advance while maintaining legacy code is desirable. The latter is what we aim to achieve with our method for integrating adaptive UI capabilities in enterprise applications.

Another integration challenge lies in the difference between research work on adaptive user interfaces presented in the literature and traditional UI development techniques. For example, many research works on adaptive UIs adopt the model-driven approach to UI development either partially (e.g., Supple [20]) or fully (e.g., MASP [10]). However, despite the advantages of the model-driven approach, the user interfaces of many existing software systems including enterprise applications have been developed using traditional techniques. Therefore, an important issue to consider for adaptive UI integration in existing applications is the means of combining new UI development approaches such as the model-driven approach with UIs that have been built using existing UI design tools such as interface builders.

This paper contributes a method for integrating adaptive UI capabilities in enterprise applications without the need for a major integration effort. We evaluated our method by establishing and applying technical metrics to measure several of its properties using the open-source enterprise application Apache Open for Business (OFBiz) as a test-case. This evaluation covered different phases including: reverse-engineering, integration, and runtime execution. We also evaluated the method's generality and flexibility based on an interview and discussions with industry practitioners and data from their real-life enterprise system projects.

Our proposed method in this paper is based on CEDAR [1], a model-driven, service-oriented, and tool-supported architecture for devising adaptive enterprise application UIs. Using an architecture for adaptive systems is promoted [23] since it provides generality, abstraction, and a potential for scalability. Our proposed method is applicable as a generic solution for adapting the UIs of different enterprise applications. Also, the abstraction provided by CEDAR offers a high-level understanding of the UI adaptation process for stakeholders interested in adopting it as a reference for devising adaptive UIs. Furthermore, UI adaptation mechanisms that are based on CEDAR are bundled as a separate system and made accessible through web-services, thereby creating a loose coupling with potential for scalability, and facilitating the integration in large-scale enterprise applications.

The type of UI adaptation that we applied in the evaluation of our integration method is UI simplification using our Role-Based UI Simplification (RBUIS) mechanism. In a previous work [2], we presented RBUIS as a mechanism based on CEDAR for providing end-users with a minimal feature-set and an optimal layout based on the context-of-use, and showed that it can improve end-user

Authors' Version
ICSE'14, Hyderabad, India

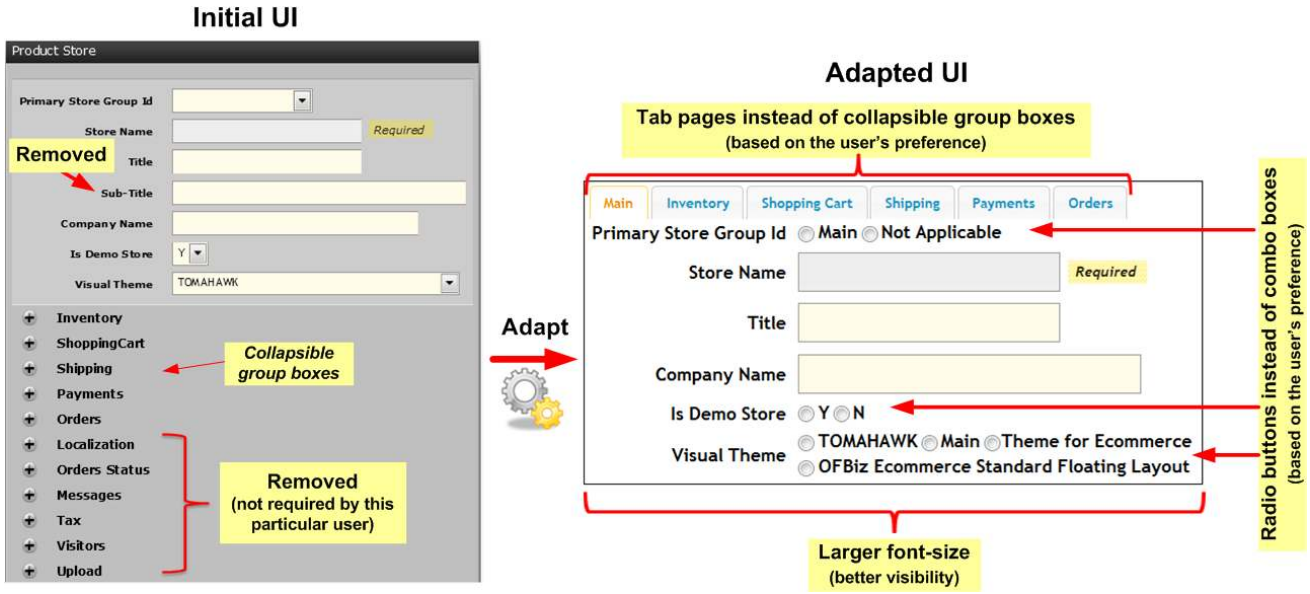

Figure 1. An Example on Adapting the "Product Store" User Interface of the OFBiz Enterprise Application

satisfaction and efficiency through a usability study. We define a feature as a functionality of the software 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 a set of factors such as: the user's skills and motor abilities, hardware devices, etc. An optimal layout is achieved by adapting concrete widget properties such as the type, grouping, size, location, etc. The example shown in Figure 1 was part of the evaluation of our integration method. It demonstrates feature-set minimization and layout optimization operations on the "Product Store" UI of OFBiz. Our adaptation and integration mechanisms can be observed in operation through demonstration videos [33].

The remainder of this paper is structured as follows: Section 2 briefly discusses the related work. Section 3 provides an overview of the CEDAR architecture and presents our technique for integrating adaptive UI capabilities in enterprise applications based on CEDAR and using OFBiz as a test-case. The metrics we established for evaluating the different phases of our method are presented in Section 4 and applied to scenarios from OFBiz. In Section 5, we assess the generality and flexibility of our method. The threats to validity and limitations are presented in Section 6, and the conclusions and future work are given in Section 7.

2. Related Work: Adaptive UI Solutions

In this section, we shall briefly cover the prior art for UI adaptation solutions and argue their strength and shortcomings in terms of how they integrate in existing software systems.

2.1 Architectures

Several architectures were proposed as a reference for applications targeting adaptive UIs. CAMELEON-RT [5] is an architecture for distributed, migratable, and plastic UIs. However, it only serves as a high-level reference without providing low-level implementation specifications including information on integrating in existing systems. Lehmann *et. al.* [25] proposed an architecture for devising adaptive smart environment UIs, which was only applied to the development of new prototype systems. Malai was presented as an architectural model for interactive systems [8]. In Malai, developers have to define several code-based presentations for the same UI at design-time. In addition to being technology dependent (a Java example is provided), UI adaptation in Malai is not decoupled from the target software systems thereby requiring significant code modification to the system. With our method, we aim to provide specifications on integrating with existing systems and to decouple the UI adaptation mechanism from the target enterprise application.

2.2 Techniques

Some works on UI adaptation such as: "multi-layered UI design" [32], "two UI design" [27], and "training wheels UI" [15], present a theoretical basis for UI adaptation but do not offer an engineering solution for applying their propositions in practice. Other existing works with practical solutions can be classified as follows:

Toolkit-based approaches for adaptive UIs have been explored extensively in the literature (e.g., caring, sharing widgets [24], selectors [21], swing states [4], etc.). Technology dependence is one of the disadvantages of toolkits in comparison to model-driven UIs. This disadvantage could impact the integration of adaptive UI toolkits in existing enterprise applications since the entire toolkit has to be redeveloped for each technology. Providing technology-independence is an important part of our CEDAR-based adaptation mechanism as will be explained in Section 3.1. The Comet(s) [13] attempts to combine the toolkit and model-driven approaches for building adaptive UIs. Nevertheless, even if the toolkit was technologically compatible with an existing enterprise system, the amount of code modification that is required to switch the UI from the classical toolkit to the adaptive one could be significant. This is especially true if the enterprise application's UI was not developed by following design patterns such as a "bridge" to decouple each widget's abstraction from its implementation. In such a situation, a conversion tool is necessary with some manual work for shifting the UI specification from one toolkit to another. Our approach can operate on existing UIs without having to update them to a new toolkit due to the separation of concerns between the adaptation mechanism and the technology dependent UI representation.

Aspect-oriented programming (AOP) was proposed for improving the separation of concerns in software systems [22]. One approach that used AOP for adapting UIs requires several presentations to be

defined for the same UI at design-time and a weaver is used to associate these presentations to instrument classes that handle the way the UI functions [9]. Our approach is conceptually similar to AOP since we are trying to achieve a separation of concerns between the UI adaptation technique and the enterprise system. Yet, our main focus is on adapting the UI's presentation and not its code-behind functionality. From this perspective, the existing AOP-based approach requires UI variations to be defined manually by developers at design-time, whereas our approach aims at adapting UIs through adaptive behavior using rules that could be applied to different UIs at runtime. For example, a rule could be defined to switch the way the UI's widgets are grouped by changing group boxes to tab pages. Adaptation rules defined outside the enterprise system could save integration time and support dynamic changes that narrow the gap between development-time and runtime.

Design-time model-driven approaches rely on ***generating*** multiple adapted UIs based on models that represent the UI at several levels of abstraction. Approaches based on software product-lines (SPL) [31] are used to tailor software systems in general and some, such as MANTRA [12], particularly target tailoring UIs. SPLs can be dynamic [7]. However, SPL-based UI adaptation approaches focus on design-time adaptation such as generating UIs with different subsets of features based on a feature model, whereas runtime adaptive behavior is not addressed. Smart templates are another generative approach and were used with ubiquitous remote control mobile UIs [28]. Code generation makes such approaches difficult to adopt for existing mature enterprise applications due to the amount of effort needed to integrate the generated code in the existing systems and the increased number of software artifacts that can require maintenance. Also, if the adopted presentation technology required compilation (e.g., Windows Forms) adding UI artifacts would increase the compilation time. Our integration method requires a few lines-of-code to be added to the enterprise application at design-time to trigger UI adaptations at runtime. Therefore, our approach can be integrated without major design-time effort or the need for a large number of new software artifacts.

Runtime model-driven approaches keep the models alive at runtime for adapting the running UI dynamically. Some are ***generative***, thereby generate an individual UI specification from the models at design-time and use the models to adapt this UI at runtime. MASP [11] follows this approach and targets ubiquitous UIs in smart environments. MASP does not provide specifications on integrating with existing systems and it was evaluated by (re)building home automation applications such as: energy, cooking, and health assistants. Other approaches such as Supple and DynaMo-AID rely on ***interpreting*** the models and dynamically rendering the UI. Supple [20] is a system, which primarily targets generating UIs that are automatically adapted to each user's motor abilities. DynaMo-AID is a design process and runtime architecture for devising context-aware UIs [17]. Both Supple and DynaMo-AID did not demonstrate and evaluate the ability to integrate their proposed approaches in existing software systems. Supple was evaluated by developing a variety of simple UI dialogs (e.g., email client, ribbon, print dialog, etc.) and DynaMo-AID was used to develop a tourist guide mobile application. An additional point that is neglected by existing runtime model-driven approaches is the support for user ***feedback*** on the adapted UI. Supporting feedback in adaptive systems is promoted for keeping users involved in the adaptation process to insure their trust [16]. Nevertheless, it could also play an important role in reducing development and integration efforts. Tuning the adaptation according to each user's needs can take several cycles of development, deployment, user-testing and change reporting. These cycles can be shortened by empowering users to report changes directly to the system using a feedback mechanism. Most existing works on adaptive UIs do not focus on feedback. One exception is Supple [20], which supports user-feedback for runtime elicitation of the adaptation rules. However, the sole reliance on runtime elicitation could be time consuming especially in large-scale enterprise applications and might not provide sufficient data. With our CEDAR-based approach we allow an initial definition of the adaptation rules (e.g., based on expert knowledge) and rely on user feedback for further tuning.

We think that runtime model-driven UI development is the most suitable approach to support a method for integrating adaptive UI capabilities in existing enterprise applications due its dynamic nature. Yet, the lack of attention from existing works in the literature towards integration drives us to present an integration method based on our CEDAR architecture. Section 3 provides an overview of CEDAR and explains our method for integrating adaptive UI capabilities in enterprise applications using OFBiz as a test-case.

3. INTEGRATING ADAPTIVE UIS IN OFBIZ

This section provides an overview of CEDAR and the way of using it for integrating adaptive UI capabilities in enterprise systems. The open-source enterprise application OFBiz is used as a test-case.

Apache Open For Business (OFBiz) [34] is an open-source enterprise automation software project that contains several sub-systems such as: Enterprise resource planning (ERP), manufacturing resources planning (MRP), customer relationship management (CRM), e-business and e-commerce, and supply chain management (SCM). It could be considered as a general-purpose, large-scale, enterprise system having the characteristics shown in Table 1.

Table 1. Some of OFBiz's Characteristics

OFBiz Release 12.04	
Number of User Interfaces	> 750
Number of Lines-of-Code	≈ 1,466,000
Projects Based on OFBiz	20
Public Sites using OFBiz	90

Although commercial enterprise systems can be larger, for example SAP has over 250,000,000 lines-of-code [35] and Lawson has over 10,000 UIs [36], OFBiz has complex UIs with a large number of widgets that may need adaptation making it a good candidate for our study. For example, the main UIs from its Catalog module have an average of 55 widgets and a maximum of 170. Also, an open-source system is necessary to test our integration method. Our method could work with commercial systems but the company that owns the source-code should perform the integration.

3.1 The CEDAR Architecture

This section offers an overview of CEDAR [1], and the way we used it for integrating the RBUIS [2] UI adaptation mechanism in OFBiz as shown in Figure 2. The CEDAR architecture serves as a reference for stakeholders interested in developing adaptive enterprise application UIs based on a model-driven approach. It promotes the use of interpreted runtime models, which allow UIs to be loaded, adapted, and rendered dynamically without resorting to code generation. Although CEDAR has the potential to make the UIs of software systems adaptive, it had not been integrated with complex enterprise applications.

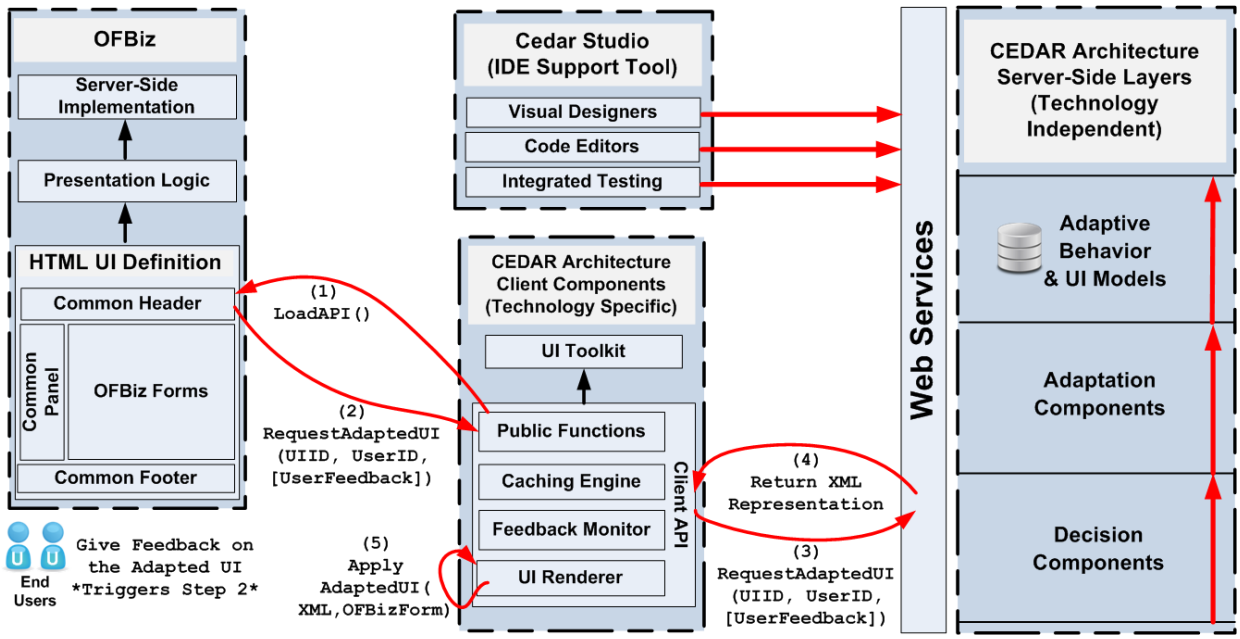

Figure 2. Integrating Adaptive User Interface Capabilities in OFBiz based on our CEDAR Architecture

As illustrated in Figure 2, CEDAR has three server-side technology-independent layers. The *decision components* handle decision making in various adaptive UI scenarios such as evaluating whether a change in the context-of-use requires the UI to be adapted. The *adaptation components* are mainly responsible for adapting the UI models by executing the appropriate adaptive behavior on them. The *adaptive behavior and UI models layer* hosts the models that comprise the different levels of abstraction representing the UI. These levels of abstraction follow the *CAMELEON* [14] framework and include task, domain, abstract UI (AUI), and concrete UI (CUI) models. The adaptive behavior is also hosted on this layer and could be represented visually as workflows or using scripts that dictate how the UI models are adapted for the different contexts-of-use. The components of a server-side layer can access those of the layer above it as depicted by the vertical arrows in Figure 2 (right).

CEDAR's client components integrate in enterprise applications and empower them with adaptive UI capabilities as illustrated by arrows (1) to (5) in Figure 2, using OFBiz as an example. These components are dependent on the programming and presentation technologies, since they have to be integrated in the enterprise application's code. Hence, different sets of components are required. These components offer an application programming interface (API) that is loaded globally (1) in the enterprise application (e.g., common header in OFBiz). Whenever the end-user launches a UI, a request is made to the API for adapting this UI; the identifiers of the end-user and the UI are passed as parameters (2). The API uses web-services to pass the UI adaptation request to the server-side layers (3), which perform the adaptation and return the result to the API as XML (4). The API's *UI Renderer* is responsible for applying the adaptation result to the running enterprise application UI, which is an HTML page in the case of OFBiz. Once a UI is adapted, the *Caching Engine* is responsible for caching the adapted version on the client-side in case the end-user requests it again. Adaptive UI mechanisms can affect an end-user's UI control [27]: End-users might feel loss of control if the adaptive UI mechanism makes decisions they cannot understand or change. Reduction mechanisms can affect feature-awareness [19]: If a UI was adapted by reducing features without providing a means of exploring the features that were removed and possibly bring them back, the end-users can become unaware of some features that they might want to use in certain contexts. These negative effects could be overcome if the end-users are kept in the adaptation loop by supporting feedback on adaptations. Hence, the *Feedback Monitor* allows end-users to report their feedback on the UI adaptations presented by the system. End-users are given the ability to reverse adaptations or choose other possible alternatives.

Cedar Studio [3] is an integrated development environment (IDE), which helps developers and I.T. personnel in defining and managing artifacts such as UI models and adaptive behavior, which are stored in a server-side database. This IDE can access the server-side layers through web-services in order to request or update artifacts. Cedar Studio can be observed in operation through online demonstration videos [33].

CEDAR and RBUIS were only evaluated in our previous work by constructing new UI prototypes. In this paper, we contribute a method for integrating RBUIS in existing enterprise applications following the CEDAR architecture. The OFBiz system is used as a test-case for evaluating if the proposed integration method works without incurring a high development cost or significantly disrupting the way the enterprise application functions.

3.2 The RBUIS UI Adaptation Mechanism

CEDAR is a generic architecture that can form the basis for a variety of UI adaptation mechanisms such as RBUIS [2]. RBUIS was created in the spirit of RBAC [18] and was evaluated in terms of usability enhancement. In RBUIS, roles are applied to task models (represented as ConcurTaskTrees [30]) for adapting the UI's feature-set by removing features that are not required by certain end-users. Also, the layout can be optimized by adapting concrete widget properties such as: size, location, type, etc. Layout optimization is done by executing adaptation workflows that can embody visual and code-based constructs, on the concrete UI (CUI) models. To adapt a UI using RBUIS, a call is made to the server-

side layers with the identifiers of the end-user and the UI as parameters. The end-user-identifier is used to retrieve the roles, which are granted to the logged-in end-user. Then, the adaptive behavior associated with these roles is executed on the UI models relevant to the UI identifier. Finally, the adapted UI is transmitted to the client-side as XML to be rendered on the screen. In this paper, we used RBUIS to give OFBiz adaptive UI capabilities.

3.3 Adaptive UI Integration Technique

OFBiz uses HTML to represent its UIs. Hence, in order to integrate RBUIS in it, we developed a JavaScript version of CEDAR's client API that works with HTML UIs. Since RBUIS adopts a model-driven UI development approach, we devised a procedure for reverse engineering HTML forms into a model-driven representation supporting the levels of abstraction suggested by CAMELEON (Task, AUI, and CUI models). The reverse engineering is done at design-time. However, our technique launches the HTML pages of OFBiz in the browser then acquires the HTML through JavaScript to include the elements that are generated by server-side scripts. Our procedure transforms an HTML form into an XML document, which is used to create a CUI model. Then, the CUI is reverse engineered into an AUI model and the AUI into a task model automatically. The only manual part in this procedure is the definition of mapping rules. An excerpt of the code for reverse engineering an HTML table is shown in Listing 1.

Listing 1. Code for Reverse Engineering HTML UI to a Model-Driven Representation: Excerpt of HTML Table Example

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

After reverse engineering the UIs that require adaptation, we can apply RBUIS on the obtained UI models using Cedar Studio. To make the adaptation work at runtime on OFBiz's HTML pages, we need to extend OFBiz with a few lines-of-code that load the CEDAR API, call its web-service, and apply the obtained result. OFBiz uses a master page to wrap its UI forms with a common header, footer, and panel as shown in Figure 2. To reduce the integration effort we loaded the API and performed the adaptation call in the common header using the code shown in Listing 2.

Listing 2. Code for Enabling Adaptive UI Capabilities

```
   //Load the API Scripts
1: <script type="text/javascript" src="http://
   [ServiceAddress]/CedarScripts.js"></script>
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 "getUserID()" function call on Line 5 in Listing 2 should be implemented by the developer to obtain the identifier of the logged-in user from the OFBiz system. The "LoadAdaptedUI" function can internally acquire the UI identifier through a mapping table that contains the UI's URL and a number to identify the UI's models in the CEDAR database. The UI's URL is obtained from the web-browser and passed as a parameter to the adaptation function on CEDAR's web-service. The mapping is done on the server-side by querying a mapping table in the CEDAR database. After receiving an XML representation of the adapted UI from the server, the UI renderer component will apply the changes to the HTML page loaded on the client by modifying the widgets' properties. An excerpt of the code that applies the adaptations is shown in Listing 3. This code excerpt demonstrates hiding the widgets that were set to be invisible by an adaptation (e.g., removing features that are not required by a certain user).

Listing 3. API Code for Applying the Adapted User Interface: Excerpt of Widget Hiding Example

```
1: function ApplyAdaptedUI(UIXML){
   //Loop around the UI widgets
2: $(UIXML).find("Control").each(function () {
     //Get the name and visibility attributes
3:   var technicalName=$(this).attr('TechnicalName');
4:   var isVisible = $(this).attr('Visible');
     //Hide the invisible elements
5:   if(isVisible == 'false'){
6:     var element = GetElement(technicalName);
       //Hide the element if it exists
7:     if (typeof (element)!= 'undefined')
8:       {element.style.visibility = 'collapse';}}
9: }); }
```

3.4 User Feedback Mechanism

The *Feedback Monitor* presented in Section 3.1 allows users to change simplification operations by bringing back features in the case of feature-set minimizations or choosing alternatives in the case of layout optimizations as shown in Figure 3-A. Based on a recommendation we obtained by interviewing an industry expert, we extended this mechanism's functionality to allow users to add fields that did not previously exist in the enterprise application as illustrated by Figure 3-B. Changing simplification operations is enabled for the adapted UIs whereas adding new fields is enabled for all the reverse engineered UIs. Users can access the feedback mechanism by clicking a chameleon icon that appears in the corner of the UI. Upon changing the simplification operations, a request is made to the server passing the changes as a parameter and the UI is readapted accordingly. As for adding new fields, the UI models are extended and the UI is reloaded to show the addition.

(A) Changing Simplification Operations

(B) Adding New Fields

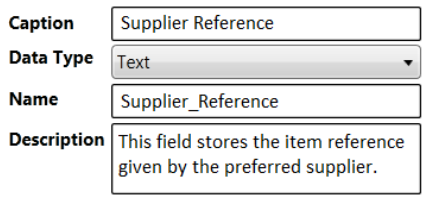

Figure 3. User Feedback Mechanism

4. METRIC-BASED EVALUATION

The process of integrating UI adaptation capabilities in enterprise applications starts by reverse engineering the target application's UIs. Afterwards, the application is extended to support adaptation hence becoming able to adapt its UIs at runtime. This section explains the metrics that we used to evaluate our integration method at all the stages of the process and demonstrates an application of these metrics to scenarios from OFBiz.

4.1 Reverse Engineering the User Interfaces

As we mentioned in Section 3.3, we devised a procedure for reverse engineering HTML forms into a model-driven representation that can be adapted by RBUIS. Although it is automated, this procedure requires mapping rules to be defined manually. Hence, the first question that might come to mind is about the difficulty of deducing these rules from the existing enterprise system since it has a large number of UIs. Assuming that there is no prior knowledge of the types of mapping rules required for reverse engineering the enterprise system at hand, we defined the following metrics for estimating the number of UIs that require manual work before the majority of the mapping rules are detected. These metrics indirectly show the level of diversity in an application's UIs. More diversity could signify that there are more mapping rules, which are more uniformly distributed over the entire system.

The **approximate mapping rule detection saturation point** *SP* indicates that the number of new encountered mapping rules stabilized after reverse engineering a number of UIs *a*. This metric will allow us to test if the Pareto principle (70-30 rule) applies for detecting 70% of the mapping rules in the first 30% of the UIs. If this principle applies, it indicates that less manual work is required for reverse engineering since the UIs have similar characteristics. To check if the **Pareto principle** holds, we define the following equation where $\{R\}$ is the set of rules detected in the UIs before *SP* and $\{MR\}$ is the set of all the detected mapping rules:

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

The saturation point *SP* is defined as follows:

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

where *UI* is a user interface being reverse engineered, *C* is the number of new mapping rules detected in this UI, the subscript *b* of *C* indicates the next UI to be reverse engineered, and *T* is the total number of UIs to be reverse engineered. The types of mapping rules that are encountered when reverse engineering a UI can differ depending on the characteristics of the software application being reverse engineered. We hypothesize that the Pareto principle holds for enterprise applications due to the use of similar WIMP style UIs.

OFBiz Scenario: We selected a sample formed of the 19 main input UIs from the "Catalog" and "Human Resources" modules. We were able to deduce two types of mapping rules necessary for reverse engineering these UIs into a model-driven representation: (1) The most common type of rule is the one that maps individual HTML elements to CUI elements that are in turn mapped to AUI elements then tasks in the task model, and (2) the second type of rule is related to grouping widget pairs composed of a label and an input widget into logical groups that are reflected in the AUI and task models. Defining rules from these two types alongside getting information provided by the HTML UI (e.g., widget properties such as name, size, location, etc.) was sufficient to obtain a model-driven user interface representation that we can adapt using our RBUIS mechanism.

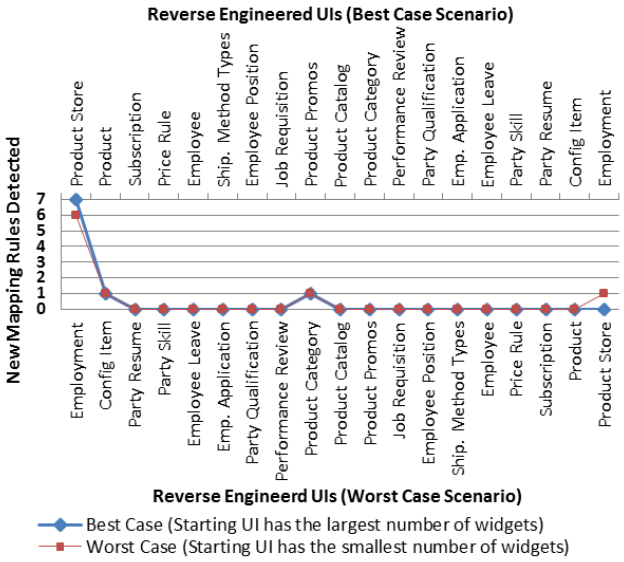

Figure 4. Saturation Point for Mapping Rules

We encountered 8 different widget types, each requiring 1 mapping rule, and were able to detect the second mapping rule relating to logical widget grouping in the first UI. We obtained a saturation point *SP* = 2 / 19 = 0.10 signifying that after the second UI the mapping rules become minimal as shown in Figure 4. Following our example where *SP* = 0.1, *P* is: 7 / 9 = 0.77 (77%) in best case scenario and 6 / 9 = 0.66 (66%) in the worst case one. With an average of 71.5 % of the rules detected in the first 10% of the UIs, we can say that the Pareto principle holds and the UIs of OFBiz are highly similar.

4.2 Integrating the Adaptive UI Capabilities

After reverse engineering the UIs, we can assess the level of change the integration will incur on the enterprise application. We defined the *lines-of-code* and *change-impact* metrics for this assessment.

The **lines-of-code** metric refers to the code required locally in each UI or globally in the enterprise application to apply a type of adaptation. This metric excludes the API code since CEDAR requires each presentation technology (e.g., HTML) to have one API that is reusable with any enterprise application. The lines-of-code metric is given as follows:

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

where *LLOC* represents a UI's local lines-of-code, whereas *GLOC* represents the global lines-of-code common across the application, *A* is the required adaptation, *UI* is the user interface to which this adaptation will be applied, and *EA* is the enterprise application. The values for LLOC and GLOC represent the number of lines-of-code that must to be added to make the adaptation operational.

OFBiz Scenario: As an example test-case, we considered the context-driven UI adaptations listed in Table 2 and applied them to OFBiz. An example of the output was shown earlier in Figure 1. Adaptation A1 is a feature-set minimization, whereas adaptations A2, A3, and A4 are examples of layout optimizations.

Table 2. Example User Interface Adaptations

Code	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., larger fonts for visually impaired users)

Our method only requires the 5 lines-of-code shown in Listing 2 to be added globally to OFBiz's common header to empower it with adaptive UI capabilities. Consider {AE} to be the set of adaptations listed in Table 2. The lines-of-code needed to make these adaptations work in OFBiz using our method are $\forall x \mid x \in \{AE\}$, *GLOC* (x, OFBiz) = 5 and *LLOC* (x, AnyUI) = 0. Achieving this low number of lines-of-code is possible because all the adaptation rules are defined on the server-side as shown in Figure 2.

Some approaches discussed in Section 2 operate by changing the UI's representation (e.g., HTML tags) at design-time. Therefore, we established the **change-impact** (CI) metric to measure the level of change each approach will incur on the enterprise application. A higher change-impact could signify that: (1) More time and effort could be needed to perform the integration and (2) the compilation time could increase if a compiled presentation technology such as Windows Forms was used. Since we can think of UIs in terms of widgets, the change-impact metric is given as follows:

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

where A is the adaptation being applied, UI is the user interface being adapted, 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), and $|\{W\}_k|$ is the number of widgets of a certain type that have been changed by the adaptation.

The variable v represents the number of generated UI versions and is > 1 for approaches that cannot adapt the same UI copy (e.g., a single HTML page) but generate multiple copies of the UI each of which is adapted to a certain context-of-use. Widget toolkits aim at replacing existing widgets from the standard toolkit with adaptive equivalents. Hence, the value of v for widget toolkits would be = 1 since the change is occurring in the initial UI copy. We should note that widget toolkits are generally used to adapt the layout and do not have the ability to adapt the feature-set due to their lack of a high-level UI model such as the task model. Model-driven design-time generative approaches generate multiple versions of the same UI adapted to different contexts-of-use. Hence, the value for v in this approach would be > 1. The research work that used AOP for adapting the UI's behavior [9] (Section 2.2) relied on manually creating multiple adapted UI layouts hence we also consider its v value to be > 1. As for our method, *CI* is always = 0 since we use runtime adaptation hence the UI representation (e.g., HTML pages) will remain completely intact at design-time.

Table 3. Integration Time of Different Adaptation Approaches

Approach	Integration Time
Widget Toolkits	Average / High
Model-Driven Generative D.T.	Average
AOP + D.T. Manual Adaptation	High
Model-Driven Interpreted R.T.	Low

Based on *CI* we provided a conceptual comparison between the different UI adaptation approaches as shown in Table 3. Our aim is to give an idea about the differences in the required integration effort between approaches, while recognizing that there could be slight differences between adaptation techniques using the same approach. Widget toolkits require an average amount of time if a conversion tool existed to automatically convert the UI otherwise a high amount of time is needed. Model-driven generative design-time approaches require an average amount of time since the adapted versions could be automatically generated but more time could be still required to integrate them with the software application. Logically, manual adaptation requires a high amount of time. The integration time of our method is low since *CI* is always = 0, hence the developers can continue working on the application without major disruptions.

OFBiz Scenario: We attempted to apply adaptation A2 (Table 2) to the 19 main input UIs of the Catalog and Human Resources modules of OFBiz. This adaptation switches combo boxes with three other types of widgets including: radios buttons, list boxes, and lookups. These possibilities indicate that we could obtain three different versions of the UI hence v (Equation 4) = 3 for the model-driven generative and manual design-time approaches and v = 1 for the widget toolkit approach. The value for n (Equation 4) is 1 since we are only adapting combo boxes, and we consider that each combo box is represented by a single HTML tag hence l (Equation 4) = 1. The results we obtained from calculating *CI* are listed in Table 4, and show that the CEDAR approach has the lowest change-impact.

Table 4. CI Example Based on 19 UIs from OFBiz

	Change-Impact	
Approach	**Mean**	**Total**
Widget Toolkits	6.94	132
Model-Driven Generative D.T.	106.73	2028
AOP + D.T. Manual Adaptation	106.73	2028
Model-Driven Interpreted R.T.	0	0

* The applied adaptation switches combo boxes with radio buttons, list boxes, and lookups

4.3 Level of Decoupling

The level of decoupling shows how much intertwining exists between the adaptive behavior and the enterprise application. It is affected by the percentage of adaptive behavior defined in the enterprise application versus that defined separately. Decoupling provides a separation of concerns that could offer potential for scalability and facilitate the integration of an adaptation technique in existing enterprise applications. As shown earlier in Figure 2, CEDAR provides complete separation between the implementation of the adaptive UI technique (e.g., RBUIS), which resides on a server and the enterprise application that uses a client-side API to communicate with it through a web-service.

It is important to maintain the **backward compatibility** of UI adaptations as enterprise applications evolve. We consider an adaptation A to be backward compatible if it can be applied to previous UI versions *successfully* and *without reintegration effort*. Decoupling helps in improving backward compatibility in terms of eliminating *reintegration effort*. A conceptual assessment of the backward compatibility of UI adaptation approaches is presented in Table 5 based on the need for *reintegration effort*.

Table 5. Backward Compatibility of UI Adaptation Approaches

Approach	Backward Compatible
Widget Toolkits	Depends on the ability to load a new widget toolkit version at runtime
Model-Driven Generative D.T.	False
AOP + D.T. Manual Adaptation	False
Model-Driven Interpreted R.T.	True

Widget toolkits can be backward compatible if it is possible to load a new toolkit version at runtime to update the existing adaptive behavior in older versions of the enterprise system. This is not possible with model-driven approaches that generate UIs at design-time since the generated artifacts have to be manually integrated in all the previous enterprise application versions. Manual design-time adaptation suffers from a similar problem. If we consider the adaptations listed in Table 1, we can say that our approach is backward compatible since it is only necessary to define a global code once to make these adaptations work for all the UIs. Hence, the adaptations would work for all the previous versions that have this code since the adaptive behavior are being defined separately.

An adaptation's *success* can be partial due to differences in the UI definition between one version and another. We defined a metric for calculating the backward compatibility success ratio as follows:

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

where UI_{vn} is a UI from the enterprise application version into which the adaptation A was integrated for the first time, and UI_{vn-k} is one of the previous versions; $\{W\}$ is the set of widgets in a UI and $\{AW\}$ is the set of widgets affected by an adaptation A.

As an example of partial UI adaptation success, let us consider a UI for managing customer records. Consider that CustomerUI_{v2} has multiple fields, 10 of which are for data selection and are represented as combo boxes (e.g., gender). Assume that the previous UI version CustomerUI_{v1} has the same data selection fields but only 8 are represented as combo boxes and the other 2 are list boxes. If we introduce an adaptation to switch data selection widgets with radio buttons in CustomerUI_{v2}, we might ignore list boxes. In this case, $BC = 8/10 = 0.8$ indicating an 80% success rate. With approaches that are not dynamic and rule-based (e.g., design-time generative), two adapted UIs have to be generated and integrated into each respective CustomerUI version to achieve a 100% success rate. As for our approach, we only have to adjust the adaptation rule in our RBUIS mechanism to take into consideration list boxes as well as combo boxes to obtain a 100% backward compatibility.

4.4 Runtime Performance

Considering that our approach is highly dynamic we had to test its runtime *efficiency* and *scalability* especially since we are working with UIs that are expected to load in real-time. In a previous work [2], we conducted a complexity analysis to show that the algorithms behind our RBUIS mechanism are theoretically scalable. In this paper, we tested our technique's runtime efficiency and scalability after integrating it with an existing real-life system (OFBiz). To perform this test we defined the following **efficiency** metric as a function of an adaptation A and a user interface UI:

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

where t_{0a} is the time required to perform an adaptation on the server-side, t_{0b} is the common server-side time required for any number of adaptations (e.g., loading common data before applying the adaptations), t_1 is the time needed to transmit the adapted UI as XML back to the client, and t_2 is the time it takes the API to apply the adaptation on a running UI such as an HTML page in OFBiz.

We used this metric to test the efficiency of the four example adaptations listed in Table 2 on the three UIs with the highest number of widgets in OFBiz's Catalog module. The test was conducted on a single machine with an Intel Core 2 Duo 2.93GHz CPU and 4 GB of RAM running a 32 bit edition of Windows 7. We used the Firefox web-browser to run OFBiz.

We determined the t_{0b} variable to be equal to 30 milliseconds (ms). The t_1 variable depends on the network connection and is negligible for our test since we were operating on a single machine. We calculated the average XML document size for the 3 selected UIs to be 20kb. Based on this file size, t_1 will be very small over an internet connection (e.g., ≈15ms / 10Mbps) and negligible over a corporate network (e.g., ≈0.15ms / 1Gbps). The values of variables t_{0a} and t_2 are shown in Figure 5 for each UI and adaptation.

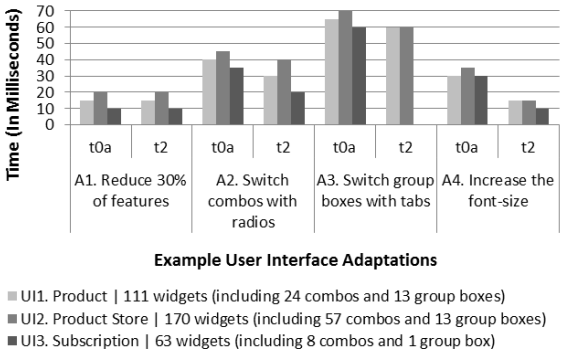

Figure 5. Results of the Efficiency Test on 3 OFBiz UIs Using 4 Example Adaptations (t_{0b} = 30ms and t_1 =15ms)

Using the data shown in Figure 5 and considering t_1 to be 15ms we determined the average efficiency for each adaptation to be: $E(A1)$=75ms, $E(A2)$=115ms, $E(A3)$=150ms, and $E(A4)$=90ms. The general average is (75 + 115 + 150 + 90) / 4 = 107.5ms. If we do not consider the fixed values t_{0b} (30ms) and t_1 (15ms), the general average will be 62.5ms. Based on this number, we can say that our technique can perform around 15 different adaptations on the same UI, transmit it, and display the result all in less than 1 second (62.5 × 15 + 30 + 15 = 982.5ms).

Since the CEDAR architecture supports client-side and server-side caching, performance can be further enhanced. Client-side caching is used if a user that is still operating in the same context (e.g., still logged in with the same roles) requests a UI that has already been adapted. In this case the efficiency metric will be: $E(A, UI) = t_2$ (general average 24.5ms). As for the server-side caching, it is used when a user requests a UI that has already been adapted for another user operating in the same context (e.g., a user that has the same roles). In this case, the efficiency metric will be: $E(A, UI) = t_1 + t_2$.

After testing the efficiency of our technique we verified its **scalability** by load-testing CEDAR's UI adaptation web-service. We selected the largest of the three UIs that were used in the scalability test (Product Store UI with 170 widgets) and applied to it the four adaptation operations shown in Table 2. We submitted increasing requests of that UI to the server over five minute periods and repeated the whole cycle five times. The web-service was hosted on an Amazon cloud server with a single Intel Xeon CPU with 2 cores (2.40 GHz, 2.15GHz), 3.75 GB of RAM, and running a 64-bit edition of Windows Server 2012 Standard with the IIS 7 web-server. We consider this setup to be an average configuration since enterprises with hundreds of users usually setup servers with multiple CPUs and a larger amount of RAM. We simulated the load using an application that we developed and ran simultaneously on three client machines. The resulting server response times ($t_{0a} + t_{0b}$ from Equation 6) are shown as a box plot in Figure 6.

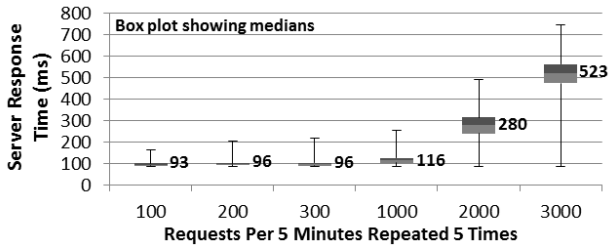

Figure 6. Box plot of Load-Testing Results (showing medians)

The fitting curve of the mean response times shown in Figure 7 is polynomial of the 4th order with R^2=0.9999431. We should note that the polynomial curves of the 2nd and 3rd orders also produced a high R^2 where R^2 (2nd) = 0.9977252 and R^2 (3rd) = 0.9989506. Based on this test, we can say that our UI adaptation service is scalable and will not form a bottle-neck if it receives a high number of requests.

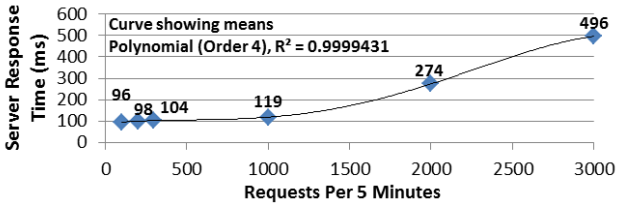

Figure 7. Curve of the Load-Testing Results (showing means)

5. GENERALITY AND FLEXIBILITY OF OUR METHOD: INDUSTRIAL EXPERTISE AND DATA

This section presents an evaluation of the generality and flexibility of our method based on industrial expertise and data.

To evaluate our method from an industrial perspective, we drew on the expertise and data from real-life projects offered to us by a software company that sells enterprise systems to medium and large enterprises in China. We selected this company due to its expertise in enterprise systems, UI adaptation, and our test-case OFBiz.

We initially visited the company to get information on their work and the problems that they face with enterprise applications. In this initial visit, we discovered that one of the major problems they face is usability related. The enterprise applications that they sell suffer from a diminished user experience due to the diverse end-user needs that make one UI not fit for all users. We established through a verbal explanation of our UI adaptation technique that it could be useful with real-life enterprise systems such as OFBiz. At a later stage, since we were able to integrate our UI adaptation technique successfully in OFBiz, we sought to further evaluate its usefulness by assessing its *generality* and *flexibility*. These two criteria were introduced (alongside others) by Olsen [29] for evaluating UI research including architectures such as CEDAR. According to Olsen, *Generality* evaluates the possibility of using the proposed solutions with different use cases and *flexibility* evaluates "*the possibility of making rapid design changes that can be evaluated by the users*" (*p.255*). We demonstrated our UI adaptation and integration techniques to the manager with videos [33] of running examples on using our IDE Cedar Studio for developing adaptive model-driven UIs and an example on integrating these capabilities in OFBiz. Afterwards, we conducted a semi-structured interview over the phone with the manager and followed it with several discussions.

To achieve **generality**, our method only requires an API for the presentation technology adopted by the target enterprise application. As shown by the CEDAR architecture in Figure 2, all the server-side components are technology independent and can be accessed from a technology dependent API through web-services. An API for a particular presentation technology can be used with any application adopting this technology by following the integration procedure described in Section 3. This is deemed acceptable by the manager especially since we developed an API and demonstrated it in a working example alongside our IDE, Cedar Studio.

According to Olsen's definition [29], **flexibility** is regarded as a development metric that assesses how easy it is for developers to make rapid design-time changes using a tool. It is achieved from this perspective by our IDE Cedar Studio, which supports visual-design tools for both UI models and adaptive behavior in addition to integrated testing of the adapted UIs. These features allow changes and testing to be done rapidly. Nevertheless, during our interview we deduced a helpful end-user perspective of flexibility. It covers the possibility and ease through which end-users can change the UI themselves without referring to software developers.

Based on his company's experience, the manager said that UIs are initially adapted by the developers based on initial knowledge acquired on the needs of an enterprise's end-users. Afterwards, the UI adaptation is tuned over several cycles in a process that includes user evaluation, change reporting and discussion, and readapting the UI based on the newly reported changes. He noted that the adaptation mechanism available to them in OFBiz supports reducing features (layout optimization is not supported) through XML configuration files, which are defined by the developers. Therefore, as he stated, the feedback mechanism provided by our approach is an important advantage that empowers end-users to provide direct feedback to the system in order to shorten the cycles of the adaptation process. This reduces the implementation cost and allows the users to obtain an adapted UI more quickly. As a result of this interview, we were able to establish the process shown in Figure 8, which demonstrates conceptually these advantages.

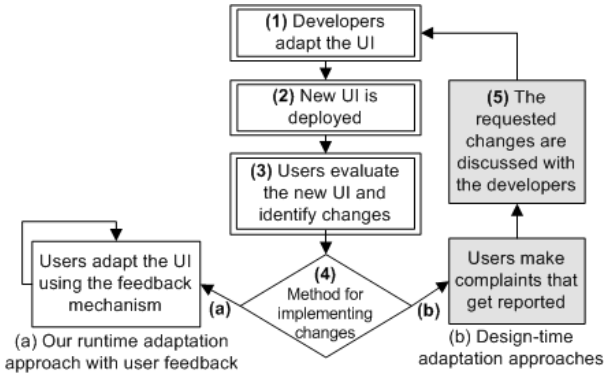

Figure 8. UI Adaptation Process: Design-Time versus Runtime UI Adaptation Cycles (based on interviewing industry experts)

A complementary indication on the importance of runtime adaptation approaches is made by an existing research work, which states that software systems should attempt to break the boundary between development-time and runtime to handle the changes that cannot be anticipated or predicted beforehand [6]. Empowering users with control over the UI adaptations narrows this boundary and helps in reducing the round trip in the adaptation process.

In a previous work [2], we conducted a usability study with 25 participants that demonstrated the ease of use of our feedback mechanism since 80% of the participants were able to use it by only referring to a few written words of instruction on its purpose.

In this paper, we estimated the time that the feedback mechanism could save in the UI adaptation cycle based on real-life data. We asked the manager who we interviewed to provide us with timestamps of requests on the different steps of the UI adaptation process from past projects. We were provided with a sample of 36 timestamps of requests from 3 past projects that were running in parallel. The timestamps were obtained by referring to historic emails of requests on development, deployment, and change reporting and discussion. Based on these timestamps, we calculated the mean number of days for developing and deploying the adapted UIs and reporting and discussing change requests between the enterprise employees and the software company. The results are shown in Figure 9 but the project names are hidden for confidentially purposes. The results indicate that the highest mean days in the UI adaptation process are allocated to user evaluation, and change reporting and discussion (Project A=45.25, Project B=25.66, Project C=35) and a smaller mean number of days is allocated to the development and deployment of UI adaptations (Project A=9, Project B=4.75, Project C=5.25).

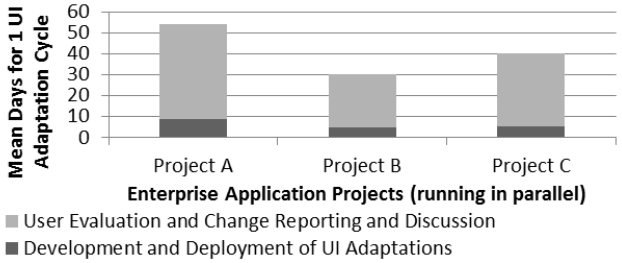

Figure 9. Mean Number of Days for 1 UI Adaptation Cycle from Real-Life Enterprise Projects Running in Parallel

The results in Figure 9 show that if the UI adaptation process was repeated from the start with every cycle, a period of over 1 month could pass before the users get their requested UI adaptations. On the other hand, if the users were given the ability to report the changes directly to the system through a feedback mechanism this process could become much shorter by eliminating the time required for development, deployment, and change discussion.

6. THREATS TO VALIDITY AND LIMITATIONS

The data presented in this paper is based on applying our UI adaptation approach to scenarios from OFBiz. The figures we obtained by applying the saturation point (SP) metric give us an indication about the nature of enterprise application UIs without claiming generalizability to all enterprise 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 Figure 7 is intended to show that our UI adaptation mechanism is scalable. Determining an accurate regression equation, which is not the purpose of this test, requires a larger sample of mean execution times. Interviewing more industry experts could support our generality claim further. Concerning the UI adaptation cycle data (Figure 9), as we mentioned earlier, it is based on a sample of 36 request timestamps from 3 projects. Therefore, our intention is not to generalize it but to give an indication about the time each adaption cycle could take to show the usefulness of our runtime feedback mechanism in shortening these cycles.

Task models represented as ConcurTaskTrees support temporal operators, which can help in determining inter-task dependency. Determining this dependency is helpful for feature-reduction adaptation operations. Currently, we are unable to automatically detect these operators when reverse engineering a UI specified in a presentation technology such as HTML to a model-driven representation. It is possible to specify these operators manually using the task model design tool in our IDE Cedar Studio. Another limitation lies in the addition of new fields using the feedback mechanism. This functionality allows the new fields to be rendered on the screen by updating the UI models. However, for the fields' values to be stored in the enterprise's database, the enterprise application should support domain model extension. OFBiz allows its domain model to be extended by the developers but the feedback mechanism makes it extensible by the end-users. In case other enterprise applications did not support domain model extension, this functionality has to be programmed before the end-users can use the field addition part of our feedback mechanism.

7. CONCLUSIONS AND FUTURE WORK

Adaptive UIs can help enterprise applications to overcome some of their usability problems [2]. Many of these systems have a large number of UIs and are at a mature stage in their development life-cycle. However, existing works on adaptive UIs mostly test their approaches by building new prototype systems but do not present and evaluate methods that can integrate such capabilities in existing systems without causing major changes to the way they function or incurring a high integration cost.

In this paper, we presented a method for integrating adaptive UIs in enterprise applications based on our CEDAR architecture. This method uses interpreted runtime models to empower enterprise applications with adaptive UI capabilities without the need for a major integration effort. We established several technical metrics and applied them to evaluate our method based on scenarios from the open source enterprise application OFBiz. This assessment covered the different phases of our method including reverse engineering, integration, and runtime execution. We showed that due to the similarity between enterprise-application UIs, around 70% of the mapping rules required for the reverse engineering phase could be determined by examining the first 30% of the UIs to be reverse engineered. After determining the mapping rules, the reverse engineering process becomes fully-automated. Without changing the underlying functionality, our integration method only requires a few lines-of-code to work, and does not have a high change-impact on existing UI definitions in comparison to other approaches. Furthermore, we demonstrated that our runtime UI adaptation mechanism is both efficient and scalable by applying it to real-life scenarios from OFBiz. Finally, we showed the generality and flexibility of our method based on an interview and discussions with practitioners and data from their real-life projects.

In the future, we aim to devise a technique that can automatically detect the temporal operators for the task models when reverse engineering a final user interface (e.g., HTML) into a model-driven representation. An interesting starting point could be an existing work that has explored a way to transform HTML pages into state-machine diagrams by relying on the function calls in the code behind the UI [26]. Additionally, we are aiming to ask software developers to evaluate our UI adaptation and integration approach and our supporting tool (Cedar Studio) in a focus group setting.

8. ACKNOWLEDGMENTS

We would like to thank Shanghai Banff Technology Ltd for the interviews, feedback, and data they provided us from their projects and Prof. Marian Petre for her input on this paper. This work is funded by The Open University and ERC Advanced Grant 291652.

9. REFERENCES

[1] Akiki, P.A., Bandara, A.K., and Yu, Y. Using Interpreted Runtime Models for Devising Adaptive User Interfaces of Enterprise Applications. *Proceedings of the 14th International Conference on Enterprise Information Systems*, SciTePress (2012), 72–77.

[2] Akiki, P.A., Bandara, A.K., and Yu, Y. RBUIS: Simplifying Enterprise Application User Interfaces through Engineering Role-Based Adaptive Behavior. *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ACM (2013), 3–12.

[3] Akiki, P.A., Bandara, A.K., and Yu, Y. Cedar Studio: An IDE Supporting Adaptive Model-Driven User Interfaces for Enterprise Applications. *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ACM (2013), 139–144.

[4] Appert, C. and Beaudouin-Lafon, M. SwingStates: Adding State Machines to the Swing Toolkit. *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, ACM (2006), 319–322.

[5] Balme, L., Demeure, R., Barralon, N., Coutaz, J., Calvary, G., and Fourier, U.J. Cameleon-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. *Proceedings of the 2nd European Symposium on Ambient Intelligence*, Springer (2004), 291–302.

[6] Baresi, L. and Ghezzi, C. The Disappearing Boundary Between Development-time and Run-time. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ACM (2010), 17–22.

[7] Bencomo, N., Sawyer, P., Blair, G.S., and Grace, P. Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. *Proceedings of the 12th International Conference on Software Product Lines*, Lero Int. Science Centre, University of Limerick (2008), 23–32.

[8] Blouin, A. and Beaudoux, O. Improving Modularity and Usability of Interactive Systems with Malai. *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ACM (2010), 115–124.

[9] Blouin, A., Morin, B., Beaudoux, O., Nain, G., Albers, P., and Jézéquel, J.-M. Combining Aspect-Oriented Modeling with Property-Based Reasoning to Improve User Interface Adaptation. *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ACM (2011), 85–94.

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

[11] Blumendorf, M., Lehmann, G., Feuerstack, S., and Albayrak, S. Executable Models for Human-Computer Interaction. *Interactive Systems. Design, Specification, and Verification*, Springer-Verlag (2008), 238–251.

[12] Botterweck, G. Multi Front-End Engineering. In H. Hussmann, G. Meixner and D. Zuehlke, eds., *Model-Driven Development of Advanced User Interfaces*. Springer (2011), 27–42.

[13] Calvary, G., Coutaz, J., Dâassi, O., Balme, L., and Demeure, A. Towards a New Generation of Widgets for Supporting Software Plasticity: The "Comet." In R. Bastide, P. Palanque and J. Roth, eds., *Engineering Human Computer Interaction and Interactive Systems*. Springer (2005), 306–324.

[14] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers 15*, Elsevier (2003), 289–308.

[15] Carroll, J.M. and Carrithers, C. Training Wheels in a User Interface. *Communications of the ACM 27*, 8, ACM (1984), 800–806.

[16] Cheng, B.H.C., Lemos, R., Giese, H., et al. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In B.H.C. Cheng, R. Lemos, H. Giese, P. Inverardi and J. Magee, eds., *Software Engineering for Self-Adaptive Systems*. Springer (2009), 1–26.

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

[18] Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., and Chandramouli, R. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security 4*, 3 ACM (2001), 224–274.

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

[20] Gajos, K.Z., Weld, D.S., and Wobbrock, J.O. Automatically Generating Personalized User Interfaces with Supple. *Artificial Intelligence 174*, 12-13, Elsevier (2010), 910–950.

[21] Johnson, J. Selectors: Going Beyond User-Interface Widgets. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (1992), 273–279.

[22] Kiczales, G., Lamping, J., Mendhekar, A., et al. Aspect-Oriented Programming. *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Springer (1997), 220–242.

[23] Kramer, J. and Magee, J. Self-Managed Systems: an Architectural Challenge. *Proceedings of the Workshop on the Future of Software Engineering*, IEEE (2007), 259–268.

[24] Lecolinet, E. A Molecular Architecture for Creating Advanced GUIs. *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, ACM (2003), 135–144.

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

[26] Maezawa, Y., Washizaki, H., Tanabe, Y., and Honiden, S. Automated Verification of Pattern-based Interaction Invariants in Ajax Applications. *Proceedings of 28th IEEE/ACM International Conference on Automated Software Engineering*, ACM (2013), 158–168.

[27] McGrenere, J., Baecker, R.M., and Booth, K.S. An Evaluation of a Multiple Interface Design Solution for Bloated Software. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2002), 164–170.

[28] Nichols, J., Myers, B.A., and Litwack, K. Improving Automatic Interface Generation with Smart Templates. *Proceedings of the 9th International Conference on Intelligent User Interfaces*, ACM (2004), 286–288.

[29] Olsen,Jr., D.R. Evaluating User Interface Systems Research. *Proceedings of the 20th ACM SIGCHI Symposium on User Interface Software and Technology*, ACM (2007), 251–258.

[30] Paternò, F., Mancini, C., and Meniconi, S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. *Proceedings of the 6th International Conference on Human-Computer Interaction*, Chapman and Hall (1997), 362–369.

[31] Pleuss, A., Botterweck, G., and Dhungana, D. Integrating Automated Product Derivation and Individual User Interface Design. *Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems*, Universitat Duisburg-Essen (2010), 69–76.

[32] Shneiderman, B. Promoting Universal Usability with Multi-Layer Interface Design. *Proceedings of the Conference on Universal Usability*, ACM (2003), 1–8.

[33] Our Demonstration Videos. http://adaptiveui.pierreakiki.com.

[34] Apache Open For Business (OFBiz). http://ofbiz.apache.org/.

[35] How does SAP turn 250 million lines of code into modular services? http://bit.ly/SAPLinesOfCode.

[36] Lawson Smart Office brings WPF Goodness to the Enterprise. http://www.youtube.com/watch?v=TnagncUOlVE.