Cedar Studio: An IDE Supporting Adaptive Model-Driven User Interfaces for 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

ABSTRACT
Support tools are necessary for the adoption of model-driven engineering of adaptive user interfaces (UI). Enterprise applications in particular, require a tool that could be used by developers as well as I.T. personnel during all the development and post-development phases. An IDE that supports adaptive model-driven enterprise UIs could further promote the adoption of this approach. This paper describes *Cedar Studio*, our IDE for building adaptive model-driven UIs based on the CEDAR reference architecture for adaptive UIs. This IDE provides visual design and code editing tools for UI models and adaptive behavior. It is evaluated conceptually using a set of criteria from the literature and applied practically by devising example adaptive enterprise user interfaces.

Author Keywords
IDE; Model-driven engineering; Adaptive user interfaces; Enterprise applications; User interface simplification

ACM Classification Keywords
[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

INTRODUCTION
The model-driven approach to UI development can serve as a basis for devising adaptive UIs for enterprise applications due to the possibility of applying different types of adaptations on the various levels of abstraction [2].

Yet, practically implementing adaptive model-driven UIs requires tools that support the creation of the necessary UI models and adaptive behavior. Existing tools lack many features required for supporting adaptive model-driven enterprise user interfaces. From a model-driven engineering perspective, such tools should be able to support the modeling, generation, and synchronization of all the levels of abstraction. Also, these tools should provide the ability to devise the adaptive behavior both visually and through code to support developers and I.T. personnel. Furthermore, an IDE style UI could provide the necessary ease-of-use for managing the complex user interface and adaptive behavior artifacts of large-scale enterprise applications.

This paper provides an overview of *Cedar Studio*, our Integrated Development Environment (IDE) that supports the development of adaptive model-driven enterprise application user interfaces based on the CEDAR reference architecture, which promotes the use of interpreted runtime models instead of code generation [1]. CEDAR is based on the: CAMELEON reference framework [4], Three Layer Architecture [11] and Model-View-Controller paradigm [12]. The UI and adaptive behavior models created using *Cedar Studio* are stored in a relational database, which provides an easier means for managing these artifacts at runtime. CEDAR's implementation is offered as a service consumed by *Cedar Studio* and technology specific APIs, which allow more enterprise applications to integrate with our solution. APIs can be devised for any presentation technology (e.g., HTML, Swing, etc.) and used in combination with *Cedar Studio* for developing adaptive UIs. The adaptations currently supported by *Cedar Studio* are primarily focused on UI simplification, which we define as a mechanism for increasing usability through adaptive behavior by providing users with a minimal feature-set and an optimal layout based on the context-of-use (user, platform, and environment). These adaptations are part of our Role-Based UI Simplification (RBUIS) mechanism [2].

Cedar Studio provides developers and I.T. personnel with an ease of access to all the visual design and code editing tools in one place. Currently, it supports visual design tools for the following artifacts: (1) *Task Models*, (2) *Domain Models*, (3) *Abstract UI (AUI) Models*, (4) *Concrete UI (CUI) Models*, and (5) *Goal Models*. Also, it supports automatic generation and synchronization between various levels of abstraction (Task Model, AUI, and CUI) and offers the possibility of making manual changes at any level. Additionally, *Cedar Studio* supports a combination of visual design and code editing tools that are necessary for implementing adaptive UI behavior including: (1) *Visual Adaptive Behavior Workflows* and (2) *Dynamic Scripts* for optimizing a UI's layout, (3) *Visual Role Assignments* and (4) *Code-Based Rules* for minimizing a UI's feature-set to a

Authors' Version
EICS'13, London, United Kingdom, *Forthcoming*

particular context, and (5) *SQL-based Model Constraints* for verifying manually created models.

Cedar Studio is meant to be used during various phases of the software lifecycle (development, deployment, and post-deployment). The UI models are created at development time and the adaptive UI behavior could be added at deployment time according to the needs of each enterprise.

The remainder of this paper is structured as follows: The next section briefly describes the gaps in existing tools. Then, we present the features of *Cedar Studio* and the process of using it for devising adaptive model-driven enterprise application UIs. Afterwards, we assess *Cedar Studio* based on criteria from the literature [18]. Finally, we give the conclusions and state our future work.

RELATED WORK

This section provides a brief overview of existing software tools that target model-driven and adaptive user interfaces.

Some tools supporting the development of model-driven UIs such as UsiComp [10], Xplain [9], Damask [14], and Gummy [15] are early stage research prototypes that do not provide an IDE style UI that generally helps developers and I.T. personnel in managing a large number of artifacts (e.g., UI models, code files, etc.) for real-life enterprise applications. Other similar tools such as SketchiXML [5], IdealXML [17], GraphiXML [16] just target specific phases of the UI construction process. MASP [7] provides tool support for devising adaptive UI layouts for home systems but does not provide a canvas-style visual design tool for devising WIMP style concrete UIs. Some approaches such as Supple [8] partially implement model-driven engineering of user interfaces, which is reflected in the accompanying tools that do not support all the levels of abstraction. *Cedar Studio* was developed in the form of an IDE that is aimed at providing integrated features and full support for the model-driven approach to user interface development.

There are commercial tools for supporting model-driven UI construction. Leonardi [24] is a UI design tool owned by the W4 company. Since Leonardi is a rapid application development tool, it limits its UI representation to the CUI level of abstraction. Additionally, various frameworks and tools (e.g., OpenXava [25], Himalia [26], etc.) provide different model-driven approaches for constructing UIs. Yet, the tight coupling of these tools with programming languages (e.g., Java, .NET, etc.) discourages their adoption as a generic solution. The UIs created with *Cedar Studio* are technology independent and are interpreted by separate APIs that could target any presentation technology.

A survey [21] on model-driven engineering tools for developing UIs included: ACCELEO, AndroMDA, ADT, AToM3, DSL Tools, Kermeta, ModFact, Merlin, MDA Workbench, MOFLON, OptimalJ, QVT Partners, SmartQVT, and UMLX. The models generated by these tools are static hence only adaptable at design-time whereas *Cedar Studio* is intended to support both user interface and adaptive behavior models that can be interpreted at runtime.

The next section presents *Cedar Studio* and explains how it can be used for simplifying UIs using adaptive behavior.

CEDAR STUDIO FEATURES AND PROCESS

This section presents the features of *Cedar Studio*, and explains the process of using this tool to devise adaptive model-driven UIs. *Cedar Studio* allows the process to start at any level of abstraction but we only demonstrate it starting from the task model due to space limits.

Task Models

The task model design tool, illustrated in Figure 1, supports visual composition of task models using ConcurTaskTrees (CTT) [20]. The importance of this tool is that it provides designers with the ability to visually design task models and allocate roles to them through the dialog shown in Figure 2 while maintaining the ability to allocate roles through more general code-based rules using a code editor. This visual and code-based combination for applying RBUIS in enterprise scenarios could enhance the *expressive match* denoting the closeness between the means for applying design choices and the problem at hand [19].

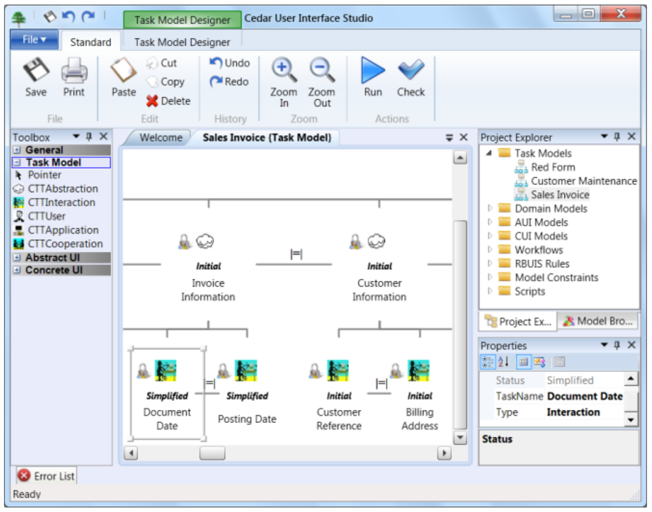
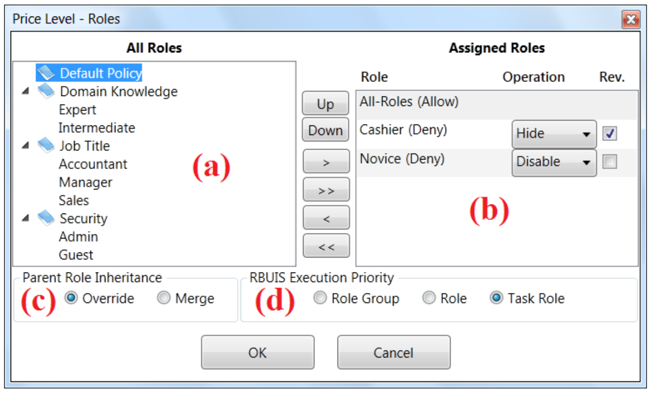


Figure 1. Task Model Design Tool

This tool supports a tree layout algorithm that can automatically adjust the presentation of large task models. Visual and code-based support is provided for the simplification process through role allocation to tasks. The lock-shaped button on each task allows a visual allocation of access rights using the UI shown in Figure 2. A default policy ("*All-Roles*") is implicitly assigned to grant access to all the roles on any given task. This policy could be overridden by explicitly assigning roles from different groups (Figure 2 - a) to each task. The concrete operation (e.g. hide, disable, etc.) and the ability to reverse it by the user are specified for each role (Figure 2 - b). A task can inherit or override roles assigned to its parent task

(Figure 2 - c). The order of each role can be changed to indicate its priority. An assignment can be made to indicate the priority source (Figure 2 - d).



- Default Policy
- Domain Knowledge
  - Expert
  - Intermediate
- Job Title
  - Accountant
  - Manager
  - Sales
- Security
  - Admin
  - Guest

Role	Operation	Rev.
All-Roles (Allow)		
Cashier (Deny)	Hide	✓
Novice (Deny)	Disable	

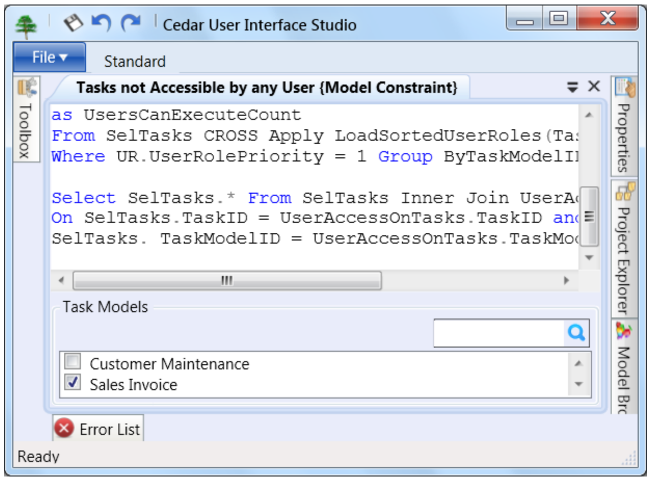
Parent Role Inheritance: [✓ Override] Merge

RBUIS Execution Priority: Role Group Role [✓ Task Role]

Figure 2. Visual Role Allocation on Tasks

The allocation of roles to tasks can also be done through SQL-based rules. RBUIS rules are written in the form of an SQL condition conforming to our meta-model [2]. This condition is assigned roles and allocated to the task models on which it should be executed. *Cedar Studio* provides an editor for RBUIS rules and the ability to validate the SQL syntax and display errors in the "*Error List*".

Due to possible human errors in the allocation of roles to tasks, model verification is required. The example SQL-based constraint illustrated in Figure 3 retrieves all the tasks not accessible by any user in the system. These tasks are then displayed in the "*Error List*" as errors or warnings. Furthermore, the SQL syntax itself can be validated in a similar manner to how RBUIS rules are validated.



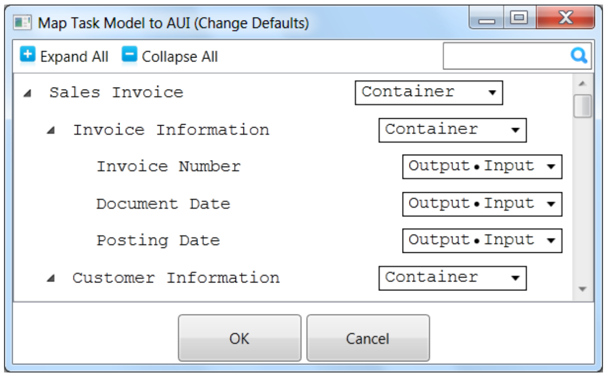
```
as UsersCanExecuteCount
From SelTasks CROSS Apply LoadSortedUserRoles(Ta
Where UR.UserRolePriority = 1 Group ByTaskModelI

Select SelTasks.* From SelTasks Inner Join UserA
On SelTasks.TaskID = UserAccessOnTasks.TaskID an
SelTasks. TaskModelID = UserAccessOnTasks.TaskMo
```

- [ ] Customer Maintenance
- [✓] Sales Invoice

Figure 3. Model Checking Constraints Code Editor

The second level of abstraction, namely AUI models can be automatically generated from task models. It is possible to visually override the default mapping using the UI shown in Figure 4 by allocating each task one or more AUI elements. This option spares the designers from having to individually add, delete, or modify elements on the canvas.

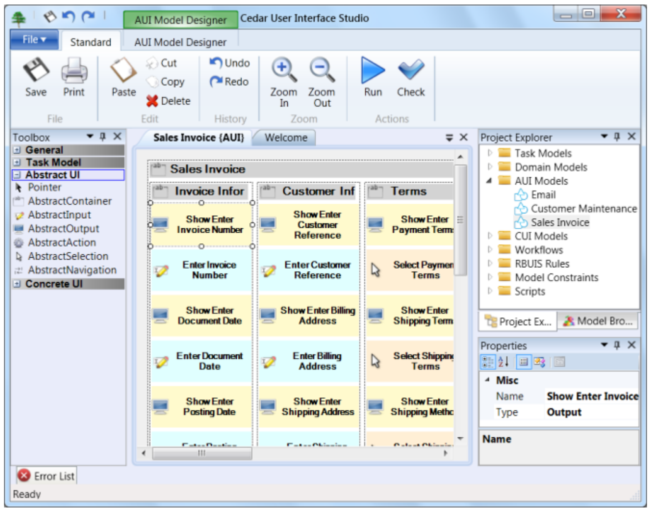


Sales Invoice	Container
Invoice Information	Container
Invoice Number	Output • Input
Document Date	Output • Input
Posting Date	Output • Input
Customer Information	Container

Figure 4. Mapping Task Model to AUI

Abstract User Interface Models

The generated AUI is easily modifiable through the visual design tool illustrated in Figure 5. Simplicity is the main advantage of this tool that supports the specification of AUIs with basic building blocks on a flow-style layout canvas, which could be used by non-technical designers.



Name	Show Enter Invoice
Type	Output

Figure 5. Abstract User Interface Design Tool

Since AUI models are a modality independent representation, the design canvas shows each element as a box with a name, icon, and color. This tool allows AUI containers to be nested within one another and provides an easy-to-use flow style for visually manipulating the AUI elements. The properties box allows the modification of an element's properties including its type. As suggested in existing literature [22], placeholder elements are used upon deletion to maintain the mapping between the models. The type of the placeholder can be switched to an AUI element type without affecting the mapping. New elements can be added from the toolbar and manually mapped to their related tasks in the task model.

CUI models can be automatically generated from AUI models similarly to how AUI models are generated from task models. An interface, similar to the one in Figure 4, is also provided for manually adjusting the default mappings.

Concrete User Interface Models

The input of the human designer is highly desirable for achieving higher usability [22] through the manipulation of concrete objects rather than just an abstract representation [6]. Providing a robust CUI design tool helps designers in providing their input on the look on feel of the UI. Visual user interface builders provide a graphical means for expressing graphical concepts thereby providing a low threshold due to the reduction of the learning curve [18].

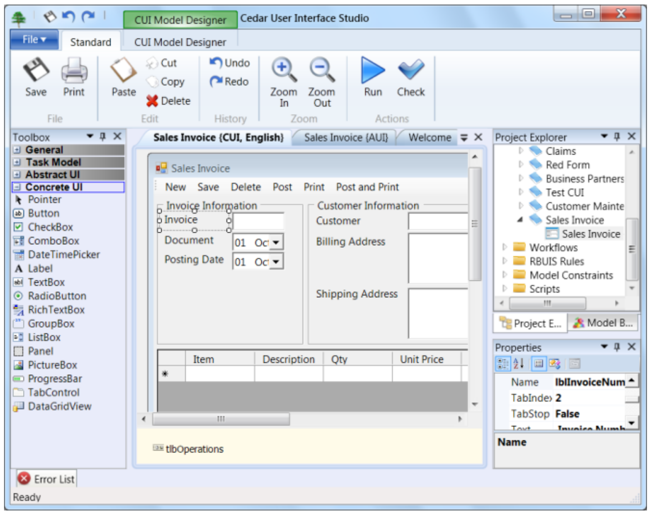


Figure 6. Concrete User Interface Design Tool

Cedar Studio provides a feature-rich CUI design tool (Figure 6) by seamlessly integrating and extending the "*Windows Forms*" design tool of "*Visual Studio .NET*". This design tool has been time tested through its usage in developing UIs for many enterprise applications. Similar to that of the AUI, the CUI design tool supports placeholders upon deletion in addition to complete deletion of elements which could be manually replaced and mapped to the AUI model. A rich toolbar is provided including both basic (e.g., date-time picker) and advanced (e.g., data grid) widgets required by enterprise applications.

Adaptive Behavior Workflows

Workflows are common in enterprise applications for representing business rules. Our approach takes advantage of workflows to represent adaptive behavior both visually and through code. This approach gives the opportunity for both developers and I.T. personnel to implement this behavior through a straight forward visual canvas (Figure 7 - a). Similar to the task model design and role assignment tool, the visual and code-based combination also enhances *expressive match*. Furthermore, *expressive leverage* by promoting reusability [19] is achieved by supporting the integration of reusable visual components and scripts.

Workflows can be assigned roles and the CUI models to be executed on. We integrated the "*Windows Workflow*" design tool of "*Visual Studio .NET*". This tool provides a rich set of visual programming constructs (Figure 7 - b), which can be dynamically extended with custom activities (Figure 7 - c) written in "*C#*" or "*VB.NET*". One of the extensions we have built supports calling adaptive behavior written in the scripting language "*Iron Python*". *Cedar Studio* stores workflows in an XML format that allows any workflow to be dynamically loaded and executed.

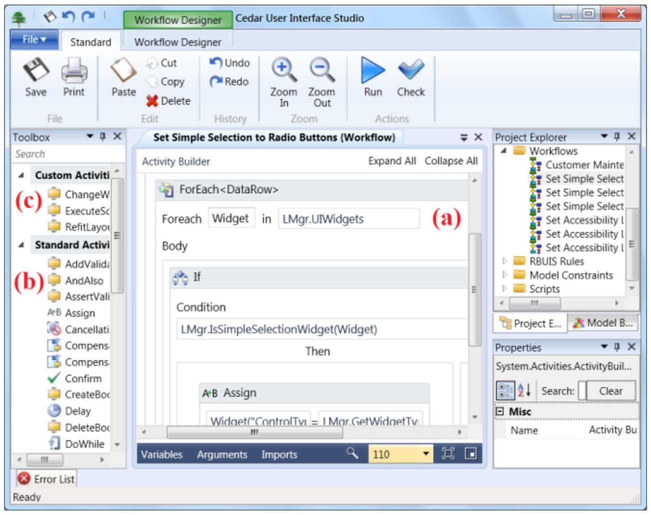


Figure 7. Adaptive Behavior Workflow Design Tool

Cedar Studio supports an "*Iron Python*" script editor. Scripts are created separately and can be called from within any workflow by selecting the script, specifying the method to call, and passing it the appropriate parameters. The entire process is done visually through the workflow design tool.

Testing Adapted UIs from within Cedar Studio

Cedar Studio provides developers with the ability to run the devised UIs with and without adaptations using "*Run*" and "*Run As*" commands respectively. By combining this feature with the previously described design tools, we achieved *flexibility* in terms of supporting rapid design changes that can be performed and evaluated by the developers [19].

The "*Run*" command simply executes the initial version of the UI whereas "*Run As*" prompts the developer to enter a user identifier and executes the UI version corresponding that user's roles. This functionality allows developers to test UIs and adaptive behavior from within the IDE.

The UI illustrated in "Figure 8 – Left" represents a fully-featured "*Sales Invoice*", which is one of the cases we used for testing RBUIS and *Cedar Studio*. We considered a role called "*Cashier*" requiring a simplified version of this UI.

By allocating the role "*Cashier*" to the appropriate tasks, applying the necessary adaptive behavior workflows, and running the UI with a user allocated the role "*Cashier*", the version illustrated in "Figure 8 – Right" will be displayed.

When the user's role is modified (e.g., *Cashier* to *Manger*, *Novice* to *Expert*, etc.), the adaptation will dynamically change according to the new role. This conforms to the concept of multi-layer interface design [23].

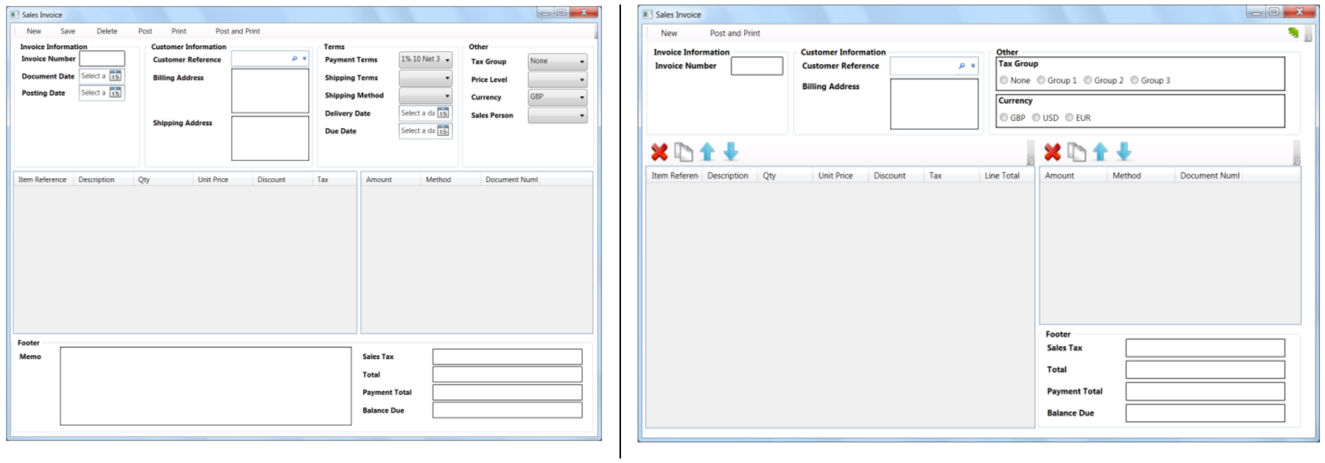


Figure 8. Sales Invoice Initial Version (Left) and Simplified Version (Right)

ASSESSING CEDAR STUDIO

Cedar Studio was practically assessed by constructing a few enterprise resource planning (ERP) UIs, such as the one shown in Figure 8, and basic adaptive behavior. One of the main observed strengths of using *Cedar Studio* in practice is in its design tools (AUI, CUI, and Workflow) that are based on existing mature *Visual Studio* components. The task model design tool can be developed further to reach the same level of maturity and the code editors can be enhanced by adding intelligent-sense. In the future this assessment will be expanded and applied in an industrial scenario.

In the previous sections we described the advantages of *Cedar Studio* in terms of criteria such as *flexibility, expressive match*, and *expressive leverage*. In this section, we assess *Cedar Studio* based on another set of criteria recommended for user interface development tools [18]:

- **Threshold and Ceiling**: The "*threshold*" represents the difficulty in learning and using the tool, and the "*ceiling*" relates to how advanced the tool's outcome can be. An ideal tool would have a low threshold and a high ceiling.
- **Path of Least Resistance**: Developers should be guided to construct the UI in an appropriate manner by making the right approach easier to follow than the wrong one.
- **Predictability**: Any automated approach provided by the tool should be predictable to the developers using it.
- **Moving Targets**: The tool should be able to keep up with the rapid developments in user interface technology.

Upon designing and developing *Cedar Studio* we tried to meet the above mentioned criteria as much as possible.

It might not be feasible to achieve low threshold and high ceiling in all cases. This is due to the learning curve created by any additional features that would allow the tool to produce a more advanced outcome. Yet, we aimed towards achieving a proper balance between *threshold* and *ceiling*. We integrated automated generation and synchronization between models (*low threshold*), alongside the possibility of conducting manual adjustments (*high ceiling*). Furthermore, if developers understand the semantics of the model they can use the visual design tools to produce an advanced outcome (*medium threshold / high ceiling*). In the cases where coding could be used a visual design tool alternative was provided (e.g., *Visual Workflows* instead of *Scripts*, *Visual Role Assignments* instead of *RBUIS Rules*) or the language the most familiar to developers was chosen (e.g., SQL instead of OCL for *Model Verification*).

The *path of least resistance* is maintained by allowing developers to easily apply the model-driven approach. The automated generation of models representing the various levels of abstraction and the mapping between them saves the time of having to perform the model design and mapping manually. The automatic generation preserves *predictability* by allowing developers to customize the default mappings between the different model elements (e.g., abstract input to text box). Furthermore, the support for visual adjustment and resynchronization provides an easy way to customize what was automatically generated.

Concerning the *Moving Targets* criteria, the model-driven approach supported by *Cedar Studio* was initially created to absorb the effect of changes in technology and requirements. The model-driven approach allows our IDE to be independent from presentation technologies and to evolve more easily alongside them. If new techniques for building UIs or even new UI types emerge in the future, models are a good approach to cope with such change since it is possible to rely on the existing abstract representations to regenerate different types of concrete user interfaces.

CONCLUSIONS AND FUTURE WORK

This paper presented an overview of *Cedar Studio*, an IDE for developing adaptive model-driven enterprise application user interfaces. *Cedar Studio* supports model-driven UI development, based on the CEDAR architecture, through a set of visual design and code editing tools that can be used by both developers and I.T. personnel. Additionally, *Cedar Studio* supports integrated testing of the devised adaptive behavior by running the developed UI from within the IDE itself. The supported adaptive behavior is primarily targeted at the simplification of enterprise UIs by minimizing the feature-set and optimizing the layout based on the context-of-use. We evaluated *Cedar Studio* conceptually based on a set of criteria suggested by the literature and practically by developing example adaptive enterprise application UIs.

Currently, the user interface models (Task, AUI, and CUI) are supported by visual design tools. We plan on extending *Cedar Studio* with a code view for each of these models for supporting XML-based representations, which could make it easier to define and manage larger models. UI description languages (UIDL) provide technology independent XML-based representation for user interfaces. One promising UIDL to consider is UsiXml [13]. Also, we intend to extend an early-stage tool that we developed in the spirit of *Cedar Studio* for engaging user communities in the adaptation process [3]. We intend to evaluate *Cedar Studio* with an industrial case study. The study would involve asking both developers and I.T. personnel to use the tool for developing real-life user interfaces and providing their feedback on how *Cedar Studio* and the model-driven approach compare to their traditional development techniques and tools.

ACKNOWLEDGMENTS
This work is partially funded by ERC Advanced Grant 291652.

REFERENCES

1. Akiki, P.A., Bandara, A.K., and Yu, Y. Using Interpreted Runtime Models for Devising Adaptive User Interfaces of Enterprise Applications. ICEIS'12, 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. EICS'13, ACM (2013), *Forthcoming*.
3. Akiki, P.A., Bandara, A.K., and Yu, Y. Crowdsourcing User Interface Adaptations for Minimizing the Bloat in Enterprise Applications. EICS'13, ACM (2013), *Forthcoming*.
4. 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, 3, Elsevier (2003), 289-308.
5. Coyette, A. and V, J. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. INTERACT'05, Springer-Verlag (2005), 12-16.
6. Demeure, A., Meskens, J., Luyten, K., and Coninx, K. Design by Example of Graphical User Interfaces Adapting to Available Screen Size. Computer-Aided Design of User Interfaces VI, Springer, (2009), 277-282.
7. Feuerstack, S., Blumendorf, M., Schwartze, V., and Albayrak, S. Model-based Layout Generation. AVI '08, ACM (2008), 217-224.
8. Gajos, K.Z., Weld, D.S., and Wobbrock, J.O. Automatically Generating Personalized User Interfaces with Supple. Artificial Intelligence, Elsevier (2010), 910-950.
9. García Frey, A., Calvary, G., and Dupuy-Chessa, S. Xplain: An Editor for Building Self-Explanatory User Interfaces by Model-Driven Engineering. EICS'10, ACM (2010), 41-46.
10. García Frey, A., Céret, E., Dupuy-Chessa, S., Calvary, G., and Gabillon, Y. UsiComp: An Extensible Model-Driven Composer. EICS'12, ACM (2012), 263-268.
11. Kramer, J. and Magee, J. Self-Managed Systems: an Architectural Challenge. FOSE'07, IEEE (2007), 259-268.
12. Krasner, G.E., Pope, S.T. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. JOOP 1, 3, SIGS (1988), 26-49.
13. Limbourg, Q. and Vanderdonckt, J. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. ICWE'04 Workshops, Rinton Press (2004), 325-338.
14. Lin, J. and Landay, J.A. Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces. CHI'08, ACM (2008), 1313-1322.
15. Meskens, J., Vermeulen, J., Luyten, K., and Coninx, K. Gummy for Multi-Platform User Interface Designs: Shape me, Multiply me, Fix me, Use me. AVI'08, ACM (2008), 233-240.
16. Michotte, B. and Vanderdonckt, J. GrafiXML, a Multi-target User Interface Builder Based on UsiXML. ICAS'08, IARIA (2008), 15-22.
17. Montero, F. and López-Jaquero, V. IdealXML: An Interaction Design Tool. Computer-Aided Design of User Interfaces, Springer (2007), 245-252.
18. Myers, B., Hudson, S.E., and Pausch, R. Past, Present, and Future of User Interface Software Tools. TOCHI 7, 1, ACM (2000), 3-28.
19. Olsen,Jr., D.R. Evaluating User Interface Systems Research. UIST'07, ACM (2007), 251-258.
20. Paterno, F. Model-based Design and Evaluation of Interactive Applications. Springer-Verlag (1999).
21. Pérez-Medina, J.-L., Dupuy-Chessa, S., and Front, A. A Survey of Model Driven Engineering Tools for User Interface Design. Task Models and Diagrams for User Interface Design. Springer (2007), 84-97.
22. Pleuss, A., Botterweck, G., and Dhungana, D. Integrating Automated Product Derivation and Individual User Interface Design. VaMoS'10, Universitat Duisburg-Essen (2010), 69-76.
23. Shneiderman, B. Promoting Universal Usability with Multi-Layer Interface Design. CUU'03, ACM (2003), 1-8.
24. LEONARDI. http://www.leonardi-free.org.
25. OpenXava. http://www.openxava.org.
26. Himalia.net. http://bit.ly/HimaliaDotNet.