

# MODEL-DRIVEN ENGINEERING OF BEHAVIORS FOR USER INTERFACES IN MULTIPLE CONTEXTS OF USE

Efrem Mbaki Luzayisu and Jean Vanderdonckt

*Université catholique de Louvain*

*Pole of Research on Information and Services Management and Engineering (PRISME)  
Louvain School of Management (LSM), Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)*

## ABSTRACT

This paper describes a model-driven engineering approach for specifying, designing, and generating consistent behaviors in graphical user interfaces in multiple contexts of use, i.e. different users using different computing platforms in different physical environments. This methodological approach is structured according to the levels of abstraction of the Cameleon Reference Framework: task and domain, abstract user interface, concrete user interface, and final user interface. A behavior model captures the abstractions of the behavior in terms of abstract events and abstract behavior primitives in the same way a traditional presentation model may capture the abstraction of the visual components of a user interface. The behavior modeled at the abstract level is reified into a concrete user interface by model-to-model transformation. The concrete user interface leads to the final user interface running thanks to code by model-to-code generation.

## KEYWORDS

Behavior model, model-to-model transformation, model-to-code generation, user interface extensible markup language.

## 1. INTRODUCTION

For many years, the hardest part in conceptual modeling of User Interfaces (UIs) has been its dynamic part. All other aspects, such as presentation, help, tutorial, etc. have received considerable attention and results, especially in model-based approaches and model-driven engineering. We hereby refer to presentation as the static part of a UI such as the description of all windows, dialog boxes, widgets, and their associated properties. In contrast, we hereby refer behavior to as being the dynamic part of a UI such as the physical and temporal arrangement of widgets in their respective containers. The behavior has been also referred to as dialog, navigation, or feels (as opposed to look for presentation). Some typical behaviors are:

- When a language is selected in a list box, the rest of a dialog box is updated accordingly.
- When a particular value has been entered in an edit field, other edit fields are deactivated because they are no longer needed.
- When a validation button is pressed, the currently opened window is closed and another one is opened for pursuing the dialog.

The behavior received limited attention for many reasons: declarative languages that have been typically used for modeling presentation are hard to use for modeling behavior. Procedural languages could be used instead, but these induce a mixed-model-based approach that is complex to implement. Languages used for the final behavior are very diverse (markup or imperative), hold different levels of refinement (ranging from simple properties to sophisticated behaviors), are hard to abstract into one single level of abstraction (especially for different platforms), are hard to implement for model transformation. There is no consensus about what type of model should be used: some models exhibit a reasonable level of expressiveness, but prevent the designer from specifying advanced behaviors while other languages benefit from more expressiveness, but are more complex to handle, especially for non-trained designers. Which appropriate modeling approach is also open: taking the greatest common denominator across languages (with the risk of limited expressiveness) or more (with the risk of non-support), especially because many different implementations exist based on code templates and skeletons, deterministic algorithms, graph transformation, etc.

Finally, we are not aware of any existing approach that consistently applies model-driven engineering principles for UI behavior from the highest level (computing-independent model) to the lowest level (platform-specific model). Existing approaches only address some parts of some levels.

This paper is aimed at addressing these shortcomings by consistently applying principles of model-driven engineering starting with model-to-model transformation until model-to-code generation for multiple platforms. The rest of this paper is structured as follows: Section 2 summarizes the shortcomings of existing approaches. Section 3 introduces the conceptual modeling of behaviors used in this research. Section 4 presents an overview of the approach structured into steps that are supported by the software implemented. Section 5 explains the case study that we implemented to test methodology robustness. Section 6 concludes the paper and addresses some avenues.

## 2. RELATED WORK

Many different types of conceptual modeling and techniques of behaviors have been used over years [1-4, 7-21], some of them with some continuation, such as, but not limited to: Backus-Naur Form (BNF) grammars [9,12], state-transition diagrams in very different forms (e.g., dialog charts [1], dialog flows [3], abstract data views [7], dialog nets [8], windows transitions [14]), state charts [10] and its refinement for web applications [20], and-or graphs coming from Artificial Intelligence (e.g., function chaining graphs [19]), event-response languages, and Petri nets [2]. Comparing all these models in a sound way represents a significant contribution to what is yet to appear. Green [9] compared to a long time ago three dialog of them to conclude that some models share the same expressivity, but not the same complexity.

Cachero et al. examine how to model the navigation of a web application [4]. In [6], the context model is the basic model that could influence a behavior model at different steps of the UI development life cycle. So far, few attempts have been made to structure the conceptual modeling of behaviors in the same way as it has been done for presentation, the notable exception being applying StateWebCharts with Cascading style sheets [21] in order to factor out common parts of behaviors and to keep specific parts locally.

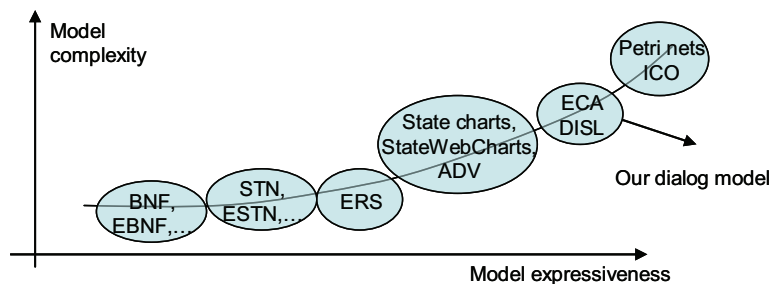


Figure 1. Model complexity as a function of their expressiveness.

Fig. 1 graphically depicts some behavior models in families of models. Each family exhibits a certain degree of model expressiveness (i.e., the capability of the model to express advanced enough behaviors), but at the price of a certain model complexity (i.e., the ease with which the dialog could be modeled in terms specified by the meta-model). At the leftmost part of Fig. 1 are located (E)BNF grammars since they are probably the least expressive behavior models ever. Then we can find respectively State Transitions Networks and their derivatives, then Event-Response Systems. Petri nets [3] are probably the most expressive models that can be used to model behaviors, but they are also the most complex to manipulate for inexperienced designers. Therefore, we believe that we could be less expressive and complex than Petri nets if Event-Condition-Action (ECA) systems are considered, such as in the DISL [18], UIML [11] and UsiXML (<http://www.usixml.org> - Fig. 2) User Interface Description Languages (UIDLs). In UIML, a behavior is defined as a set of condition-action rules that define what happens when a user interacts with an UI element, such as a button.

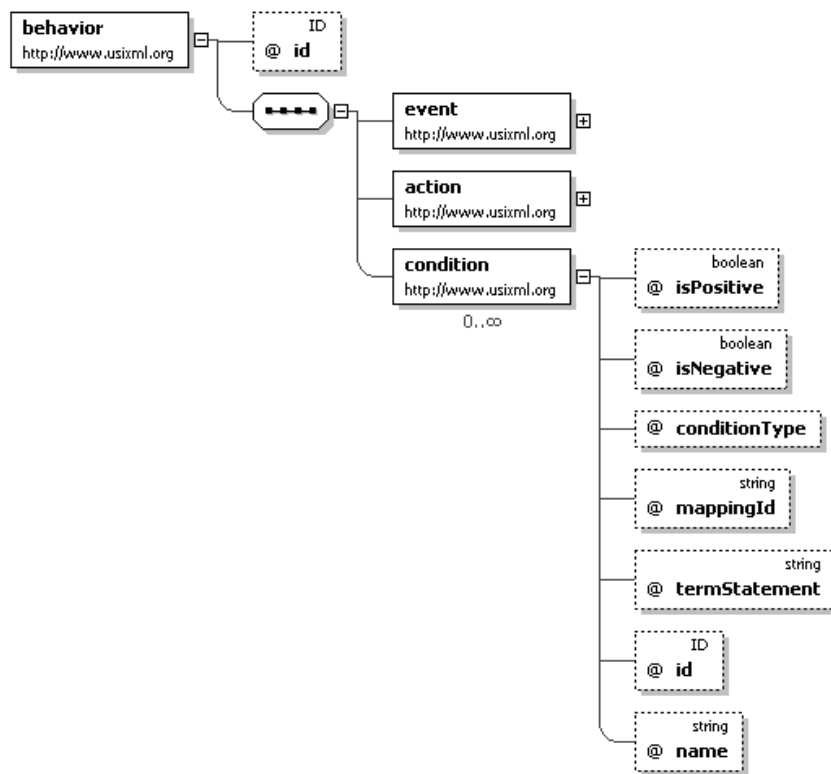


Figure 2. Modeling a behavior in UsiXML V1.6.8. Source: [http://www.usixml.org/documentation/usixml1.8.0/UsiXML.xsd.html\\_h-1518477259.html](http://www.usixml.org/documentation/usixml1.8.0/UsiXML.xsd.html_h-1518477259.html).

In UsiXML, a behavior (Fig. 2) is defined as a set of ECA rules, where: an event can be any UI event that is relevant to the level of abstraction where we are (it could be abstract or concrete).

- A condition can state any logical condition on a model, a model element, or a mapping between models.
- A condition is itself decomposed in term statements that can be reused, which are in turn decomposed into terms.
- An action can be any operation on widgets that are relevant to the level of abstraction where we are (abstract or concrete).

The UsiXML language describes the UI for multiple contexts of use such as character UIs, graphical UIs, auditory and vocal UIs, virtual reality, and multimodal UIs. As a language explicitly based on the Cameleon Reference Framework (CRF) [5], it adopts four development steps:

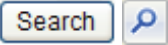

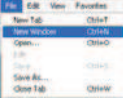
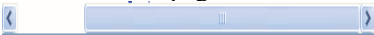
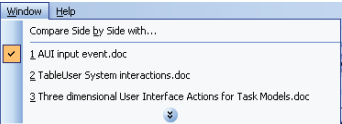

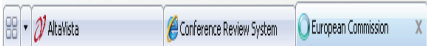
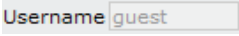

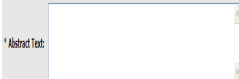

1. *Task & Concepts (T&C)*: describe the various users' tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed.

2. *Abstract UI (AUI)*: defines abstract containers (AC) and individual components (AIC) [15] two forms of Abstract Interaction Objects (AIO) by grouping subtasks according to various criteria (e.g., task model structural patterns, cognitive load analysis, semantic relationships identification), a navigation scheme between the container that selects the abstract individual component for each concept so that they are independent of any interaction modality, such as graphical, vocal, tactile, or haptic modalities. The AUI is said to be independent of any interaction modality.

3. *Concrete UI (CUI)*: concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. It abstracts a final UI into a UI definition that is independent of any computing platform. While the AUI is independent of any interaction modality, a CUI assumes that a particular interaction modality has been chosen, but that this CUI remains independent of any computing platform.

4. *Final UI (FUI)*: is the operational UI i.e. any UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after compilation of code in an Integrated Development Environment (IDE)).

Table 1. Some mappings between abstract (AUI) and concrete (CUI) events

AUI input event	Example of CUI events		
	Graphical		Vocal
	GUI	Web	
Trigger an AIC with control facet	Click on a push button or icon 	Click on a push button  Open a window  Play a video file (start)	vocalInput and grammar and submit
Trigger an AIC with output facet		Scroll a web page without scrolling 	vocalPrompt or audio or vocalFeedback or vocalMenu with vocalMenuItems
Trigger an AIC with navigation facet	Select a view of an AC 	Select back or forward to navigate among pages  Select a view of an AC 	vocalNavigation or (vocalInput and grammar and connect)
Trigger an AIC with input facet		TextField single line in which the user can enter text.  PasswordField hides characters that user enters  TextArea in which the user can enter text 	vocalInput or record
Close an AC		Close or exit from a window 	Break or exit

### 3. MODEL-DRIVEN ENGINEERING OF BEHAVIORS

In order to apply traditional model-driven engineering techniques, we need to manipulate a behavior model that is expressive enough to accommodate advanced behaviors at different levels of granularity and different levels of abstraction, while allowing some structured design and development of corresponding behaviors. For this purpose, our conceptual modeling consists of expanding ECA rules towards *dialog scripting* (or *behavior scripting*) in a way that is independent of any platform (Fig. 3). For this purpose, we define five levels of granularity:

1. *Object-level behavior modeling*: this level models the behavior at the level of any particular object, such as a CIO or a AIO. In most cases, UI toolkits and Integrated Development Environments (IDEs) come with their own widget set with built-in, predefined behavior that can be only modified by overwriting the methods that define this behavior. Only low-level toolkits allow the developer to redefine an entirely new behavior for a particular widget, which is complex.

2. *Low-level container behavior modeling*: this level models the behavior at the level of any container of other objects that is a leaf node in the decomposition. Typically, this could be a terminal AC at the AUI level or a group box at the CUI level in case of a graphical interaction modality.

3. *Intermediary-level container behavior modeling*: this level models the behavior at the level of any non-terminal container of objects that is any container that is not a leaf node in the container decomposition. If the UI is graphical, this could be a dialog box or various tabs of a tabbed dialog box.

4. *Intra-application behavior modeling*: this level models the behavior at the level of top containers within a same interactive application such as a web application or a web site. It therefore regulates the navigation between the various containers of a same application. For instance, the Open-Close patterns mean that when a web page is closed, the next page in the transition is opened.

5. *Inter-applications behavior modeling*: since the action term of a ECA rule could be either a method call or an application execution, it is possible to specify a same behavior across several applications by calling an external program. Once the external program has been launched, the behavior that is internal to this program (within-application dialog) can be executed.

Now that have defined these five levels, we introduce the concepts used towards the conceptual modeling of behaviors that could be structured according to the five aforementioned levels of granularity. Fig. 3 depicts the global structuring of these concepts:

1. *Interactive Object*: interactive objects are particular objects used in the UI design and implementation. Strictly speaking, any way for a human being to “interface” with a machine is through a UI. The touch on its GPS or its microwave is a UI, the dial on its washing machine or the choice of television channels using a remote control are UI. Theoretically speaking, there exist interactive objects without physical representation. These objects are known as abstract. In the same way, there exists a classification of the objects according to their type. As all objects, interactive objects are characterized by their attributes (properties), behavior. Some objects have the capacity to react to the actions which they undergo by generating events. We can define a class as an abstraction of objects characteristics, including its attributes (fields or properties), its behaviors (methods or operators) and its events. In other terms, a class can be considered here as a model.

2. *Instance*: an instance is an individual object of a certain class. While a class is just the type definition, an actual usage of a class is called "instance". Each instance of a class can have different values for its attributes. At a given moment, the state of an instance is the set of its attributed values. By respecting the encapsulation i.e., the process of hiding all of the attributes of an object from any outside direct modification, object methods can be used to change an instance state.

3. *Toolkit*: in the modern graphical environments of design or development, Toolkits are represented as an objects box in which the developer can drag and drop items. In a UI Dialog Editor, the software that we developed in order to support this model-driven engineering of UI behaviors, the toolkit is characterized by its name, its level (e.g., a version), and a series of templates describing how this toolkit implements particular behaviors. Three values are accepted for the level: abstracted, concrete or final.

4. *Project*: any work session of UI Dialog Editor supposes the opening of a project. Indeed, a project can be viewed as a collection of interactive objects, or more precisely a collection of instances of interactive objects. To open and/or create a project, the user must first choose his toolkit.

5. *User Interface*: we define a user interface as the end result of a project. It represents a structured and harmonious set of interactive objects through which developer can implement interactive actions

6. *Mapping*: In the context of UI Dialog Editor, Mapping is a mechanism which makes it possible to define rules of transformation from a Toolkit towards another, possibly the same one. A given object can change with another or several others. In this second case, rules of transformations are necessary. They are written by means of the regular expressions applied to names of interactive objects. We can notice that the definition of a mapping is based primarily on two toolkits; a source and a destination. Its application requires presence of a user interface relating to the source Toolkit to lead to an interface in the destination Toolkit. It's possible to apply the same mapping into to several interfaces.

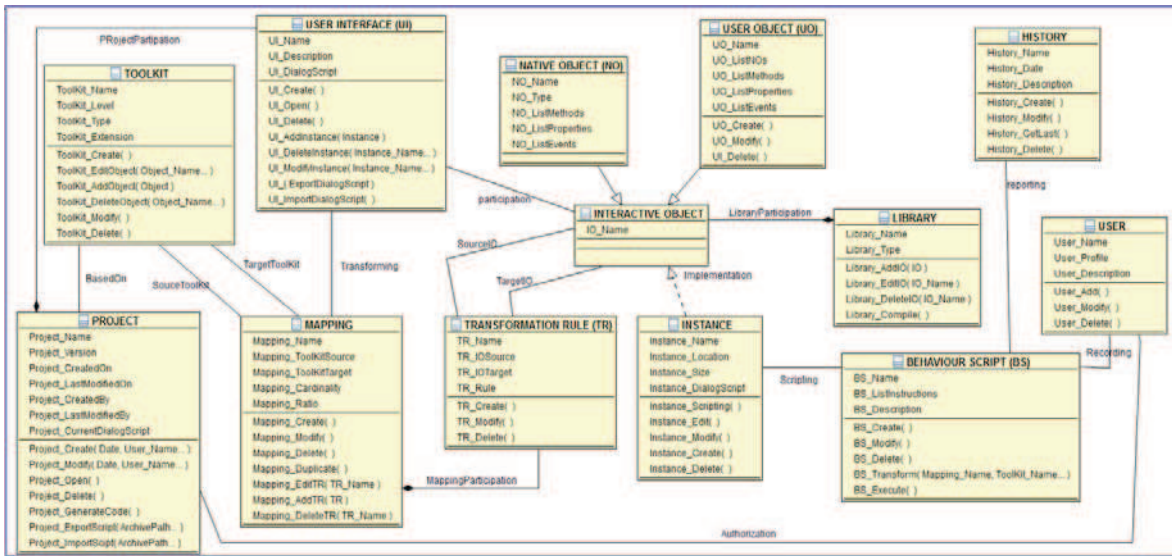


Figure 3. Conceptual modeling of behaviors for model-driven engineering.

7. *Behavior Script*: a script of dialogues is a sequential text furnished with the logic and conditional elements. It describes the desired actions according to a given interaction scenario. An action can be the change of an attribute value; the call of a mathematical function, the opening or the closing of a user interface, etc.

8. *History*: the history is the only means where we can ensure the constant traceability of scripts. It is possible to know which has modified which script of the dialogues and how. Thus, it will be possible to validate or cancel a given action.

We implemented a graphic dialog editor in which all these classes are exploited. This editor is presented in the next section.

#### 4. DIALOG EDITOR IMPLEMENTATION

To support the above meta-model, we exploit MDA to build a graphic Dialog Editor in which Models are objects boxes, called Toolkit by abuse language. Models are organized in three levels (abstract, concrete and final according to CAMELEON architecture. Model elements are Interactive Objects (IO). In this context, a User Interface Project (UIP) is a sub-box in which all objects belong to a same given Model. In others terms, programming with Visual Basic 6 (VB6) and Visual Basic for Application (VBA), we implemented a dialog editor which

Is based primarily on the concept of objects Box. Developer has the choice between creating his own objects, using existing interactive objects or making both. It is important to determine which attributes, methods and events are necessary in dialog script.

1. Gives freedom concerning the level of specification. The user can choose to specify his project at the abstract, concrete or final level;

2. Generates the user interface at final level according to the platform toolkit. The resulting executable proposes architecture with three layers: The user interface which offers all interaction possibilities, the functional machine which contains semantic services and pilots database accesses and, the dialog controller which pilots exchanges between the two first layers;

3. Provides reification and concretization functionalities. Indeed, the user could, for example, use the same abstract specification to provide two or several different concrete specifications. In the same way, it could start from a concrete specification to lead to another concrete specification by skews of the mappings; and

4. Manages dialog scripts traceability. It is constantly possible to know who did what which day and at what time; and also, if necessary, it's possible to cancel recent modifications, or simply to revert to or resume with an old version of a given project

The Dialog Editor architecture is composed of four components presented in the figure 4 below.

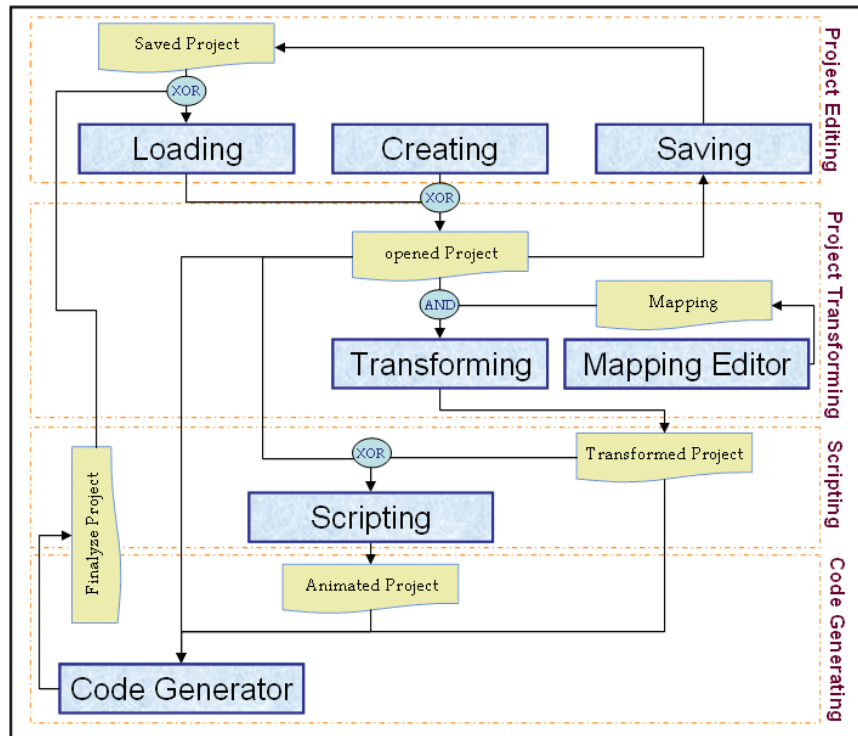


Figure 4. Dialog Editor Components

1. Project editing component proposes the functionalities concerning the creation of a new project. Firstly, it is necessary to choose the level of abstraction and the box with which to work. Then, by adding object after object, to build its project;

2. Project transforming component proposes mapping interfaces and functionalities of transforming a given project to another. It's possible to change specification level and/or specification toolkit.

3. Scripting component defines syntax and semantic of dialog language. In the objective to unify the specification of the interaction, we built a generic language which offers expressions for intra-object, intra-window, inter-windows, intra-process and inter-process dialog scripts. This generic language has a parser and a translator towards some computer programming languages.

4. Code generator integrate the algorithms which make it possible to transform a specification into a final project which is usable

We insist on the fact that the editor is developed primarily in Visual BASIC 6. Its interfaces are graphic. The internal data respect XML models. Looking in from the outside, the editor proposes four components or modules as shows in the dataflow diagram. The figure 5 below concerns project edition in BCHI Dialog Editor. It illustrates a project, named "106 – CTI Order AUI", where objects belong to "Abstract User Interface Model" Box. Each object is defined by its type, its name and its properties.

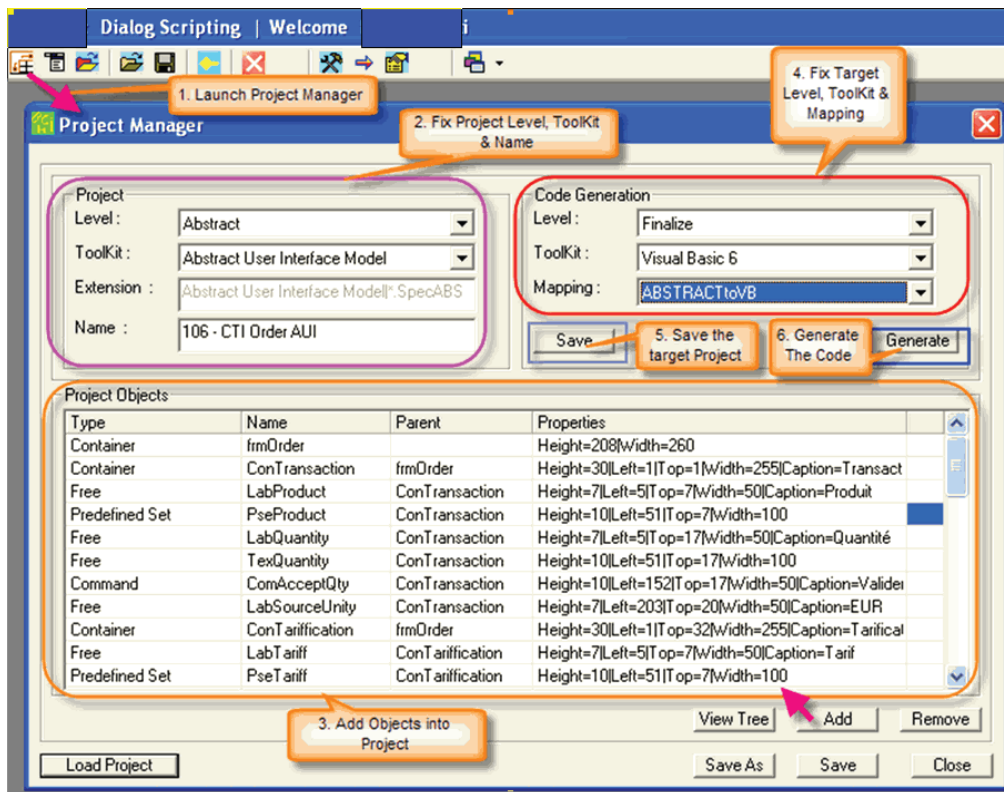
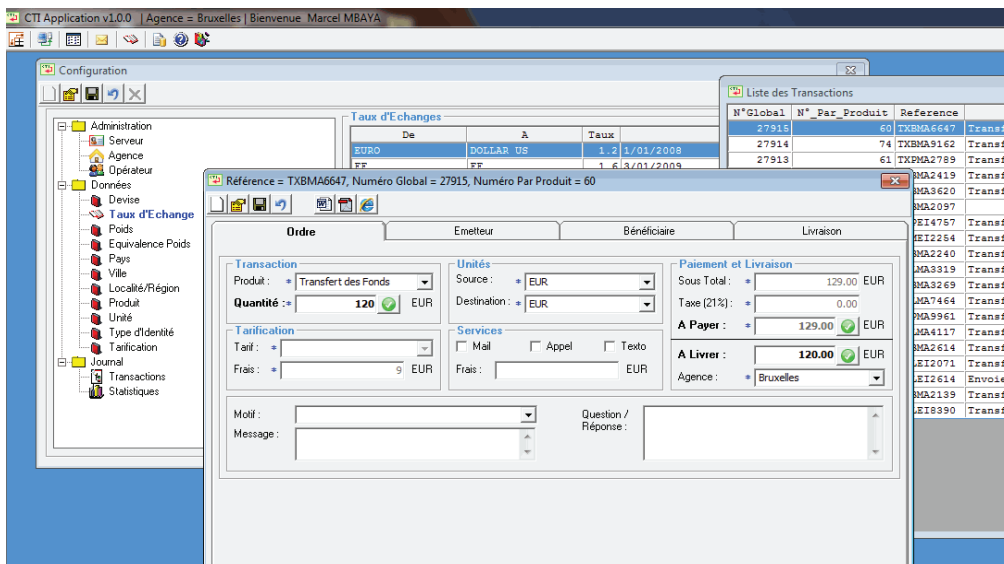


Figure 5. Project Interface

## 5. CASE STUDY

With our dialog editor, we developed a software aimed at covering the activities of a company which is specialized in the international transfer of money and import express worldwide services.





We needed an application based on a real case. But, we also needed an End-User oriented dialog application. For these two reasons, we asked the employers of CTI Company, whose head office is located at Liege in Belgium, to participate to the analysis and the validation of this software. Insofar as the employers of CTI Company speak only French, interfaces are in French in order to allow a better communication. In this context, commercial transaction is defined by: a Shipper (the customer which deposits the money or the object), a Sender (the person for which is intended the money or the object) and An Order (the details of transaction contents).

The recording of a new transaction requires other information which is saved beforehand in the database. Without being exhaustive, we can enumerate type of products, list of customers, list of currencies and their correspondences, tariffs, etc. Table 2 gives an outline temporary of the made tasks and, expressed as a percentage indicating the parts carried out manually and those generated automatically.

Table 2. Tasks distribution

Task	Timing	Manuel	Automatic
Interactive Objects library implementation	25%	100%	0%
Shipper, Sender and Order Management	65%	20%	80%
Reporting Management	10%	50%	50%

The interactive objects of the library which we have developed are useful both for the dialog editor and the CTI Application. To meet specific communications needs of company CTI, we integrated in this software, automatic functions of sending SMS and/or emails for transactions.

## 6. CONCLUSION

This paper introduces a particular method for supporting model-driven engineering of UI behaviors that are compliant with the Cameleon Reference Framework (CRF) [6]. For this purpose, a UI Dialog Editor has been implemented that: imports an AUI resulting from a generation from task and concepts, enriches it with dialog scripts that are abstract, apply model-to-model transformations in order to reify these scripts into concrete ones, and then apply model-to-code generation for three computing platforms. Three target contexts of user are supported: HTML for Applications (HTA) in a mobile context, Microsoft Visual Basic 6 and Microsoft Visual Basic for Applications (VBA) in a stationary context. Two operating systems are equally addressed: Microsoft Windows and Mac OS X. Five levels of behavior granularity are considered: object-level (behavior of a particular widget), low-level container (behavior of any group box), intermediary-level container (behavior at any non-terminal level of decomposition such as a dialog box or a web page), intra-application level (application behavior), and inter-application level (behavior across different interactive applications). Intra-container and inter-container behaviors are exemplified throughout a step-wide methodology that is supported by a dialogue editor, a model transformer, and a code generator, integrated into one single authoring environment.

The behaviors that can be modeled according to the approach presented in this paper can indeed lead to automated generation of their corresponding code. But these behaviors are for the moment straightforward and only handle general widget events such as activate, deactivate, etc. Very-fine grained behaviors affecting detailed properties of widgets are not supported.

## ACKNOWLEDGMENTS

The authors would like to acknowledge the support of the ITEA2-Call3-2008026 USI-XML (User Interface extensible Markup Language) European project and its support by Région Wallonne (DGO6). The authors would like to thank their wife and children for their great love and support.

## REFERENCES

- Ariav, G. and Calloway, L.-J. Designing conceptual models of dialog: A case for dialog charts, *SIGCHI Bulletin*, 20, 2 (1988) 23–27.
- Bastide, R. and Palanque, P. A Visual and Formal Glue Between Application and Interaction. *Journal of Visual Language and Computing*, 10, 5 (October 1999) 481–507.
- Book, M., Gruhn, V., and Richter, J. Fine-grained specification and control of data flows in web-based user interfaces. In *Proc. of ICWE'2007* (Como, 16-20 July 2007). LNCS, Vol. 4607, Springer-Verlag, Berlin, 2007, 167–181.
- Breiner, K., Maschino, O., Görlich, D., Meixner, G. Towards automatically interfacing application services integrated in a automated model based user interface generation process. In *Proc. of MDDAUI'2009*.
- Cachero, C., Melia, S., Poels, G., and Calero, C. Towards improving the navigability of Web Applications: a model-driven approach. *European Journal of Informations Systems*, 16 (2007) 420–447.
- Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonck, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15,3 (2003) 289–308.
- Clerckx, T., Van den Bergh, J., and Coninx, K. Modeling Multi-Level Context Influence on the User Interface. In *Proc. of PERCOMW'2006*. IEEE, 2006, 57–61.
- Cowan, D. and Pereira de Lucena, C. Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. *IEEE Trans. on Software Engineering*, 21,3 (1995) 229–243.
- Elwert, T. Continuous and Explicit Dialogue Modelling. In *Proc. of EA-CHI'96*.
- Green, M. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5, 3 (July 1986) 244–275.
- Harel, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8 (1987) 231-274.
- Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., Vanderdonck, J. *Human-Centered Engineering with the User Interface Markup Language*. In “Human-Centered Software Engineering”, Chapter 7, HCI Series, Springer, London, 2009, 141–173.
- Jacob, R.J.K. A specification language for direct manipulation user interfaces. *ACM Transactions on Graphics*, 5, 4 (1986) 283–317.
- Lewis, J.R. IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. *Int. J. of Human-Computer Interaction*, 7, 1 (1995) 57-78.
- Luyten, K., Clerckx, T., Coninx, K., and Vanderdonck, J. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In *Proc. of DSV-IS'2003*. LNCS, Vol. 2844, Springer-Verlag, Berlin, 2003, 203–217.
- Mbaki, E., Vanderdonck, J., Guerrero, J., and Winckler, M. Multi-level Dialog Modeling in Highly Interactive Web Interfaces. In *Proc. of IWWOOST'2008*, CEUR Workshop Proc., Vol. 445, 2008, pp. 38-43.
- Menkhaus, G. and Fischmeister, S. Dialog Model Clustering for User Interface Adaptation. In *Proc. of ICWE'2003*. LNCS, Vol. 2722, Springer-Verlag, 2003, 194–203.
- Meixner, G., Görlich, D., Breiner, K., Hußmann, H., Pleuß, A., Sauer, S., Van den Bergh, J. Proc. of 4th Int. workshop on model driven development of advanced user interfaces. MDDAUI'2009. In *Proc. of IUI 2009*, pp. 503-504.
- Meixner, G., Seissler, M., Nahler, M., Udit – A Graphical Editor For Task Models. In *Proc. of MDDAUI'2009*.
- Pleuß, A. Modeling the User Interface of Multimedia Applications. In *Proc. of MoDELS 2005*, pp. 676-690.
- Pleuß, A. MML: A Language for Modeling Interactive Multimedia Applications. In *Proc. of ISM'2005*, pp. 465-473.
- Reichart, D., Dittmar, A., Forbrig, P., and Wurdel, M. Tool Support for Representing Task Models, Dialog Models and User-Interface Specifications. In *Proc. of DSV-IS'2008*. LNCS, Vol. 5136, Springer, Berlin, 2008, 92–95.
- Rückert, J. and Paech, B. The Guilet Dialog Model and Dialog Core for Graphical User Interfaces. In *Proc. of EIS'2008*. LNCS, Vol. 5247, Springer, 2008, 197–204.
- Schaefer, R., Bleul, S., and Müller, W. Dialog Modeling for Multiple Devices and Multiple Interaction Modalities. In *Proc. of TAMODIA'2006*. Lecture Notes in Computer Science, Vol. 4385, Springer-Verlag, Berlin, 2007, 39–53.
- Traetteberg, H. Dialog modelling with interactors and UML Statecharts. In *Proc. of DSV-IS'2003*. LNCS, Vol. 2844, Springer-Verlag, Berlin, 2003, 346–361.
- Vanderdonck, J., Limbourg, Q., Florins, M. Deriving the Navigational Structure of a User Interface. In *Proc. of Interact'2003*. IOS Press, Amsterdam, 2003, 455-462.
- Winckler, M. and Palanque, P. StateWebCharts: A formal description technique dedicated to navigation modelling of web applications. In *Proc. of DSV-IS'2003*. LNCS, Vol. 2844, Springer-Verlag, Berlin, 2003, 61-76.
- Winckler, M., Trindade, F., and Vanderdonck, J. Cascading Dialog Modeling with UsiXML. In *Proc. of DSV-IS'2008*. LNCS, Vol. 5136, Springer, Berlin, 2008, 121-135.
- W3C State Chart XML (SCXML), State Machine Notation for Control Abstraction, Working Draft, 16 May 2008. Accessible at <http://www.w3.org/TR/SCXML>.