An Eclipse GMF Tool for Modelling User Interaction

Jesús M. Almendros-Jiménez[1], Luis Iribarne[1], José Andrés Asensio[1], Nicolás Padilla[1], and Cristina Vicente-Chicote[2]

[1] Dpto. de Lenguajes y Computación, Universidad de Almería, Spain
{jalmen,jacortes,liribarn,padilla}@ual.es
[2] Dpto. de Tecnologias de la Información, Universidad Politécnica de Cartagena, Spain
Cristina.Vicente@upct.es

Abstract. Model-Driven Development (MDD) has encouraged the use of automated software tools that facilitate the development process from modelling to coding. User Interfaces (UI), as a significant part of most applications, should also be modelled using a MDD perspective. This paper presents an Eclipse GMF tool for modelling user-interaction diagrams –an specialization of the UML state-machines for UI design– which can be used for describing the behaviour of user interfaces.

1 Introduction

The adoption of *Model-Driven Development* (MDD) [16] in the design of *User Interfaces* (UI) allows software architects to use declarative and visual models for describing the multiple perspectives and artifacts involved in UI development. In the literature there are several works [3,22,21,7,20,19,18,14,6,8] whose aim is to apply the MDD approach to the development of user interfaces. In most cases they consider the following models: *task*, *domain*, *user*, *dialogue* and *presentation* models. The task model specifies the tasks that the user will carry out on the user interface. The domain model describes the domain objects involved in each task specified in the task model. The user model captures the user requirements. The dialogue model allows to model the communication between the user and the user interface. Finally, the presentation model describes the layout of the user interface.

In a previous work [2], we have proposed a MDD-based technique for user interface development. It involves the adoption of several UML models, which have been adapted to UI development. In particular, our UI-MDD technique can be applied to the modelling of *WIMP* (*Windows, Icons, Menus*, and *Pointers*) user interfaces. Our proposal mainly uses three models: (1) a dialogue model, which makes use of the so-called *user-interaction diagrams* (a special kind of UML state-machines for UI design); (2) a task model, which makes use of the so-called *user-interface diagrams* (an specialization of the UML use case diagram for UI specification); and, (3) a presentation model, which uses the so-called *UI-class diagrams* (UML class diagrams describing UI objects).

The main goal of our proposed UI-MDD technique is to allow designers to model the user interfaces with UML. The proposed technique is intended to be useful for rapid prototyping. Our technique has been extensively used by our students in the classroom. However, students need to manually carry out most of the tasks due to the lack of a tool for supporting our technique. In particular, one of the most tedious tasks that they have to manually carry out is the modelling of user interfaces by means of user-interaction diagrams.

In this paper we present $\mathcal{ALIR}$, a graphical modelling tool that implements the *core* of our UI-MDD modelling technique, described in [2]. $\mathcal{ALIR}$ allows the development of user-interaction diagrams. This tool has been implemented using some of the MDD-related plug-ins provided by the *Eclipse platform*, namely: the *Eclipse Modelling Framework* (EMF)[5] and the *Eclipse Graphical Modelling Framework* (GMF)[4]. Some further details about these tools will be provided in Section 3.

The structure of the paper is as follows. Section 2 presents the user-interaction diagrams. Section 3 describes the implementation of the Eclipse GMF Tool called $\mathcal{ALIR}$. Section 4 compares our work with existent proposal. Finally, Section 5 presents some conclusions and future work.

2 User-Interaction Diagrams

User-interaction diagrams are an specialization of the UML state-machine for describing the user interaction with the user interface. User-interaction diagrams include *states* and *transitions*. The states represent *data output/request actions*, that is, how the system responds to user interactions showing/requesting data. The transitions are used to specify how the user introduces data or interacts with the system, and how an *event* handles the interaction. Transitions can be *conditioned*, that is, the event is controlled by means of a *boolean condition*, which can either specify *data/business logic* or be associated to a *previous user interaction*. User interaction diagrams can include states with more than one outgoing transition, and which of them is executed depends on either data/business logic or the previous interactions of the user. The initial (resp. final) state is the starting (resp. end) points of the diagrams.

From a practical point of view, it is convenient to use more than one user-interaction diagram for describing the user interface of a software system, since the underlying logic of the user windows is usually too complex to be described in a single model. For this reason, a user-interaction diagram can be deployed in several user-interaction diagrams, in which a piece of the main logic is described in a separate diagram. Therefore, user interaction diagrams can include states which do not correspond to data output/request, rather than they are used for representing a sub-diagram. In this case, the states are called *non-terminal states*; otherwise, they are called *terminal states*.

Our proposed modelling technique provides a mapping between UI models and the *Java Swing package*. The wide acceptance of the *Java* technology: *applets*, *frames*, and *event-handlers*, etc for UI design guarantees the practical application of our proposal. The mapping to Java can be viewed as a *concrete modelling* of a more *general modelling technique*. In such mapping, UI components can be classified as *input* (e.g. a *button*) or *output/request* components (e.g. either a *text field* or a *list*). *Input/output* components are labelled by means of *UML stereotypes*. For instance, the `<<JTextField>>` and`<<JList>>` stereotypes are used in states, while the stereotype `<<JButton>>` is used in transitions.

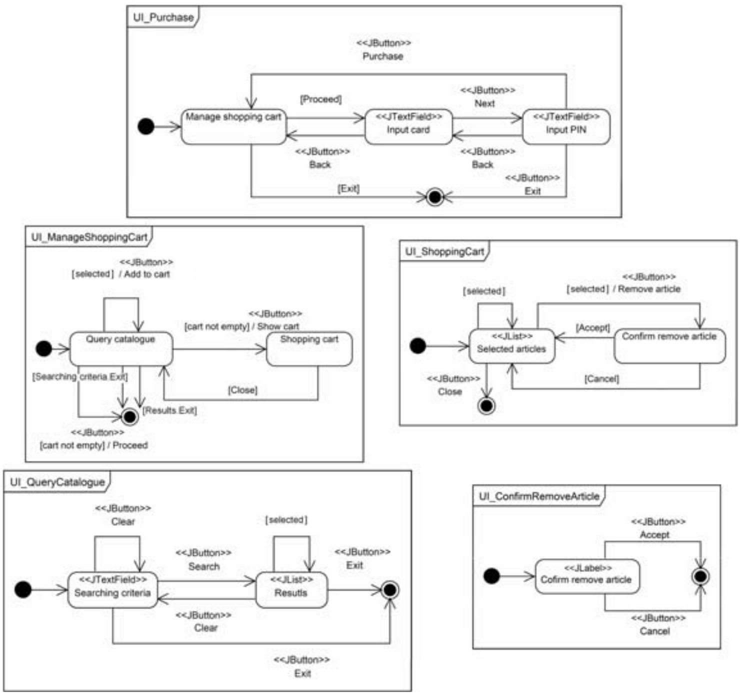

Fig. 1. The dialogue model of the purchase task

As running example, we will consider an *Internet Shopping System*. Figure 1 shows examples of user-interaction diagrams. They describe the *purchase* task. The *main* user-interaction diagram is `UI_Purchase`. This diagram includes non-terminal states (i.e. non-stereotyped states), which are described by means of *secundary* user interaction diagrams. They are sub-diagrams of the main diagram. Both the name of the non-terminal states and the name of its associated diagrams must be the same. The running example shows how the customer carries out the purchase task by querying from a catalogue and by adding or removing articles to/from a shopping cart. After, the shopping system requests the customer a card number and a PIN to carry out the order. The boolean conditions occurring in transitions specify the requirements to be fulfilled for state change. For instance, the *"cart not empty"* condition, in the `UI_ManageShoppingCart` diagram, means that the shopping cart can be only reviewed whenever it is not empty.

2.1 User-Interaction Diagrams Metamodel

User interaction diagrams can be considered as an specialization of UML state-machines in which states and transitions can be stereotyped by means of UI component names, but also states can represent subdiagrams. In order to implement in our tool user-interaction diagrams, we have to move to the UML metamodel. Figure 2 shows the user-interaction diagram metamodel (a *Platform-Independent Model* (*PIM*) perspective). The metamodel can be mapped to the elements of user-interaction diagrams (a *Platform-Specific Model* (*PSM*) view). Figure 3 summarizes the elements of user-interaction diagrams, and Table 1 describes the mapping of the elements defined in Figure 3 and the metamodel described in Figure 2. Due to lack of space, we have described the mapping of a subset of the elements of the meta-model in a subset of the elements of user-interaction diagrams. They are enough for the running example. The metamodel of Figure 2 includes the following elements:

States. They necessarily fall into one of the two following categories: *terminal states* (TS) and *non-terminal states* (NTS). A terminal state is labelled with a *UML stereotype*, representing a *Java* data output/request UI component. A non-terminal state is not labelled, and it is described by means of another *user-interaction diagram.*

Pseudo-states. They necessarily fall into one of the following categories: *initial* pseudo-state, *final* pseudo-state, *choice* states: which define alternative paths of type "OR" between two or more interactions and, finally, *fork/join* pseudo-states, which defines paths of type "AND". For instance, using a fork and a join, one can specify that the user can introduce the card number and the PIN in any order.

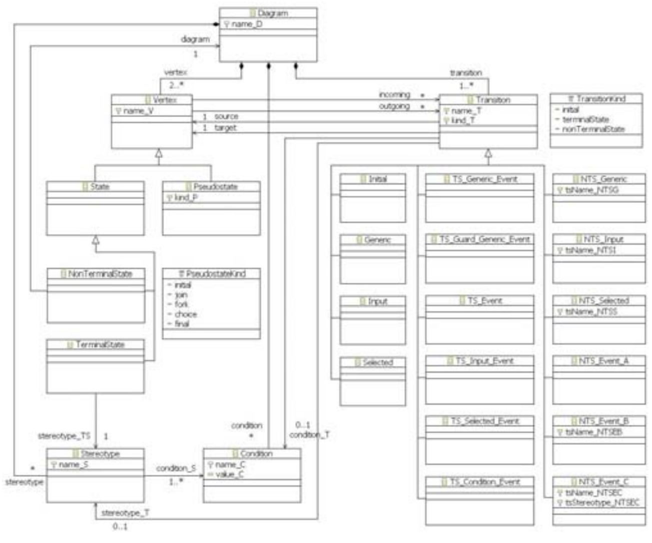

Fig. 2. The metamodel of user-interaction diagrams

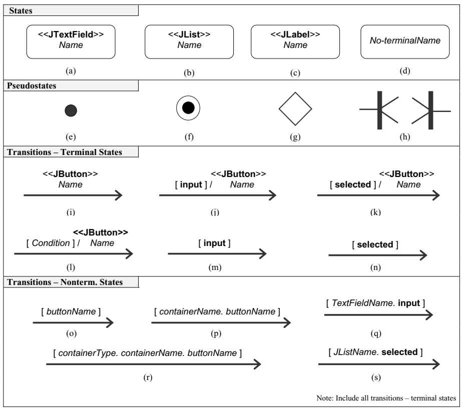
States
<<JTextField>> Name
(a)
<<JList>> Name
(b)
<<JLabel>> Name
(c)
No-terminalName
(d)
Pseudostates
(e)
(f)
(g)
(h)
Transitions – Terminal States
<<JButton>> Name
(i)
<<JButton>> [input] / Name
(j)
<<JButton>> [selected] / Name
(k)
<<JButton>> [Condition] / Name
(l)
[input]
(m)
[selected]
(n)
Transitions – Nonterm. States
[buttonName]
(o)
[containerName. buttonName]
(p)
[TextFieldName. input]
(q)
[containerType. containerName. buttonName]
(r)
[JListName. selected]
(s)
Note: Include all transitions – terminal states

Fig. 3. Elements of the user-interaction diagrams

Table 1. Mapping between UI elements and Metamodel elements

#	Class (MM)	Parameters
(a)	TerminalState	(Name = Vertex.name_V) and (Stereotype.name_S = "JTextField")
(b)	TerminalState	(Name = Vertex.name_V) and (Stereotype.name_S = "JList")
(c)	TerminalState	(Name = Vertex.name_V) and (Stereotype.name_S = "JLabel")
(d)	NonTerminalState	Name = Vertex.name_V
(e)	Pseudostate	kind_P = "PseudostateKind::initial"
(f)	Pseudostate	kind_P = "PseudostateKind::final"
(g)	Pseudostate	kind_P = "PseudostateKind::choice"
(h)	Pseudostate	(kind_P = "PseudostateKind::join") or (kind_P = "PseudostateKind::fork")
(i)	TS_Event	(Name = Transition.name_T) and (Steretype.name_S = "JButton")
(j)	TS_Input_Event	(Name = Transition.name_T) and (Steretype.name_S = "JButton") and (Condition.name_C = "input")
(k)	TS_Selected_Event	(Name = Transition.name_T) and (Steretype.name_S = "JButton") and (Condition.name_C = "selected")
(l)	TS_Condition_Event	(Name = Transition.name_T) and (Steretype.name_S = "JButton") and (Condition = Condition.value_C)
(m)	Input	Condition.name_C = "input"
(n)	Selected	Condition.name_C = "selected"
(o)	NTS_Event_A	buttonName = Transition.name_T
(p)	NTS_Event_B	(containerName = NTS_Event_B.tsName_NTSEB) and (buttonName = Transition.name_T)
(q)	NTS_Evemt_C	TextFieldName = NTS_Input.tsName_NTSI
(r)	NTS_Input	(containterType = Stereotype.name_S) and (containerName = NTS_Event_B.tsName_NTSEB) and (buttonName = Transition.name_T)
(s)	NTS_Selected	JListName = NTS_Selected.tsName_NTSS

Vertex. States and Pseudo-states are special cases of the class *Vertex* included in the metamodel.

Transitions Coming from Terminal States. They may connect either TS to TS or TS to NTS. Transitions in the metamodel are classified according to the kind of *event* they represent. Events can include *input events* and *conditions*. Conditions can represent either *previous user choices* (i.e. previous events) or *business/data logic* (in particular, UI component status).

In the running example, transitions can be labelled with a button name (stereotyped with `<<JButton>>`) (case (i) of Figure 3). The button represents an input event. But they can also be conditioned (cases (j), (k) and (l) of Figure 3). Transitions can be conditioned to a *previous user interaction* (cases (j) and (k) of Figure 3), or can be conditioned to *business/data logic* (case (l) of Figure 3). Finally, they can be conditioned to a previous user interaction but they are not associated to a input event (cases (m) and (n) of Figure 3). Boolean conditions about *business/data logic* are used, for instance, for checking the status of UI components. This is case of the user introduces a text into a *text field* (case (m) of Figure 3) or selects an element from a *list* (case (n) of Figure 3).

Transitions Coming From Non-Terminal States. They may connect either NTS to NTS or NTS to TS. They are transitions with boolean conditions about the user interaction in the non-terminal state (i.e. sub-diagram). There are five kinds of transitions. In the case of transitions of type (o), (p) and (r) of Figure 3 the boolean condition is related to the *"exit condition"* in a subdiagram. In other words, they check which button has been pressed as exit transition of the sub-diagram.

For instance, the terminal state *Query catalogue* of Figure 1 has two *"exit"* buttons, qualified as follows: `[Results.Exit]` and `[Searching criteria.Exit]`. They describe the action to be achieved when the user clicks the *"exit"* button and closes the `Query catalogue` window. However, the "exit" button can be pressed from two states: `Results` and `Searching criteria`. This is the reason boolean conditions are qualified in the main diagram. There are more cases in Figure 1 of *"exit"* conditions: `[Close]`, `[Cancel]`, `[Accept]` or `[Proceed]`. In these cases the *"containerName"* is not specified because there is only one button with this name in the sub-diagram.

The cases (q) and (s) of Figure 3 check in the main diagram the user interaction in the sub-diagram. They consider two cases: the introduction of a text into a *text field* (case (q)) and the selection from a list (case (s)).

For instance, the `[selected]` condition, in the transition triggered from the `Query catalogue` state of Figure 1, is not an *"exit condition"*, rather than it is an *"internal transition"* of the `UI_QueryCatalogue` diagram, which is checked from outside of the sub-diagram. This makes possible that `UI_ManageShoppingCart` controls the user interaction in the `UI_QueryCatalogue` window. The keyword `''selected''` is associated to the `JList` container. The keyword `''input''` is associated to the `JTextField` container. The transition `[selected] / <<JButton>> Remove article` of Figure 1, means that the button *"Remove article"* will be enabled/visible whenever the user selects one element of the container *"Selected articles"*. An input event can also represent that the mouse is placed over a *label* or that the mouse is focused on a *text field*. For this reason we need the following generalization.

Generalization. The `Generic` transition in the metamodel are transitions of type `[String]` —an abstraction of the *"Input"* (case (m) of Figure 3) and *"Selected"* (case (n) of Figure 3) transitions—. `TS_Generic_Event` transition in the metamodel are transitions of type `<<Name>> Name` (where *Name* is an string). This kind of transition is similar to `TS_Event` (case (i) of Figure 3, i.e.,

`<<JButton>> Name`). Finally, `TS_Guard_Generic_Event` transitions are transitions of type `[Guard] / <<Name>> Name` (where *Guard* and *Name* are strings). This kind of transition is similar to `TS_Input_Event` (case (j) of Figure 3), `TS_Selected_Event` (case (k) of Figure 3), and `TS_Condition_Event` (case (l) of Figure 3).

However, specific transitions, for instance, `<<JButton>> Name`, are used by the EMF/GMF implementation of the $\mathcal{ALIR}$ tool to draw an specific element of the toolbar, for instance, *"JButton"*.

3 Implementing an Eclipse GMF Tool for User-Interaction Diagrams

Now, we would like to present the implementation of an Eclipse GMF tool, called $\mathcal{ALIR}$, for the modeling of user-interaction diagrams. Such tool allows the designer to model user-interaction diagrams with the help of a toolbar in which the elements of user-interaction diagrams are available. In addition, the tool is able to check some of the constraints imposed to user-interaction diagrams.

Our Eclipse GMF tool can be integrated with our UI-MDD technique described in [2] as follows.

- The first step of our UI-MDD technique consists in the design of the layout of each window of the system. The windows have to include its UI components (i.e buttons, text fields, labels, etc). We have used the *Eclipse Window Builder-Pro Plug-in* for modelling the layout of the windows and for adding UI components. The developer can obtain prototypes of the windows by means of the code generation (step 0 of Figure 4).
- Secondly, the developer can store the layout in XMI format (step 1). The Eclipse GMF tool accepts XMI format as input and extracts (step 2 of Figure 4) the UI components (i.e., buttons, text fiels, labels, etc.), which will be used as elements in the tool (step 3 of Figure 4).
- Thirdly, the developer uses the Eclipse GMF tool to draw user-interaction diagrams.
- Next, the new models (i.e. user-interaction diagrams) are stored in XMI format (step 4 of Figure 4).
- Finally, the XMI file can be imported from the *Eclipse Window Builder-Pro Plug-in* (step 5 of Figure 4) to test the behaviour of the windows.

Both the metamodel discussed in Section 2 and the $\mathcal{ALIR}$ modeling tool presented next, have been developed using some of the MDD facilities provided by the *Eclipse* platform. This free and open-source environment provides the most widely used implementation of the *OMG standard Meta-Object Facility* (MOF) [10], called *Eclipse Modelling Framework* (EMF) [5]. Although EMF currently supports only a subset of MOF, called *Essential MOF* (EMOF), it allows designers to create, manipulate and store both models and metamodels using the OMG *XML Metadata Interchange* (XMI) [12] standard format. Many MDD-related initiatives are currently being developed around Eclipse and EMF. Among them, and directly related to our tool, it is worth mentioning the following ones:

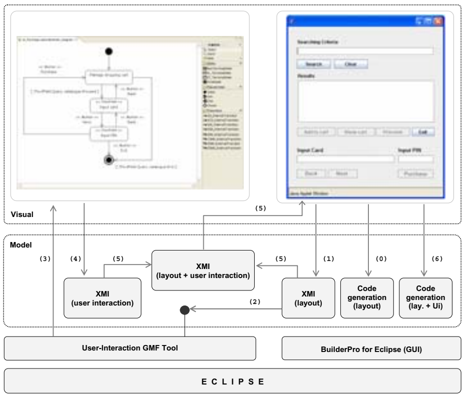
Visual
Model
(3)
(4)
(5)
(5)
(5)
(1)
(0)
(6)
(2)
XMI (layout + user interaction)
XMI (user interaction)
XMI (layout)
Code generation (layout)
Code generation (lay. + UI)
User-Interaction GMF Tool
BuilderPro for Eclipse (GUI)
ECLIPSE

Fig. 4. UI-MDD Technique Steps

- The *Eclipse Graphical Modelling Framework* (GMF) [4], which allows designers: (1) to create a graphical representation for each domain concept included in the metamodel, (2) to define a tool palette for creating and adding these graphical elements to their models (see Figure 5), and (3) to define a mapping between all the previous artifacts, i.e. the metamodel elements, their graphical representations, and the corresponding creation tools.
- The *Object Constraint Language (OCL)* [9] facilities provided by the *Eclipse Modeling Framework (EMF)*. This plug-in can be used, together with GMF, to enable the definition and evaluation of OCL queries and constraints in EMF-based models.

It is worth mentioning that both the metamodel and the Eclipse GMF tool implemented as part of this work, have been developed as an extension and modification of StateML+ [1]: a set of MDD tools aimed at designing hierarchical state-machine models (and automatically generating thread-safe Ada code from these models).

3.1 The $\mathcal{ALIR}$ Tool

Figure 5 shows an snapshot of the $\mathcal{ALIR}$ tool. It has been used for drawing the `UI_ShoppingCart` user-interaction diagram of Figure 1. Figure 6 shows the instance of the metamodel of Figure 2 in the case of the `UI_ShoppingCart` user-interaction diagram.

The $\mathcal{ALIR}$ tool is able to validate user-interaction diagrams according to some semantic and syntactic rules. More than 150 *OCL* constraints have been implemented to support the model validation. The most relevant ones are those whose aim is to check whether the boolean conditions about user interactions, that is, about "exit conditions" and "internal transitions", in a main diagram have an associated event in sub-diagrams. In Table 2 we show some of the constraints the $\mathcal{ALIR}$ tool is able to check. The reader can find more detailed information about the $\mathcal{ALIR}$ tool in our Web page: `http://indalog.ual.es/mdd/alir`.

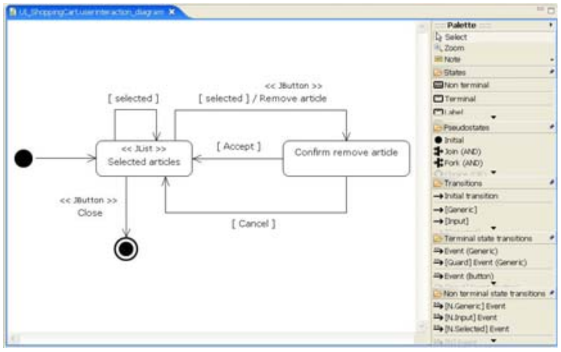

Fig. 5. Snapshot of the Eclipse GMF tool

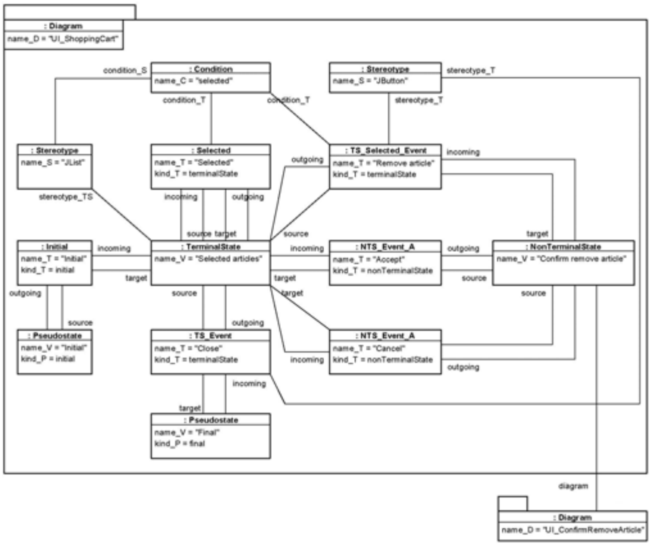

Fig. 6. Instance of the Metamodel for the `UI_ShoppingCart` diagram

4 Related Work

In the literature, there are several works about MDD and UI design, and most of them are supported by a tool. The most relevant ones are *TRIDENT*, *TERESA*, *WISDOM*, *UMLi*, *ONME* and *IDEAS*. TRIDENT (*Tools foR an Interactive Development ENvironmenT*) [3] proposes an environment for the development of highly interactive applications. User interfaces are generated in a (quasi-) automatic way. There is a CASE tool called SEGUIA (*System Export for Generating a User Interface Automatically*) [22] for generating the user interface from TRIDENT models. This approach can be considered a non-UML compliant proposal. However, most of the concepts developed in TRIDENT were later included

Table 2. Some OCL constraints of the $\mathcal{ALIR}$ tool

NTS_Event_C transition

```
self.source.oclAsType(NonTerminalState).diagram.vertex->
    one(v | (v.name_V=self.tsName_NTSEC) and
                 (v.oclAsType(TerminalState).stereotype_TS.name_S=self.tsStereotype_NTSEC) and
                 (v.outgoing->
                    one(t | (t.name_T=self.name_T) and
                              ((t.target.oclIsTypeOf(Pseudostate)) and
                              (t.target.oclAsType(Pseudostate).kind_P=PseudostateKind::final)))))
```

NTS_Input transition

```
self.source.oclAsType(NonTerminalState).diagram.vertex->
    one(v | (v.name_V=self.tsName_NTSI) and
                 (v.outgoing->
                    one(t | (t.condition_T.name_C=self.condition_T.name_C) and
                                ((t.target.oclIsTypeOf(Pseudostate)) and
                                (t.target.oclAsType(Pseudostate).kind_P=PseudostateKind::final)))))
```

Input transition

```
if (self.source.oclIsTypeOf(NonTerminalState)) then
    self.source.oclAsType(NonTerminalState).diagram.vertex->
        one(v | (v.outgoing->
            one(t | (t.condition_T.name_C=self.condition_T.name_C) and
                    ((t.target.oclIsTypeOf(Pseudostate)) and
                    (t.target.oclAsType(Pseudostate).kind_P=PseudostateKind::final)))))
else
    true
endif
```

in UML proposals. TERESA (*Transformation Environment for InteRactivE System RepresentAtions*) [21] is a tool aimed to generate user interfaces for multiple platforms from task models designed with the so-called *ConcurTaskTrees*. Partially based on UML, WISDOM (*Whitewater Interactive System Development with Object Models*) [7] is a proposal for user interface modeling. WISDOM uses UML but the authors have also adopted the ConcurTaskTrees. Fully based on UML, UMLi [20,19,18] proposes an extension of UML for user interface modeling. UMLi aims to preserve the semantics of existing UML constructors since its notation is built by using UML extension mechanisms. Like our technique, the scope of UMLi is restricted to WIMP interfaces. UMLi is supported by ARGOi [17]. UMLi is very similar to our proposal in the adoption of UML diagrams for user interface modeling, although it supports different extensions. Another case of MDD-based UI development is the OO-Method [14]. The authors have developed the ONME tool (*Oliva Nova Model Execution*, http://www.care-t.com), an implementation of the *OO-Method*, which complies with the MDA paradigm by defining models at different abstraction levels. Finally, IDEAS (*Interface Development Environment within OASIS*) [6,8] is also a UML-based technique for the specification of user interfaces based on UML and on the OASIS (*Open and Active Specification of Information Systems*) specification language [13].

5 Conclusions and Future Work

This paper presents the first step towards the development of a tool for user interface design based on MDD. User interaction diagrams are one the main elements of our technique, since they are used for dialogue modelling. However, our modelling technique requires also task and presentation modeling. The presentation model is partially handled by means of the UI design tool (like *Windows Builder-Pro of Eclipse*). However we have to integrate it with our Eclipse GMF tool by means of XMI. It is considered as future work. In addition, the presentation modelling in our proposal is also achieved by means of the so-called *UI-class diagrams*, which are an specialization of UML class diagram for UI components. An extension of the tool for supporting UI-class diagram modelling is planned to be implemented in the near future. Finally, task modelling is achieved in our UI-MDD technique by means of *user-interface* diagrams which are an specialization of use case diagrams for UI specification. We would also like to extend our tool for modeling such diagrams in order to be able to fully support in $\mathcal{ALIR}$ our UI-MDD technique.

Acknowledgements

This work has been partially supported by the EU (FEDER) and the Spanish MEC under grants TIN2008-06622-C03-03, TIN2006-15175-C05-02 (MEDWSA), and TIN2007-61497 (SOLERES).

References

1. Alonso, D., Vicente-Chicote, C., Pastor, J.A., Álvarez, B.: StateML+: From Graphical State Machine Models to Thread-Safe Ada Code. In: Kordon, F., Vardanega, T. (eds.) Ada-Europe 2008. LNCS, vol. 5026, pp. 158–170. Springer, Heidelberg (2008)
2. Almendros-Jiménez, J.M., Iribarne, L.: An Extension of UML for the Modeling of WIMP User Interfaces. Journal of Visual Languages and Computing 19, 695–720 (2008)
3. Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Sacré, I., Vanderdonckt, J.: Architecture Elements for Highly-Interactive Business-Oriented Applications. In: Bass, L.J., Unger, C., Gornostaev, J. (eds.) EWHCI 1993. LNCS, vol. 753, pp. 83–104. Springer, Heidelberg (1993)
4. Eclipse Graphical Modeling Framework (GMF), `http://www.eclipse.org/modeling/gmf/`
5. Eclipse Modeling Framework (EMF), `http://www.eclipse.org/modeling/emf/`
6. Lozano, M., Ramos, I., González, P.: User Interface Specification and Development. In: Proceedings of the IEEE 34th International Conference on Technology of Object-Oriented Languages and Systems, pp. 373–381. IEEE Computer Society Press, Washington (2000)
7. Nunes, N.J., Falcao e Cunha, J.: WISDOM - A UML Based Architecture for Interactive Systems. In: Palanque, P., Paternó, F. (eds.) DSV-IS 2000. LNCS, vol. 1946, pp. 191–205. Springer, Heidelberg (2001)
8. Molina, J., González, P., Lozano, M.: Developing 3D UIs using the IDEAS Tool: A case study. In: Human-Computer Interaction. Theory and Practice, pp. 1193–1197. Lawrence Erlbaum Associates, Mahwah (2003)

9. OMG Object Constraint Language (OCL) Specification, version 2.0, http://www.omg.org/technology/documents/formal/ocl.htm
10. OMG Meta-Object Facility, http://www.omg.org/mof/
11. OMG OMG Unified Modeling Language (OMG UML), Superstructure, V.2.1.2, http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF
12. OMG XML Metadata Interchange (XMI), http://www.omg.org/spec/XMI/
13. Pastor, O., Hayes, F., Bear, S.: OASIS: An Object-Oriented Specification Language. In: Loucopoulos, P. (ed.) CAiSE 1992. LNCS, vol. 593, pp. 348–363. Springer, Heidelberg (1992)
14. Pastor, O.: Generatig User Interfaces from Conceptual Models: A Model-Transformation based Approach. In: Chapter in Computer-Aided Design of User Interfaces, CADUI, pp. 1–14. Springer, Heidelberg (2007)
15. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer, Berlin (1999)
16. Selic, B.: UML 2: A model-driven development tool. IBM Systems Journal 45(3) (2006)
17. Paton, N.W., Pinheiro da Silva, P.: ARGOi, An Object-Oriented Design Tool based on UML. In: Tech. rep. (2007), http://trust.utep.edu/umli/software.html
18. Pinheiro da Silva, P., Paton, N.W.: User Interface Modelling with UML. In: Information Modelling and Knowledge Bases XII, pp. 203–217. IOS Press, Amsterdam (2000)
19. Pinheiro da Silva, P., Paton, N.W.: User Interface Modeling in UMLi. IEEE Software 20(4), 62–69 (2003)
20. Pinheiro da Silva, P.: Object Modelling of Interactive Systems: The UMLi Approach, Ph.D. thesis, University of Manchester (2002)
21. Berti, S., Correani, F., Mori, G., Paternó, F., Santoro, C.: TERESA: A Transformation- based Environment for Designing and Developing Multi-device Interfaces. In: Proceedings of ACM CHI 2004 Conference on Human Factors in Computing Systems, vol. II, pp. 793–794. ACM Press, NY (2004)
22. SEGUIA, System Expert Generating User Interfaces Automatically, http://www.isys.ucl.ac.be/bchi/research/seguia.htm