López Jaquero, V. , Montero, F. Comprehensive Task And Dialog Modelling. HCI International 2007 (HCII 2005). 22-27 July 2007, Beijing, China. In Human Computer Interaction, part I, HCII 2007. J. Jacko (Ed.) LNCS 4550, Springer-Verlag, pp. 1149-1158, Germany, 2007. ISSN: 0302-9743

Comprehensive Task and Dialog Modelling

Víctor López-Jaquero, Francisco Montero

Laboratory on User Interaction & Software Engineering (LoUISE)
University of Castilla-La Mancha, 02071 Albacete, Spain
{ victor | fmontero }@dsi.uclm.es

Abstract. Task modelling has proven useful as a basis for user interfaces (IU) design. Although different models have been pushed ConcurTaskTrees (CTT) notation has become without any doubt the most extended notation for task model specification. However, this notation suffers from a lack of modularity, making the creation and modification of real-world applications a cumbersome process. In this paper a notation that takes inspiration from CTT is described that allows for the specification of the tasks the user is supposed to perform through the user interface and the dialog between the user and the user interface in an intuitive manner. Furthermore, the notation makes use of an abstract operation set to help in the automatic or semi-automatic generation of a user interface that conforms with the specified model.

Keywords: User interface design, abstract user interfaces, task models, dialog models.

1 Introduction

The design of user interfaces has become a discipline of capital importance for most software enterprises. Usable user interfaces leverage user's satisfaction within an application, and therefore leverage the potential success of any application.

UI development task is one of the main design challenges in the creation of an application, since it must support the system's acceptance and be accessible and usable for everyone. Involving the users from the very beginning in the design process and focusing on usability, and not just on technology, designers have tried to address this difficult challenge.

Although different user interface design approaches have been used [13] throughout the short, but intense, graphical user interfaces history, model-based approach [17] is becoming the approach receiving a wider attention from both user interfaces research community and industry during the last decade. This growing interest in model-based development is not limited to user interface design, but it is also used in the development of the whole application, as proposed in MDA (Model-Driven Architecture) proposal.

Model-based approaches take as input a requirements specification that is converted into different declarative models. The most widely used ones are the task, the user, the domain, the dialogue and the presentation models, although currently there is no standard describing which models should be used. These declarative models are used to generate automatically or semi-automatically a user interface compliant with the requirements captured in these models. The way the

transformation from a set of declarative models into a running user interface is achieved can be made following different approaches. Nevertheless, the most widely used approaches take as the cornerstone of their design process either a task model [18][15], a domain model [1] or both [11].

In this paper a visual notation for a task model specification is introduced that is used within a model-based approach: AB-UIDE [11] as the main model guiding the whole model-based design method proposed. This notation takes inspiration from ConcurTaskTrees notation [15], introducing a greater modularity and including dialog modelling. This visual notation has been designed in a fashion close to UML statecharts notation [7], to make easier for the huge mass of UML practioners to get into model-based user interface development.

2 From Domain Model To User Interface Generation

Domain model encapsulates the important entities of a particular application domain together with their attributes, methods and relationships. Within the scope of UI development, it describes the objects that the user requires in order to carry out his tasks.

Most applications rely on a database to perform its objectives. This data dependency inspired the creation of some projects aimed at generating automatically a user interface out of the data it was supposed to handle. Examples of such projects were Janus [1] or Teallach [6].

Although these domain-based approaches are useful to quickly generate a user interface to access some data, the usability of the resulting user interface is rather low.

These domain-based user interface generation approaches produce complex user interfaces, because users can see many elements at the same time. Moreover, as long as the user-tasks are not contemplated the dialog within the user interface is rather limited and constrained, producing user interfaces quite static. Another drawback in domain-based user interface generation approaches is the lack of a proper grouping between the elements the user requires to perform a task, reducing the productivity of the application.

Next, we elaborate on the task-driven model-based approach we use, and the arguments that directed us towards this solution.

3 From Task Model To User Interface Generation

Task model describes those tasks the user is allowed to perform through the user interface. This task model can be modelled in many different ways, some of them coming from Software Engineering community, such as UML statecharts, activity or use cases diagrams [16] or Petri nets [2], and some of them more specific to human-computer interaction community, such as CTT or User Action Notation (UAN) [8].

The derivation of a user interface out of a task model adds an additional view to the design process: the user. Thus, taking into account which tasks the user is allowed to perform and the temporal relationships between those tasks it is possible to increase the overall system usability for example by grouping the related widgets or by hiding all or most of the irrelevant information for the current task.

Therefore, relying on a task model for user interface generation rather than just on a domain model is a important step forward to improve the usability of the user interfaces built by applying model-based techniques.

3.1 Task & Domain: A Marriage of Convenience

A task model by itself is not enough to generate a high quality user interface, additional information is required. Although the task model includes information regarding which are the tasks the user is supposed to carry out with the application, it does not include information regarding the data that those tasks require to be performed.

Thus, we find it is necessary to relate the tasks and those data they require. Therefore, a marriage between task and domain model is required in order to generate a good user interface. For instance, if the user is supposed to perform an input task, where the data type for the input data is integer, the generation process should generate a set of widgets, which are appropriate for that kind of task and for that data type.

Another fact that the generation process within a model-based approach relying on task and domain models is the cardinality of the domain object the tasks are related to. For instance, consider the user asks the system to show the phone numbers for a client. Obviously, in this case the cardinality between the output task "Show phone numbers" and the method of the domain object returning those phone numbers is one-to-many (1,*). Therefore, the generation process should generate a set of widgets able to show a set of data entries (for instance a list box). In some approaches this last situation is modelled by specifying a single output task with one-to-one cardinality, and marking that task as repetitive, but some tweaking is required to make it work properly.

3.2 Dialog Modelling

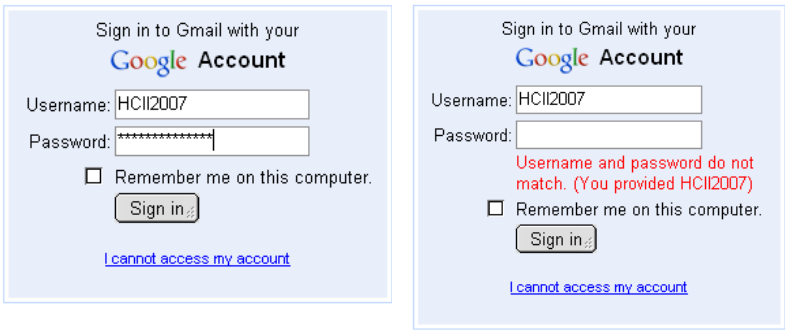
By relating task model with domain model we get some more valuable extra information for the generation of the user interface. However, for us it is not enough.

For instance, if a task model is created in CTT it is possible to describe the tasks the user will be allowed to perform and the temporal constraints between those tasks. However, it is not that easy to describe the dialog between the system and the user, that it to say, describe the situation where different branches are available depending on the actions the user take.

In figure 1 an example is shown where a user is authenticating in a webmail (i.e. GMail[1] application by providing a username and a password). The CTT notation is powerful and useful to analyse the authentication task and describe which data the user should provide and how he interacts with the system. Nevertheless, it is not flexible enough to embrace the interaction in all its dimensions. For instance, if the designer wants to specify a task modelling describing the behaviour exhibited in

[1] http://gmail.google.com

figure 1, it is really hard to specify all those situations arising from user's interaction. In the example in the figure, it is hard to specify the possible error states that a login task can produce (inform the user about a wrong password, a wrong username or both).



(a) User login in a webmail application.

(b) The user enters a wrong password or username.

Fig. 1. Authentication task in a webmail application.

This kind of exceptional situations shown in this tiny example, that have a direct impact in the usability of a software product, are some of the limitations of current task models that we overcome in our method.

Following, the description of our technique for task and dialog modelling will be shown.

4 A Comprehensible Task And Dialog Modeling Approach

Because of the facts enumerated in the previous section, a task model has been devised that takes inspiration from ConcurTaskTrees [15], the well-known technique within human-computer interaction community, UML statecharts diagrams [7], and the Canonical Abstract Prototypes [5].

Next, the most prominent features of the task-modelling approach introduced will be described in-depth.

4.1 Closer to UML

In this approach, we bring closer to the huge mass of UML practioners human computer interaction task modelling techniques by adopting a notation much alike UML statecharts diagrams. A task or subtask model specification begins with a circle whose background colour is black (starting state). On the other hand, the end of specification is indicated by a circle whose background colour is black and surrounded by another circle (see figure 2), as it is used in UML statecharts diagrams (final state).

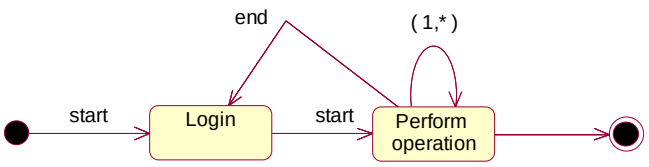
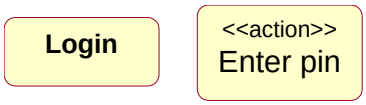


Fig. 2. Tiny example for a task model specification.

Each state in the diagram can be either a task or an action. Tasks can be further refined, while actions are elemental tasks that can not be further refined. Tasks are represented by using the same symbol that is used for a state in statecharts diagrams (see figure 2a). To represent actions the states have been stereotyped. The stereotype used has been called “action” (see figure 2b). All these tasks and actions have a set of properties to better describe de purpose and presentation of the intended goal.

Tasks, actions, starting state and final state are linked by transitions. A transition from a state *S* to another state *R* means that the task control flow can go from state *S* to state *R* if the condition given in the label of the transition is met (sequentially). The available labels for the transitions are detailed later on in the paper.



(a) Task *(b) Action*

Fig. 3. Representation of tasks and actions.

4.2 Modularity

One of the most interesting things in modelling any complex system is modularity. For instance, in UML the designer can split the design in packages to better organize the structure and readability of the models. Moreover, it also allows the designer to create a complex model where some parts are underspecified while some other parts are fully specified.

In our approach modularity has been taken into account also. First, the designer can design the general tasks structure (in our method this structure is derived out of a previous enriched use case model, capturing the initial requirements). The tasks created for that general structure are then refined either with new tasks or with the actions that will allow the user to carry out those tasks.

In figure 4 an example of this kind of modularity is depicted. The task *Login* (it represents the task of a login in a bank ATM) has been decomposed into two actions: *Enter login/card* and *Enter Pin*. Notice the resulting actions have also a starting and a final state that represent the beginning and the end of *Login* task. The designer is allowed to edit/refine a task by double-clicking on the task to be edited. By right-clicking the properties of either a task or an action can be edited.

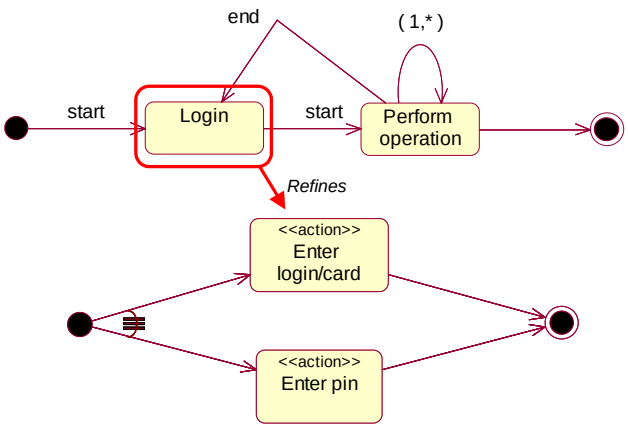


Fig. 4. Refining a task.

4.3 LOTOS-Based Temporal Relationships

This notation has been enriched by including the same Lotos [9] operators used within CTT. The graphical notation for these operators has been adapted to make it more easily usable within the statecharts notation used. For instance, in figure 2 the *repeat* operator is represented as a transition from a state to the same state. Between the parentheses the designer can specify the minimum and maximum number of times that this task can be repeated.

By default, a transition between two states means a sequential temporal relationship between those two states. Notice in figure 4 *Enter login/card* and *Enter pin* have a concurrent temporal relationship. Therefore, both actions can be executed concurrently in no particular order. The graphical representation used to represent concurrency between two tasks is the same one used in CTT.

4.4 Detailed Dialog Modelling With Abstract Tools

Constantine [4][5] proposed a set of abstract tools that represent, in an abstract manner, the complete set of actions that can be performed in an user interface. This set of actions was devised after year experimenting and gathering feedback from developers. Constantine uses this set of abstract tools to represent canonical user interfaces (abstract user interfaces).

In our approach these abstract tools are applied to express the dialog between the tasks and the actions. That is to say, which action from the user or the system are required to take a transition in our task model from one state to another. A state can have several outgoing transitions. Thus, the task flow control will choose the right transition according to the actions taken by the user or the system.

For instance, figure 5 models a scenario where a bank client wants to make a deposit. The client first enters the amount to deposit and puts the money on the ATM

slot (waiting for the ATM machine to acknowledge that the client puts the right amount of money on the slot is expressed as a post-condition for *Enter amount* action that needs to be held before *Enter amount* action is considered to be finished). If the post-condition is not held, an error is raised and the transition for error abstract action would be taken. Otherwise, the system would take the transition labelled *start*. For this transition to be taken it is required that the user enters the amount to deposit and to confirm/accept it.

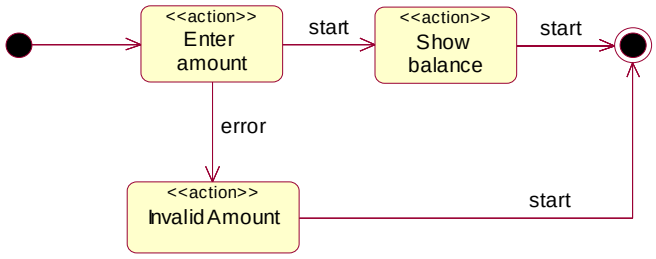


Fig. 5. Example of dialog modelling.

In short, by adding the abstract tools to the specification of our task model we provide the designer with a powerful tool to make dialog modelling easier and intuitive. Moreover, it also make generation/transformation process much more easier, by providing additional meaningful information regarding the transitions from one task to another and the type of operation required from the user in order to force each transition.

4.5 Tasks and Actions Properties

Tasks and actions need to be described in order to provide meaningful information for the generation process. In our approach a set of predefined properties has been defined (see figure 6), although it can be extended by the designer to add custom properties. A priori it is almost impossible to cover every potential property a designer might need to apply the transformation process leading to the generation of the final user interface, since different sets of properties are required to apply different heuristics or transformational approaches.

Each task or action has a descriptive name, a description in natural language that describes which is the goal of the task or action, a type that specifies whether it is abstract, for input, for output, etc. They also have a frequency attribute that stores the task/action frequency the designer expects that task/action will be executed by the user/system. This attribute is quite useful to help in finding out a good task layout in the final user interface generated. Precondition attribute includes a set of expressions that must be evaluated to true for the task to be enabled. The expressions are evaluated as a logical program. It means that the precondition will failed whenever any of the expressions that it includes is not successfully evaluated. Postconditions work in a similar manner. Nevertheless, in this case all the expression should be evaluated to true before a transition to other state is taken. Resulting from either evaluation the precondition or the postcondition, error exceptions can be raised. These

error exceptions can be handled within the dialog modelling by means of *error* abstract tool. For the specification of both precondition and postcondition attributes OCL [19] (Object Constraint language is used. OCL is widely used among UML practioners to express constraints in a variety of UML diagrams. Finally, any task can be represented at the abstract level either in a *FreeContainer* or in a *Container* [16]. The difference between both kinds of *Containers* is that the first one is a root container that cannot be included within any other container. Designers can choose to leave black this attribute, and to postpone this decision until generation process.

Task name	Login
Description	The bank customer enters the card or the login.
Type	Abstract
Frecuency	*High*
Precondition	NULL
Postcondition	Customer.checkLogin()
Presentation	FreeContainer

Action name	EnterLogin_card
Description	The user types in the login or enters the card.
Type	*Input*
Frequency	*High*
Precondition	NULL
Postcondition	*Customer.currentLogin!=“”*

Action name	EnterPin
Description	The user types in the password for the login.
Type	*Input*
Frequency	*High*
Precondition	*Customer.currentLogin!=“”*
Postcondition	*Customer.currentPassword!=“”*

Fig. 6. Properties for *Login* task and its associated actions.

Notice the expressions used in either precondition or postcondition attributes can included any valid OCL expression. In the expressions created the designer can also make use of any public method or attribute of the classes defined for the domain model.

5 Conclusions

Task modelling has become the cornerstone for model-based user interface design. Different approaches have been pushed, but CTT seems to be the most widely used. Nevertheless, CTT is not as used as it should, because of a lack of tools integrating that task modelling technique within the whole development process and because is

quite far from what most developer are used to when modelling their applications: UML.

In this paper a task modelling approach is introduced that takes inspiration from the strong points of CTT to create a graphical notation alike UML statecharts diagrams to bring closer to the huge mass of UML practioners HCI community modelling techniques. Moreover, the notation has been enriched with abstract tools to provide an easy and clear notation for the dialog between the system and the user.

Although, model-based user interface design approaches have reached some kind of maturity, they are not as much used as they should by developers. To build a bridge between HCI research community and developers we need to devise notations able to attract developers towards the good practices for user interface design. In this paper, we have tried to make another step forward to build the bridge between both communities.

6 Acknowledgments

This work is partly supported by the Spanish PAI06-0093-8836, CICYT TIN2004-08000-C03-01 and PCC05-005-1 grants.

7 References

1. Balzert, H., Hofmann, F., Kruschinski, V., Niemann, C. The JANUS Application Development Environment - Generating More than the User Interface. CADUI 1996: 183-208
2. Bastide, R., Palanque, P.A. Implementation Techniques for Petri Net Based Specifications of Human-Computer Dialogues. CADUI 1996: 285-302, 1996.
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15,3 (2003) 289–308
4. Constantine, L. L., Lockwood, L. A. D. Software for use. Addison-Wesley. 1999.
5. Constantine, L.: Canonical Abstract Prototypes for abstract visual and interaction design. Proceedings of DSVIS' 2003, 10th International Conference on Design, Specification and Verification of Interactive Systems. LNCS, Vol. 2844. Springer-Verlag, Berlin, 2003.
6. Griffiths, T., Barclay, P., McKirdy, J., Paton, N., Gray, P., Kennedy, J., Cooper, R., Goble, C., West, A., Smyth, M. Teallach: A model-based user interface development environment for object databases. In Proceedings of UIDIS'99. IEEE Press. 86-96.
7. Harel, D. Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8:231--274, 1987.
8. Hartson, R., Gray, P., Temporal Aspects of Tasks in User Action Notation. In Human Computer Interaction, vol. 7, pp1-45, 1992.
9. Information Process Systems - Open Systems Interconnection - LOTOS - A Formal Description Based on Temporal Ordering of Observational Behaviour. ISO/IS 8807. 1988.
10. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López Jaquero, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). LNCS, Vol. 3425, Springer-Verlag, 2005.

11. López Jaquero, V., Montero, F., Molina, J.P., González, P. Fernández Caballero, A. A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties. Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). LNCS, Vol. 3425, Springer-Verlag, 2005.
12. Montero, F., López Jaquero, V., Vanderdonckt, J., González, P., Lozano, M.D., Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'2005), Newcastle upon Tyne, England, July 13-15, 2005. Springer-Verlag, 2005
13. Myers, B., Hudson, S. E., and Pausch, R. Past, present, and future of user interface software tools. ACM Trans. Comput.-Hum. Interact. 7, 1 (Mar. 2000), 3-28, 2000.
14. Oeschger, I., Murphy, E., King, B., Collins, P., Boswell, D.. Creating Applications With Mozilla. O'Reilly, September, 2002.
15. Paternò, F., Mancini and Meniconi, S. "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models". Interact' 97, Chapman & Hall, July 1997. pp.362-369.
16. Pinheiro da Silva, P. Object Modelling of Interactive Systems: The UMLi Approach. Ph.D Thesis. University of Manchester, N.W. Paton (supervisor), UK, 2002.
17. Puerta, A.R. A Model-Based Interface Development Environment. IEEE Software, pp. 40-47, 1997.
18. Reichart, D., Forbrig, P., and Dittmar, A. 2004. Task models as basis for requirements engineering and software execution. In Proceedings of the 3rd Annual Conference on Task Models and Diagrams. TAMODIA '04, vol. 86. ACM Press, New York, NY, 51-58.
19. Warmer, J., Kleppe, A. The Object Constraint Language: Precise Modeling with UML. Object Technology Series. Addison-Wesley. 1999.