TOWARDS A DESIGN BASED ON THE USER TASK OF HUMAN INTERFACES IN COCKPIT

Dimitri Tabary*, Mourad Abed*, Adel Mafhoudhi§

* Université de Valenciennes et du Hainaut-Cambrésis - Le Mont Houy
F-59313 Valenciennes cedex 9 - FRANCE
{dimitri.Tabary; mourad.abed}@univ-valenciennes.fr
Tel: +33 (0) 327.51.14.61 Fax: +33 (0) 327.51.13.16

§ Département d'informatique, faculté des Sciences de Sfax
Rte Soukra km 3,5 BP 802 -- 3018 Sfax -TUNISIE
adel.mahfoudhi@fss.rnu.tn
Tél : +(216) 4.27.43.90 Fax : +(216) 4.27.44.37

Abstract. This paper is intended to present a approach to the construction of a task model of method, named TOOD (Task Object Oriented Design), used for the development of an interactive system. This approach is based on a formal notation, which gives quantitative results which may be checked by designers and which provide the possibility of performing mathematical verifications on the models. The modelling formalism is based on the joint use of the object approach and of high level Petri nets. The concepts borrowed from the object approach make it possible to describe the static aspect of tasks and the Petri nets enable the description of dynamics and behaviour. We also describe a software aid tool for the manipulation of these models, which allow the edition and the simulation of a task model. In order to facilitate comprehension of the method, an extremely simple example of procedure used in missile firing management will be given.

3 Introduction

Several research projects have been dedicated to the modelling of user tasks in the field of interactive system design (see, for example, the work concentrating on the following methods: MAD [1], DIANE [2], GOMS [3], TKS [4], Action Theory [5]) However, their actual use is far from being a widespread practice. One of the possible reasons for this is that they do not use truly formal methods, which make it possible to provide the task models with concision, coherence and non-ambiguity [6]. What is more, these projects suffer from their lack of integration into a global design process covering the entire lifecycle of the HCI and also from the lack of modelling support software. In order to overcome these problems, current research projects are oriented towards a methodological framework which covers from the first activity analysis stage up to the stage of the detailed specification of the HCI: The methods MAD* [8], DIANE+ [9], GLADIS ++ [7], ADEPT [10], TRIDENT [11] [12] go in this direction. These design methodologies are based on several models (task model, user model, interface model) and are aided by tools for the implementation of these models.
Our research work falls into this category, whilst placing the emphasis on the formal aspects of model representation and their transformation throughout the stages of the design process. The TOOD method is based on the representation that the user has of the task, apart from considerations of computer processing. It uses the object approach and the object Petri nets to describe, on the one hand, the functional aspects and the dynamics of the user tasks, and on the other hand the behavioural aspects of the HCI and of the user in order to specify how the tasks are performed. Its formalism aims at covering the entire development cycle from the analysis of what exists, up to the detailed design and implementation.
In this paper, we will restrict ourselves to presenting just the stage of task modelling using TOOD; the reader will find a more detailed description of the method in [13]. This description is illustrated by an example concerning a missile firing management task. Explanations on supporting software for TOOD are also provided.

4 TOOD Development Cycle

The TOOD design process can be divided into four major stages, Figure 1:

- The analysis of the existent system and of the need is based on its user's activity and it forms the entry point and the basis for any new designs.
- The Functional Task Model (FTM) concerns the description of the user tasks of the system to be designed. It makes it possible to describe the user task in a coherent and complete way. Two

models are created at this level: the Static Functional Task Model (SFTM) and the Dynamic Functional Task Model (DFTM), in order to be able to use it for the HCI specification.

- The Operational Model (OM) makes it possible to specify the HCI objects in a Local Interface Model (LIM), as well as the user procedures in a User Model (UM) of the system to be designed. It uses the needs and the characteristics of the functional task model in order to result in an Abstract Interface Model (AIM) which is compatible with the user's objectives and procedures.
- The Creation of the HCI concerns the computer implementation of the specifications resulting from the previous stage, supported by the multi-agent software architecture defined in the Interface Implementation Model (IIM).

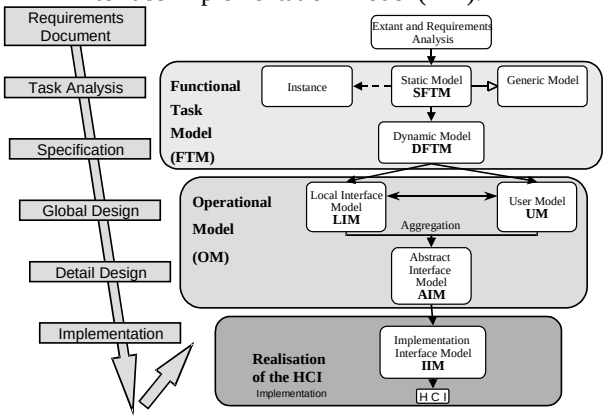


Figure 1: TOOD Method Development Cycle

The TOOD method is supported by an editor developed in Visual C++. It makes model capture and syntactic checking easier. Moreover, it supports the test and simulation activities of the dynamic task model. Examples of screen pages are given later by illustrating them with models, which come from a description of tasks relating to missile firing management. The analysis of what exists and of firing management needs show that the pilot of a bomber must be able to select a bomb, a point (PE) at which the bomb is situated on the aircraft and finally the explosion mode for this bomb (rocket). He must also choose the operational options for the execution of his fire. There is no preferential order for the performance of tasks.

5 Functional Task Model (FTM)

From the analysis of what exists and of the need, the main aim of the FTM is to establish a coherent and complete description of the "future" user task, firstly in a *functional* form (Static Functional Task Model, SFTM), and also in the *dynamic* form (Dynamic Functional Task Model, DFTM) in order to use it for the conceptual specification of the interface.

This model, like MAD [1], is designed as a means to take the user and his task into account as early as possible in the cycle. The aim is to provide the development teams with a methodological tool, which will allow them to isolate the user task information necessary for the formal design of interfaces in order to allow a more natural integration into the logical development cycle. The construction of the functional model is based on four iterative stages:

1. Hierarchical breakdown of the tasks.
2. Identification of the describer objects.
3. Definition of the dynamics of the elementary and control tasks.
4. Integration of task interruptions.

1.1 Static Functional Task Model (SFTM)

The Functional model enables the breakdown of the user's stipulated work with the interactive system into significant elements, called tasks. Each task is considered as being an *autonomous* entity corresponding to a goal or to a sub-goal, which can be situated at various hierarchical levels. This goal remains unchanged in the various work situations. In order to perfect this definition, TOOD formalises the concept of tasks using an object representation model, in which the task can be seen as an *Object*, an instance of the *Task Class*. This representation consequently attempts to model the task class by a generic structure of coherent and robust data, making it possible to describe and organise the information necessary for the identification and performance of each task.

Two types of graphic and textual document, as is shown in Figure 2 define each task class.

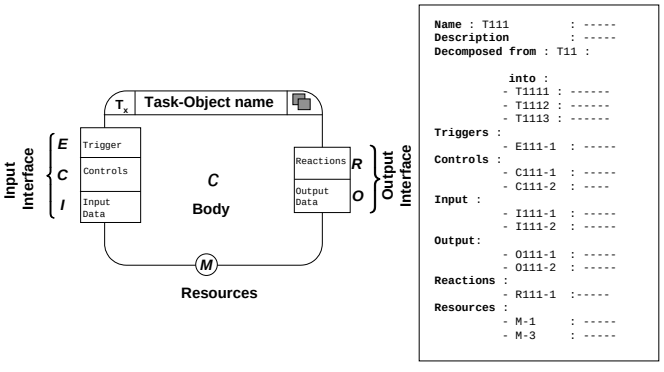


Figure 2: Graphic and textual document of a task class

The task class is studied as an entity formed using four different *describers*: the Input Interface, the Output Interface, the Resources and the Body. We also associate a certain number of *identifiers* to these describers, which

make it possible to distinguish the Task Class amongst the others: Name, Goal, Index, Type and Hierarchy, Table 1:. This parallel with software engineering guarantees a strong link between a user-centred specification based on ergonomic models and the software design based on the object model.

Attributes	Description
Name	Action verb followed by a complement (object treated by the task), reflecting the treatment to be performed by the task. It is preferable for the name to include vocabulary used by the users in order to respect the terminology during the development of the interface.
Goal	Explanation in natural language of the goal which the user or application wishes to reach via the task.
Index	Formal identifier of the task formed using the number of the master task, to which the sequential number corresponding to the said task is added.
Type	Nature of the task, it designates its category: human, automatic or interactive.
Hierarchy	Number of task classes composing it; it is represented by a series of small squares.
Triggers	Events which bring about the performance of the task. They are classed into two categories : Formal or explicit trigger events, which correspond to external triggers. They appear in an observable way in the work environment (information on screen, press on a button, communication, …). The tasks triggered by this type of event are considered, as being compulsory that is their performance is vital. Informal or implicit trigger events, which correspond to triggers, brought about following a user decision, from a set of information characterising its work situation. Unlike the formal events, they are not visible to an outside observer, but may be expressed verbally.
Contextual conditions	Information which must be checked during the performance of the task. These conditions affect the way in which the task is performed.
Input data	Information necessary during the performance of the task.
Reactions	Results produced by the performance of the task. Their content indicates the following type of modification : – Physical and, in this case, it indicates the modification of the environment (applicative call, change of state, …). – Mental, indicating the modification or a new representation of the situation by the user. The Reactions thus determine whether the aims are attained or not and, in such a case, the task will be repeated after a possible development of the situation.
Output data	Data transformed or created by the performance of the task.
Resources	Human users and/or interaction system entities involved in the performance of the task
Body	Central unit of the task class. For intermediate or hierarchical tasks, it gives the task procedure diagram, that is to say the logical and temporal relations of the sub-tasks. These relations reflect, in a certain way, the user's work organisation. On the other hand, for terminal tasks, it defines the action procedures for the HCI/user couple. The specification for these procedures is produced in the task operational model.

Table 1: Task class identifiers

The resources, and the information from the input and output interfaces are modelled by objects, called "describer objects", instances of describer classes. These objects, from a computing point of view, represent the components of a task class (Figure 3), whereas from a user point of view, they constitute the mental image of the entities manipulated in a task. They will thus have a final image in the interactive system.

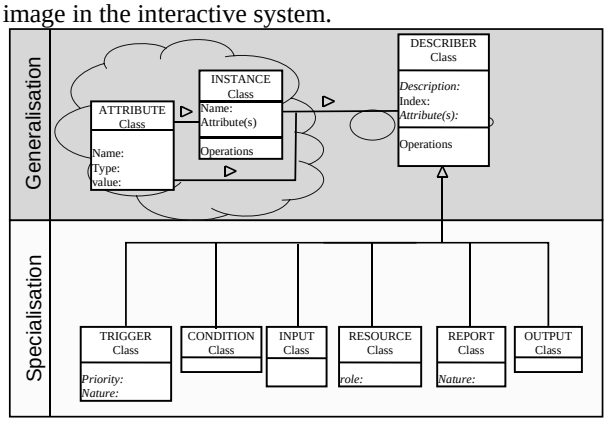


Figure 3 : Describer object Class hierarchy

These describer classes are defined by:

- *Name*: identification of the class. Each specialised class bears a name, which defines the nature of its specialisation.
- *Index*: formally references the object.
- *Description*: Explanation of the object's role in natural language.
- *Attribute*: indicates the characteristics, which one wishes to model with reference to objects in the real world.

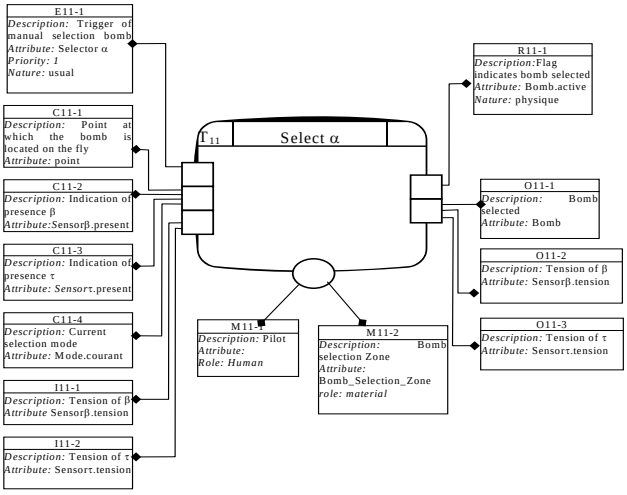


Figure 4: Identification of the Describer objects

Figure 4 gives the identification of the describer objects in the task class "Select a bomb α". In this example, the trigger class E11-1 characterises a manual bomb selection event. A real object, "Selector α" of boolean type, selector on/off corresponds to the attribute of this class.

5.2 Hierarchical breakdown

The construction of the task model structure is guided by user aims. This construction is based on the current tasks, which are translated and organised progressively into a new task, which reflects another way of working. This imagined manner could bring new tasks to light, a new functional distribution and new user roles. A graphic editor, Figure 5, facilitates these stages in the elaboration of task diagrams. The editor is divided into three zones: Menu bar, command buttons, and the task edition zone. Both the menu bar and the command buttons are divided into five groups: saving functions, edition functions (cut, copy, stick), help functions, FTM editing functions and finally the operational model construction functions. To start, all of the tasks identified are entered. Then, we define the inter-task links, which express the information flow between the tasks, which follow on from one another. For each task, we make an inventory of all the describer objects used for each task. First, the describer objects for the master task are entered, progressing towards the sheet tasks, and thus creating a database at the same time. The breakdown is presented firstly during the specification of the describer objects for each task (window, specification, specified task thumbnail, Figure 5), and then in diagrams which make it possible to define inter-task links (edition zone, Figure 5). In this way, it constitutes the static model (SFTM).

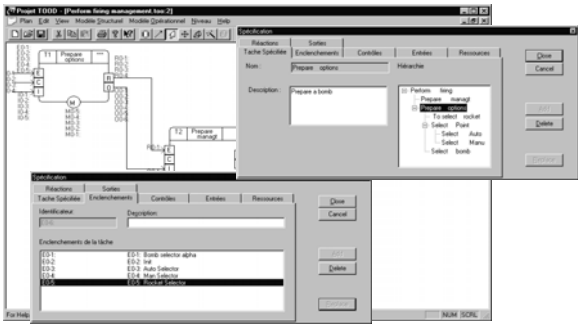


Figure 5: Task editor.

In our example of missile firing management, the task graph is made up of a root task, "Perform firing management", broken down into two parallel sub-tasks, "Prepare firing options" and "Prepare firing management options", associated to the user's aims. The task "Select a point" is made up of two elementary tasks, "Select point automatically" and "Select point manually", which correspond to two distinct user strategies for choosing the point. In this example, the task class "Prepare firing options" may be activated by five trigger events which all correspond to events in the master control task T0. The trigger class E1-1 characterises an event of manual bomb selection.

1.2 Dynamic Functional Task Model (DFTM)

The dynamic Functional task model (DFTM) aims at integrating the temporal dimension (sequencing, synchronisation, concurrency, and interruption) by completing the static model. The dynamic behaviour of tasks is defined by a control structure, called TCS (Task Control Structure), based on an object Petri net (RPO) [14], Figure 6. It is merely the transformation of the static structure. This TCS describes the consumption of the input interface's describer objects, the task activity, the release of describer objects from the output interface as well as the resource occupation.

Each TCS has an input transition t1 and an output transition t2 made up of a selection part and an action part. The functions associated to each transition allow the selection of objects and define their distribution in relation to the task activity.

The selection part of transition t1 is made up of three functions: δ, β, χ

- **Priority function** δ makes it possible to select the highest priority trigger for the task. This function is at the basis of the interruption system. It allows the initiation of a task performance, even if another lower priority task is being carried out. However, the performance of the task in relation to this trigger remains subject to the verification of the completeness and coherence functions.

- **Completeness function β** checks the presence of all the describer objects relating to an observed event, that is to say the input data, the control data and the resources used to activate the task class in relation to a given trigger event.
- **Coherence function χ** assesses the admissibility of these describers in relation to the conditions envisaged for the task. As in [8], this function is a set of verification rules which use simple logical or mathematical type operators and which obey a unique syntax making their formulation possible.

The selection part of transition t2 has a **completeness function ρ** which checks the presence of output data and resources associated to the reactions released by the body of the task.

The hierarchical tasks are considered to be **control tasks** for the tasks, which compose them. Consequently, the action parts of the input and output transitions of their TCS possess respectively an emission function **ϕ** and a synchronisation function **σ**. Function **ϕ** defines the **emission rules** (constructors of the input transition) for transition t1, for the activation of the sub-tasks, as well as the distribution of data consumed by these sub-tasks. Function **σ** defines the **synchronisation rules** (constructors of the output transition) for the sub-tasks. These rules are defined in Table 2.

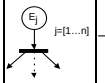
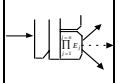
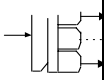
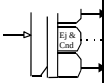
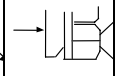
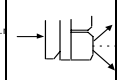
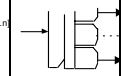
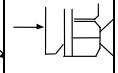
	Constructor	Symbol	Transition	Order of priority	Sharing of resource	Description
Input Transition	Junction et Distribution (simultaneity)			Cst	No	n tasks are performed at the same time by m different resources. The same trigger or else by different triggers these tasks.
	Transfer (Or)			Yes	Yes	n tasks are performed in order of trigger priority. The tasks share data and resources. These tasks can be interrupted
	Transfer with condition			Yes	-	n tasks are performed in order of trigger priority which will satisfy certain conditions. The tasks share data and resources. These tasks can be interrupted.
Output Transition	Transfer alternative			Yes	-	One single task is triggered. The triggers are similar, but only one is taken according to the context.
	Synchronisation			-	-	n sub-tasks must be finished so that the management task may be finished. The management task releases either Rj reactions or new reactions.
	Or					The management task is finished when at least one of these sub-tasks is finished.
	Alternative			-	-	The management task is finished when only one of its "daughter" tasks is finished.

Table 2: Constructors of the input transition and Constructors of the output transition.

Figure 6 presents the TCS of the "Prepare the firing options" control task. For transition t1, the priority function is δ = (E1-1, E1-2, E1-3, E1-4, E1-5). The completeness function for the trigger E1-1 : "Bomb selector α" is :β(E1-1) = <C1-1, C1-2, C1-3, C1-4, I1-1, I1-2, M1-1, M1-2>. The coherence rule χ(E1-3) = (I1-1 = AUTO) specifies the constraint, which the management system must have initially: automatic mode to change to manual point selection. Task T1 can finish in relation to R1-1:"bomb selected". This result must be accompanied by output data and resources specified by ρ(R1-1) =<O1-1, O1-2, O1-3, M1-1, M1-2> Finally, the emission and synchronisation functions indicate that task T1 is performed via three sub-tasks T11, T12, T13 carried out in parallel according to their trigger priority (constructor Or). The pilot can therefore "choose a bomb α", "select a point", or "select a rocket" in any order. It may be noted that task T12 "select a point" can be carried out in relation to three alternative triggers (alternative constructor); indeed, the choice of point is made in an way which is exclusive of the system initialisation, or during the preparation of another fire by the system during automatic selection or by the pilot during manual selection. Finally, control task T1 finishes either when the bomb is selected manually or automatically by the system (alternative constructor and synchronisation constructor).

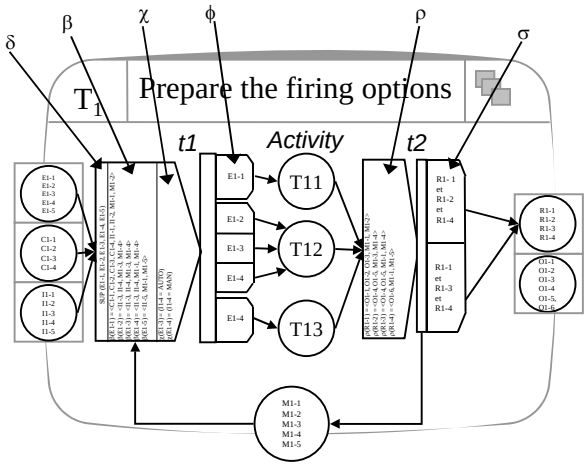


Figure 6: TCS Task Control Structure

In order to guide the designer during the specification stages, we propose a mechanism to model task interruption. Few methods make it possible to formalise task interruption; [15] presents a model of the interruptions based on Petri nets as a complement to MAD and UAN. The advantage of our method is that it completes the TCS with the interruption mechanism whilst maintaining the same formalism (Figure 7). This makes it possible to model the interruption of a task. An interruption takes place when a new higher priority trigger requires the performance of a task whose resources are being used by another lower priority task. At that moment, the task being performed goes into a suspended state (P8) and releases its resources. The high priority task is thus carried out. Once this task is finished and the resources are free once again, the performance of the suspended task is resumed according to three possible cases: at the beginning, at the point at which it stopped, or at the end when it is abandoned.

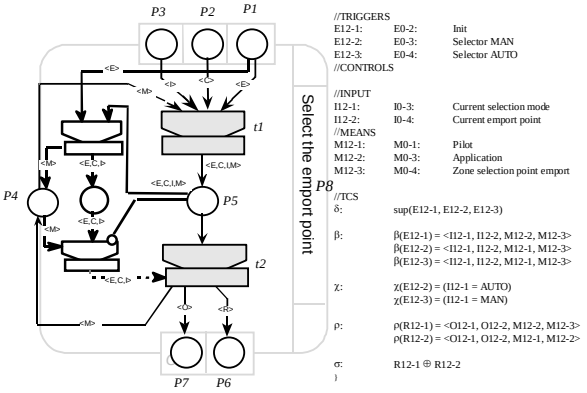


Figure 7: Interruption structure

The Figure 7 presents the interruption structure of the task "Select the point". The “Select the point” task may be performed manually or automatically. However, the pilot can, during manual selection, change to automatic mode. Automatic selection has priority over manual selection. This is modelled firstly by giving higher priority to trigger E12-3 relating to the automatic selector, and also by completing the TCS with an interruption structure.
The editor also allows the capture of the DFTM. The designer must then enter the information in the TCS on each of the tasks previously edited. For each TCS, it is necessary to indicate the distribution of input, controls and resources in relation to the trigger event as well as the release of output and resources in relation to each reaction (Implantation of the TCS, Figure 8).

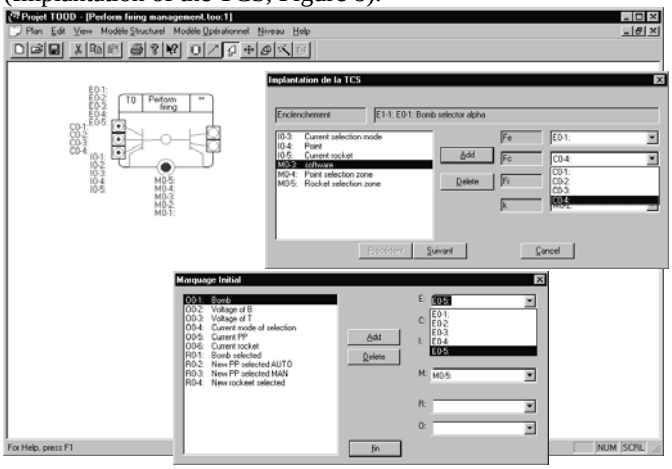


Figure 8: TCS Simulation

For the task “prepare the firing options”, the completeness function of the trigger E1-1: "Bomb selector α" is: β(E1-1) = <C1-1, C1-2, C1-3, C1-4, I1-1, I1-2, M1-1, M1-2> with C1-1 = current point, C1-2 = Sensor β present, C1-3 = Sensor t present, C1-4 = Current selection mode, I1-1 = Tension of sensor b, I1-2 = Tension of sensor t, M1-1 = Pilot, M1-2 = Bomb selection zone α. The user must then define the initial marking of the root task and launch the simulation of his scenario.
It is also possible, if necessary to enter the coherence rules for transition t1 of the TCS. The simulator makes it possible to observe the behaviour of the description such as it has been modelled under conditions described at the beginning of the simulation: it is a matter of establishing a scenario which allows the exploitation of different work situations by informing the initial marking window of the root task.

3 Operational Model (OM)

The aim of the operational model is the specification of the user interface to a high level of abstraction. In order to

achieve this, it defines the interaction between the User Model (UM) and the Local Interface Model (LIM), for each terminal task, in terms of objects, actions, states and control structure, Fig.1. An aggregation process of all the LIMs in the Abstract Interface Model (AIM) specifies the description of the final interface.
The Interface Implementation Model (IIM) is the low-level specification of the presentation of the final interface in multi-agent software architecture of the PAC-AMODEUS type. The implementation of this model is carried out by the translation of identified agents into terms of objects, states, actions, and chaining of the abstract model in the form of screens, menus, windows, icons based on a set of ergonomic criteria and recommendations [16], guidelines [12] and heuristics [17]. For more information on the operational model, the reader may refer to [18].

4 Conclusion

The use of the object oriented approach and object Petri nets presents several advantages for the modelling of the user task. Indeed, the TOOD task model, through its static and dynamic description, allows the modularity of specifications, the expression of interruptions and concurrency. The addition of describer objects to the task entity enables a connection to a programming language, which simplifies the passage to implementation.
Moreover, the TOOD method can contribute towards helping with communication between the different actors in the design process through its formal description. Because of lack of space, we have not approached the operational model, which leads to the specification of the HCI in this paper. This model is designed in continuity with the Functional model using the same formalisms, which favours the semantic stability of the TOOD method.
We are currently developing a software tool, which will support the TOOD method. The part concerning the Functional model of the task is finished. We still have to provide software tools for the operational model in order to facilitate automatic HCI generation.

Reference

1. Scapin, D.L., Pierret-Golbriech, C.: Towards a method for task description: MAD. In Berlinguet L. and Berthelette D. (Eds.). Work whith Diplay Units 89. Elsevier science publishers, Amsterdam, North-Holland (1990) 371-380
2. Barthet, M.F. : Logiciels interactifs et ergonomie : modèles et méthodes de conception, Dunod (1988)
3. Card, S.K, Moran, T.P, Newell A.: The Psychology of Human-Computer Interaction. In Lawrence Erlbaum Ass (Ed.). London. (1983)
4. Johnson, H., Johnson, P. : Task Knowledge Structures: Psychological Basis and Integration into System Design. Acta Psychologica. North Holland (1991) 3-26
5. Norman, D.A: Cognitive enginneering. In Norman, D. and Draper, S. (Eds.). User centered system design: new perspectives on human-computer interaction, Erbawn, New Jersy (1986) 31-61
6. Palanque, P. Spécifications formelles et systèmes interactifs, Habilitation à diriger des recherches, university of Toulouse I, France (1997)
7. Buisine, A. Vers une démarche industrielle pour le développement d'Interfaces Homme-Machine. Ph.D. Dissertation. INSA Rouen, France (1999)
8. Gamboa, F., Scapin, D.L.: Editing MAD* task descriptions for specifying user interfaces, at both semantic and presentation levels. In Harrison, M.D. and Torres, J.C. (Eds.), Proceedings of DSV-IS'97, Springer-Verlag, Vienne (1997) 193-208
9. Tarby, J.C., Barthet, M.F.: The Diane+ Method. In Vanderdonck, J. (Ed.): CADUI'96. Presses Universitaires de Namur, Namur. (1996) 95-119
10. Johnson, P., Johnson, H. Wilson, S.: Scenario-based design and task analysis. In Carroll, J.M. (Ed.). Scenario-based design: Envisionnong work and technology in system development. Willey (1995)
11. Bodart, F., Hennebert, A.M., Leheureux, J.M., Provot, I., Vanderdonckt, J.,Zucchinetti, G.: Towards a systematic building of software architecture: The TRIDENT methodological guide. In Palanque, P. and Bastide, R.(eds.), Proceedings of DSV-IS'95, Springer-Verlag, Vienne (1995) 262-278
12. Vanderdonckt, J.: Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive. PhD dissertation, Faculty Notre-Dame de la Paix, Belgique (1997)
13. Tabary, D., Abed, M.: TOOD: An Object-oriented methodology for describing user tasks in interface design and specification. In La Lettre de l'IA, vol 134 (1998) 107-114.
14. Sibertin-Blanc, C.: High-level Petri nets with Data Structure. 6th European Workshop on Petri Net Application, Espoo (1985)
15. Jambon, F.: Erreurs et Interruptions du point de vue de l'ingénierie de l'interface homme-machine. PhD dissertation, University Joseph Fourier, France (1997)
16. Bastien, J.M.C, Scapin, D.L.: Evaluating a User Interface With Ergonomic Criteria. Int. J. of Human Computer Interaction, vol.7(2) (1995) 105-121
17. Nielsen, J., Molich R.: Heuristic evaluation of user interfaces. In Proceedings CHI'90 (1990) 249-256
18. Mahfoudhi, A. TOOD: Une méthodologie de description orientée objet des tâches utilisateur pour la spécification et la conception des interfaces homme-machine. PhD dissertation, University of Valenciennes, France (1997)