Task model simulators: a review

THOMAS LACHAUME
LAURENT GUITTET
PATRICK GIRARD
ALLAN FOUSSE
LIAS / ISAE-ENSMA – Université de Poitiers

Abstract: Task modelling has enabled the building of models of human activity for a long time. In the early years, pencil and paper where the only means available to build task models from task model notations. Because of the lack of computed constraints, task models often did not conform to the notation. To solve this problem, some tools were designed by authors in order to help users create, modify and save correct models that conform to the notation syntactic rules. However, understanding the full semantics of task models appeared difficult for practitioners. The dynamic aspects of task models could only be understood "in the user's head". New tools, named simulators, emerged to solve this problem. They allow to "run" or to "simulate" task models and to record scenarios. This execution fulfils the semantics of task model operators, which define the task dynamic semantics. Simulators can be used in many ways such as understanding model semantics, verifying or validating models, building valid scenarios, etc. In this article, we describe and compare currently available and maintained task model simulators, and explain the different usages of these tools, according to user goals and qualifications. Then, we explore the different challenges for these tools to exploit the complete semantics of task models.

Key words: Task model, Simulator.

Résumé : La description de l'activité humain-système par des modèles de tâches existe depuis plusieurs années. Après l'époque papier-crayon, les outils d'édition des modèles de tâches ont permis la conception et l'archivage des modèles selon une notation rigoureuse, et la vérification de leur cohérence. Mais la compréhension de la dynamique est restée affaire de spécialiste jusqu'à l'apparition des simulateurs de modèles. Une simulation permet d'appréhender les enchaînements réels de tâches - décrits implicitement par les opérateurs temporels - et de valider les scénarios ainsi réalisés.
Cet article décrit et compare les simulateurs actuellement disponibles et maintenus, et explique leurs différents usages en fonction des buts et niveaux d'expertise des utilisateurs. De nouvelles perspectives d'évolutions de ces modèles et outils sont alors définies dans le but d'améliorer leur sémantique.

Mots clés : Modèles de tâches, simulateurs.

Adresse des auteurs : <prenom.nom>@ensma.fr – LIAS / ISAE-ENSMA – Université de POITIERS – Téléport 2, 1 rue Clément Ader, BP 40109, 86961 Chasseneuil Futuroscope Cédex, France

1. INTRODUCTION

During the last forty years, task modelling has been an increasingly researched topic. Methods became more and more precise, and increased their expressive power by the way of new features, such as calculable expressions and objects. Several tools were designed to support the edition, understanding and validation of task models. Some success stories were reported (see for example [Paternò et al. 2012; Martinie et al. 2012]), while usability in real projects is becoming a major issue.

Nevertheless, task modelling still needs much involvement on the user's part in understanding the notations. Sometimes, personal interpretation of notations is required, despite the efforts made by their authors to make them unambiguous. The best way to eliminate ambiguities in interpreting task modelling notations should be to use the tools that are associated to the methods. These tools are supposed to cover all aspects of task modelling such as enforcing the syntactic rules of languages, helping model understanding, and illustrating their dynamic semantics[1].

This last point is performed by tools named simulators[2], which allow the user to interactively simulate the activity that is described by the task model. Unfortunately, no complete description is available for these tools, and when comparing them many differences can be observed in their layout and behaviour. More, some concepts, which are described in the notation, are not taken into account in the simulation. Last, in some cases, behaviours are not really coherent when different concepts are associated in models.

In this article, we present a comparative review of different simulators, chosen for their current availability. In section 2, we describe the state of the art of task modelling, and focus on common concepts and existing tools. In section 3, we review the behaviour of simulators, with two illustrative examples. Section 4 summarizes the different usages of simulators, and section 5 highlights the current challenges for task model simulators.

2. TASK MODELLING STATE OF THE ART

Several authors compared task modelling approaches and tools. Nevertheless, none of them really focused on simulators. In this section, we first give an overview of task modelling approaches, which evolved among the past forty years. Then, we introduce the basic principles of task modelling, and we survey the different task modelling tools. At the end of the section, we focus on simulators, and justify our choice of four systems to study in the remainder of this paper.

2.1. OVERVIEW OF TASK MODELLING APPROACHES

Task modelling emerged from attempts to understand human activity through activity analysis [Diaper 2004; Sebillotte and Scapin 1994]. As written in [Annett 2004], "analysis is not just a matter of listing the actions or the physical or cognitive processes involved in carrying out a task, although it is likely to refer to either or both. Analysis, as opposed to description, is a procedure aimed at identifying performance problems

[1] Beyond the syntactical aspects of models, the dynamic or behavioural semantics of models
[2] One could say that these tools mainly animate the models, and should be called "animators". In this article, we follow the common author naming practices.

(i.e., sources of error) and proposing solutions". Basically, two different approaches can be identified.

The first one addresses mainly predictive evaluation, and can be illustrated by GOMS. Goals, Operators, Methods, and Selection rules [John and Kieras 1996; Kieras 2004] leans on the Keystroke Level Model (KLM) [Card et al. 1983]. GOMS models provide a way to quantitatively predict human learning and performance for an interface design. It can be used to make a qualitative description of how the user will use a computer system to perform a task. Lots of extensions [Kieras 2004; John and Kieras 1996] and derived approaches such as UAN [Hix and Hartson 1993] were later developed.

The second approach can be illustrated by methods such as HTA (Hierarchical Task Analysis [Annett et al. 1971; Annett 2004], TKS (Task Knowledge Structure) [Johnson and Johnson 1991; Johnson et al. 1988], TAKD (Task Analysis for Knowledge Descriptions) [Diaper 1989], or MAD (Method for Analytical Description [Scapin and Pierret-Golbreich 1990; Sebillote 1992]). These methods assume that it is important for task analysis to describe not only the activities people carry out, but also the contexts in which they perform those activities, as well as the ways those activities are performed, and the tools and methods that performers use. Their primary purpose is to elicit the knowledge structures that skilled performers (usually) construct when performing tasks. Beyond evaluation, these methods can be used to support interactive system design. Further works explored these issues more deeply; they propose methods with associated tools, such as Diane+ [Tarby and Barthet 1996], Mad* ([Gamboa et al. 1997]) and TOOD (Task Object Oriented Design [Tabary et al. 2000]).

Because the real activity is not generally a single user activity, several methods focused on cooperative activity, such as GTA (Groupware Task Analysis [van der Veer 1996]), CTT (ConcurTaskTrees [Paternò et al. 1997; Paternò 1999]) and CTML (Collaborative Task Model Language [Wurdel et al. 2008]), or reliability and critical systems, such as VTMB (Visual Task Modelling Builder [M Biere et al. 1999]) or AMBOSS ([Giese et al. 2008]).

Research activity on task modelling remained active for the last ten years, with attempts to generalization, with K-MAD (Kernel of Model for Activity Description [Baron et al. 2006]) and USIXML [Limbourg et al. 2005], or standardisation (W3C[3]), as well as new formalisms associated with powerful tools (ECOMM [Jourde et al. 2010a], HAMSTERS [Martinie 2011]), aimed at providing efficient assistance in designing interactive systems.

2.2. BASIC PRINCIPLES OF TASK MODELS

Despite this large number of approaches and models, basic principles of task modelling remain very stable. Methods do not differ much from initial approaches, such as HTA, where tasks are defined in terms of goals and operations, are divided into subtasks that follow a strict hierarchy, and embed a "plan", which defines rules for subtask execution. Following a general tendency of formalisation in HCI, task modelling evolved towards more and more precision of manipulated concepts.

This section offers a short overview of the main concepts that are used by most task models. The use of such concepts is illustrated by an example: *"Take the train"*. This task aims at modelling the activity of a person who arrives at the railway station to

[3] http://www.w3.org/TR/2014/NOTE-task-models-20140408/

travel by train. The system is composed of a ticket vending machine, large displays providing departure announcements, and stamping machines. Depending on the circumstances, the traveller may already have a ticket, or, if not, may buy a ticket at the ticket window or at the ticket machine. The ticket may be a physical one, which requires the traveller to stamp it before boarding, or an electronic one, which does not require stamping. If the traveller has omitted stamping, the ticket can exceptionally be stamped in the train by the human controller. The traveller is supposed to watch the departure announcement panel to find his/her way. This example is illustrated with a CTT model (probably the most largely used task model today) in Figure 1, but other models with different formalisms are given in Annex.

Basics of task modelling approaches are hierarchical decomposition of tasks into subtasks, and characterization of said tasks. Models converged to the ability of distinguishing roles during activities, assuming that activities involve human beings and systems. *User tasks*, which involve only human beings, *system tasks*, which conversely involve only systems, and *interactive tasks*, which involve both, are the main categories. Generally, *Abstract tasks* are used in task hierarchy to figure out nodes that can be refined in different types of subtasks. In our example (Figure 1), *Take the Train* and *Get Ticket* are considered as Abstract tasks because they are complex tasks, which are broken down into tasks from different categories. *Get Ticket* is composed of two options, *Get Ticket at Ticket Window*, which is considered as a User task (it involves a relationship between two human beings), and Get Ticket at Ticket Machine, which is Interactive, because it involves the user and the system. *Find Train* is also split in subtasks, which are not abstract, because its two subtasks belong to the same interactive category.

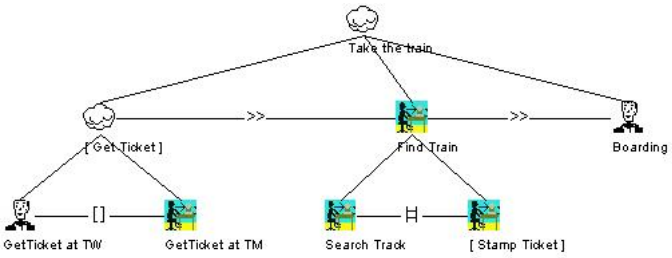

Figure 1: Take the train, CTT model

Annett's and Johnston's *Plans* between subtasks use quite equivalent ordering operators on every notation, which are based on a subset of LOTOS operators [Systems 1984; Paternò et al. 1992]. The basic Annett's plans, named *fixed sequence* (the activity is supposed to follow a predefined sequence of subtasks), *selective rule* (the user can choose between several subtasks), and *time-sharing* (subtasks are run concurrently) match respectively to *enabling* >>, *choice* [] and *concurrent* ||| operators. One more operator, *order independent* |=| (subtasks are not required to be made in a specific order), is generally added to this set. In our example, the first level of decomposition supposes a strict sequence between getting a ticket, searching for the train, and boarding. On the opposite, getting a ticket may be done at the ticket window or at the ticket machine, requiring a choice operator, while the *Find Train* task does not require any precedence between subtasks. Ordering operators are one of the major elements of task models, because they define the dynamics of the model.

Tasks own several attributes, which allow for a more precise description of the modelled activity. Some of them have consequences on task model dynamics, such as the optional attribute and the iteration attribute. The first one stands for optional tasks, which can be omitted during the activity. The second one defines repetitive tasks. In our model, *Get Ticket* and *Stamp Ticket* are optional. The first one needs to be done only if the user does not have previously bought a ticket. The second one is not required when e-tickets are used, and can be omitted if the traveller is late. In CTT, optional task names are represented between brackets.

Many task models and methods used several concepts to define operations that are the heart of activity. The need for context definition and for precision about conditions led to general notions of *precondition*, which defines the pre-requisite of the task, *action*, which describes the action to do to reach the goal of the task, and *post-condition*, which expresses the state required after the action. These elements can be informal (only texts are defined in the method) or formal; in that case, they involve objects and some kind of expression language. In our example, preconditions can be expressed on tasks, in order to state more precisely how they can be included in the activity. *Get Ticket* is subordinated to the absence of ticket, for example with a sentence like "*The traveller does not have ticket*" or with a condition on a Boolean object (*hasTicket* is ***false***). In the same way, *Stamp Ticket* is also guarded by a condition on the ticket (only possible if having physical ticket), and *Boarding* is guarded by the fact the traveller owns a ticket. Post-condition can be illustrated by an expression, which can be attached to *Get Ticket*: "*Assume that the traveller owns a ticket, either physical or electronic*". Actions can be described relatively to objects; for example, *Get Ticket at the TW* should be described with the action "The traveller buys a physical ticket" or *hasTicket* := ***true***[4]. With these elements, it is possible to verify the activity correctness by evaluating the pre/post-conditions, improving the global semantics of task models.

2.3. OVERVIEW OF TASK MODELLING TOOLS

Because modelling requires precision and non-ambiguity, many tools were designed to help task-modelling practitioners. Most tools are editors, which are able to enforce the notation used in the method. We can split these tools in two categories.

- Tools from the first category were designed after publishing the method, in order to help using it. The GOMS family owns several tools, such as compared in [Baumeister et al. 2000], and even first methods were supported by tools, such as ADEPT [Johnson et al. 1993] for TKS, Task Architect [Stuart and Penn 2004] for HTA, EMAD [Delouis and Pierret 1991] and ALACIE [Gamboa et al. 1997] for MAD and MAD*, or EUTERPE [van Welie et al. 1998] for GTA.
- Starting with CTT/CTTE [Paternò et al. 2001; Mori et al. 2002] the new generation of tools is deeply associated to the method and the notation from the early beginning. We can mention in this second category AMBOSS [Giese et al. 2008], TOOD/ETOOD [Tabary and Abed 2002], VTMB [M. Biere et al. 1999], and more recently K-MAD/K-MADe[5], e-COMM [Jourde et al. 2010b] and HAMSTERS[6].

[4] Which means: the Boolean object whose name is "hasTicket" becomes ***true***
[5] http://lisi-forge.ensma.fr/forge/projects/kmade
[6] http://www.irit.fr/recherches/ICS/softwares/hamsters/

All tools allow users to build task models, with respect to model rules. Most often, tools do no implement the whole task model, some rules being kept away.

Some tools stand on the dynamic semantics of notations to propose a specific category of tools, which allow the user to interactively simulate the modelled activity. Again, two main categories can be observed: simulators that require watching the model itself to simulate it, and simulators that do not display the model during the simulation (ProtoTask [Lachaume, Girard, et al. 2012]). We discuss more in depth this point in section 4.

We restricted our study to task model tools that fulfil the following requirements:

- Including a simulator that can illustrate most concepts of the underlying model
- Being widely available today, by free download or free registration
- Being supported, to allow bug correction and evolutions

Only three families of models/tools meet these requirements: CTT/CTTE, HAMSTERS, and K-MAD/K-MADe/ProtoTask. ProtoTask is part of the K-MADe environment, but must be considered as a specific simulator, very different from the standard K-MADe simulator. At our knowledge, the other systems are no longer available, or do not provide simulators.

3. COMPARISON OF SIMULATORS

Based on the same principles, all task model simulators work in a similar manner. Users are guided by task model semantics to simulate an activity, which follows strictly the task model dynamic behaviour. Beyond the obvious difference of visualisation, we can define some common principles that lead the simulation. Nonetheless, depending on the prime objective of the tool's authors towards modelling, some differences appear when carefully inspecting simulator capabilities.

The difference in task model visualisation is more than a simple choice between layout strategies, or icons to display tasks. It results from essential design choices, which lean on different principles.

The dynamic behaviour is the richest part of the models. Once again, choices that depend from authors' objectives lead to diverse solutions, which result in diverse possibilities for modelling. They result in the definition of constraints on the models.

In this section, we carefully study the solutions each system proposes. We reuse our first example, "Taking the train", and to illustrate some specific topics that cannot be explained on this example, we bring up the classical example of the "Cash machine", well-known in most HCI works. The "Cash Machine" is fully described in the Annex.

We start by describing basic simulator principles, which are all used by the different tools. Then, we detail the feedback the different simulators are able to provide to users. Last, we explain the simulator behaviour during simulation. In the last two sections, a table will summarize the studied criteria.

3. 1. SIMULATOR PRINCIPLES

Simulators are supposed to illustrate the concrete semantics of task models, as expressed by using task attributes and ordering operators. From the point of view of simulator's users, two basic functionalities can be brought out:

- interactive simulation of activities,
- building scenarios.

3.1.1. Interactive simulation of activities

The first objective of simulators is to show a concrete illustration of a simulated activity, "running[7]" task model semantics. At simulation run-time, subtasks are proposed to the user according to the different operators attached to the model. Similar to a debugger tool, the simulator works step-by-step, each step simulating one task; after each step, the user is asked to choose the next task. For example, in Figure 2, we can see the CTTE simulator during a cash withdraw task, while the user already entered his/her password and the required amount, and just before recovering his/her bank card. At this point in the task, the only possible action to do is *Withdraw Card*, because of the ***Enable*** operator (>>) between the two tasks. The only choice for the user is to accept the proposed task.

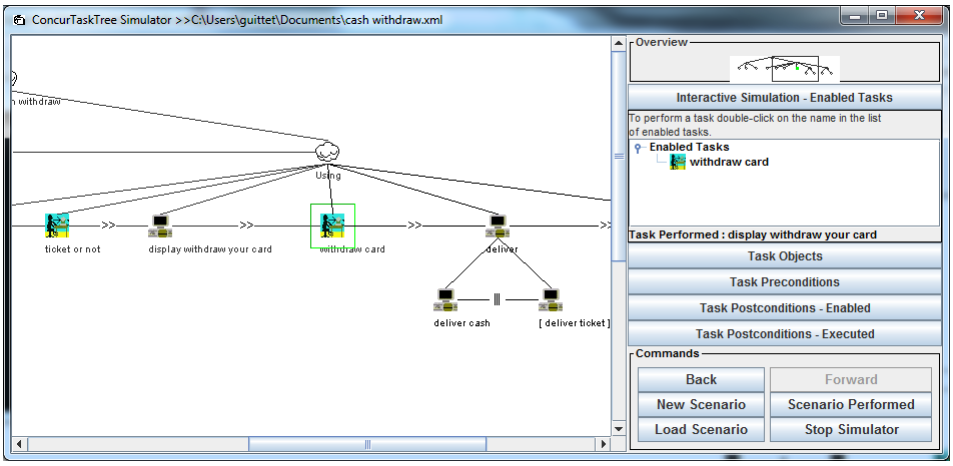

Figure 2: View of the CTTE simulator in action

Figure 3 shows the same simulator several steps later, when cash and ticket are available for the user, and *Withdraw Cash* and *Withdraw Ticket* are available at the same time because of the ***OrderIndependance*** operator (|=|). At this time, the user of the simulator is given the choice of selecting the order in which s/he wants these two tasks to be performed.

[7] in the computer science sense of the word

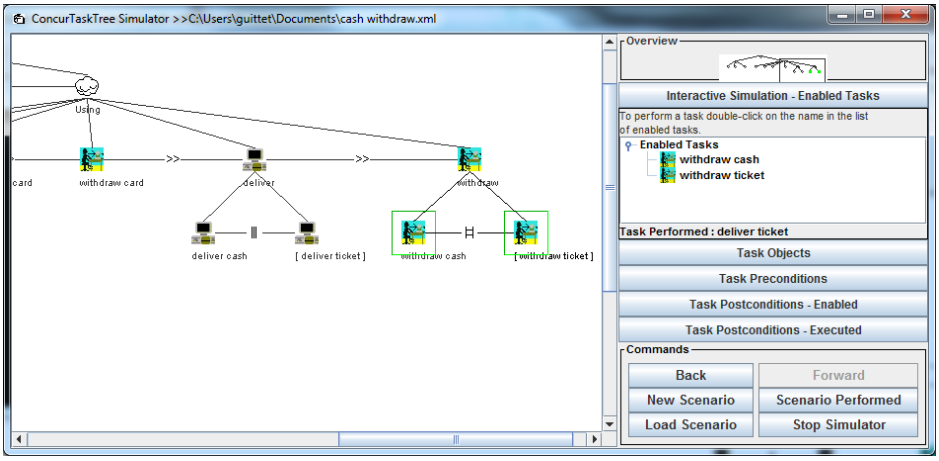

Figure 3: a simulator with several choices

While "running" the task model, the simulator builds a scenario, which stands for operating a concrete activity, in line with the task model. This scenario can be saved, and reloaded, in order to play it again onto the simulator for this particular task model.

The heart of simulators is the Enabled Task Set (ETS) calculus, which calculates all possible tasks within the model at each step [Paternò 1999]. In this definition *Enabling* means making a task available for simulation, alone or with all other available tasks conforming to task model operators.

Attributes can also be taken into account by simulators. For example, the ***Optional*** attribute states that a task can be omitted. So, when included in an ETS, this task does not prevent nor enforce the execution of other tasks. For example, if task *Withdraw Ticket* is defined as ***Optional*** (because the user does not need the ticket) the optional task can be skipped and the simulator goes to the next available task in the model.

Another attribute acts like an operator. It is the ***Iterative*** attribute. Attached to a task, it means that this task is in fact a repetitive task, whose simulation does not just end after all its subtasks have been done once.

The last versions of simulators take into account objects and/or pre/post-conditions to enable tasks. Preconditions, which can be based on objects, may guard task enabling. For example, the *Withdraw ticket* task should not really be optional in the point of view of the user. It should depend on the choice the user made several steps beforehand, when s/he decided whether s/he wanted a ticket. Without using preconditions, scenarios can be built that are not practical or accurate, for example by choosing first the option of not asking for ticket, and then attempting to withdraw a ticket. Preventing this kind of bad use of models can be made via the use of pre/post-conditions. The most general way to make pre/post-conditions in task models supposes the definition of objects (Boolean variables for example) to be attached to tasks as a precondition in the form of an expression, which can be evaluated by the simulator to state whether to enable the task. For example in this case, the *Withdraw Ticket* task can be guarded by a Boolean value whose meaning could be "Ticket required". We explain later how this function is implemented differently in each of the tools.

Each task model tool has its own specificities, depending on specific objectives followed by its authors.

CTT focuses on cooperation, with cooperative tasks, and on generation possibilities. It proposes more precise descriptions of objects, and introduces the platform the application is supposed to work on. HAMSTERS addresses "modelling in the large", with modularization features, and interactive animation with a link to the PetShop tool [Bastide et al. 2002]. K-MAD (including both K-MADe and ProtoTask) manages actors, improving the expressive power of task models.

All these topics induce specific management in the simulator tools. Because they all are specific to one tool, we did not include them in this review. We also did not specifically study the simulator's management of iterative tasks. In fact, the iteration itself is managed the same way in all tools. The difference in managing the end of iteration depends completely on the choice made in the notation: in CTT and HAMSTERS, ending iterations need the usage of a specific operator, called "disabling", while K-MAD provides a computer language like mechanism based on a loop condition.

3.1.2. Building scenarios

The second objective of simulators consists in building scenarios. While "running" a task model, the user builds a list of tasks, which is in fact an example of a concrete activity. All simulators are able to record in a file such scenarios, and to load and "run" again these scenarios. These scenarios do not generally include any value for objects, but only elementary tasks.

Beyond the above main principles, simulators differ in the following two ways. First, they display different information, depending on their global philosophy. Second, their behaviour during the simulation may be different.

3. 2. SIMULATORS LAYOUT

The main two simulator feedbacks concern the way tasks are proposed to the user, and the layout of current simulation. At the end of the section, we propose a table, which summarizes the differences between the systems.

3.2.1. Displaying the Enabled Task Set

The main objective of simulators is to allow the user to simulate, step-by-step, task models. At each step, the user is required to choose what task to do among the ETS. For this purpose, the system must give the user all information required to understand the task context in order to make an informed choice on the next task to perform. The only common view between all simulators is the ETS view, which allow the user to check what task can be run at each step.

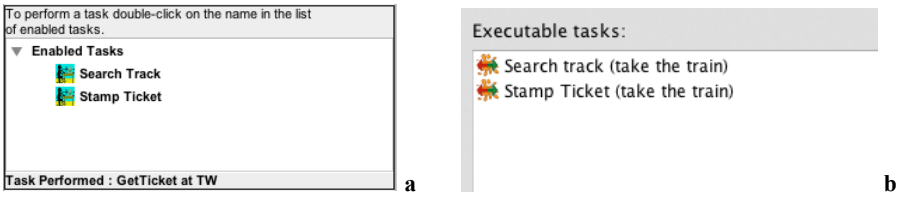

Figure 4: ETS in CTTE(a) and HAMSTERS(b)

HAMSTERS and CTTE display a list of enabled tasks, with no more information (Figure 4). K-MADe adds a qualification to each task, to make explicit the fact that the

user wants to "skip" the task. This point relates to optional tasks, whose execution depends on a choice made by the user. Figure 5a illustrates a state where a user can choose to stamp a ticket or not.

While only terminal tasks (the leaves of the task tree) are displayed in the first three simulators, the Prototask simulator chooses to represent things differently. First, it allows the selection of nodes as well as terminal tasks. Then, in a panel where all tasks from the same decomposition level are shown, only enabled tasks appear as enabled buttons while other tasks are visualized with disabled buttons. In the example of Figure 5b, we can see the ETS of the "Take the train" task model, after simulation of task *Get Ticket*. The other two tasks of the main level of decomposition are visualized, but greyed because they are not currently enabled (*Get Ticket* is done, and *Boarding* not yet enabled). We can also see that preconditions are also written near the task name.

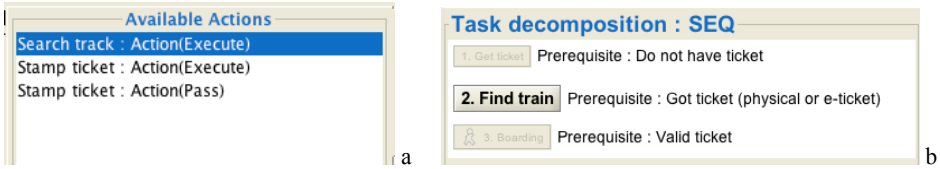

Figure 5: ETS in K-MADe(a) and Prototask(b)

3.2.2. Understanding the goal/subgoal hierarchy

To make the user understand the context of the current task, displaying the ETS is not enough. Because task trees reflect goal/subgoal mental decomposition of users, displaying only enabled tasks with no reference to the goal hierarchy would lead to an overflow of the cognitive load on the user's part. Two different strategies were used to solve this problem. The first strategy leans on the task model itself. It can be illustrated by the CTTE, HAMSTERS, and K-MADe simulators. In Figure 6, the task tree in CTTE uses the main part of the window, with colour codes to help understand what tasks are enabled: the simulation step is inside the *Find Train* task, after completion of task *Search Train*; conforming operator semantics, the only enabled tasks are *Stamp Ticket* (on the same level with *Search Train*, and an ***OrderIndependence*** operator), and *Boarding* (because *Stamp Ticket* is an optional task). All two are rounded by a green square on the graphic panel. Only ETS is added to the visualization, and buttons to control the simulation.

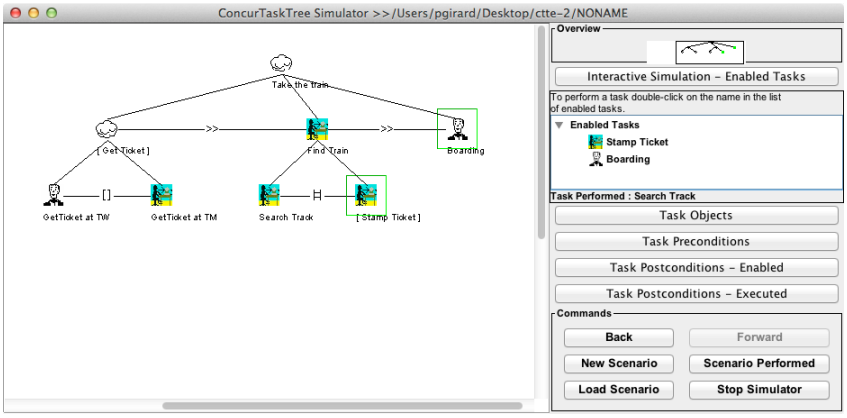

Figure 6: The simulator in CTTE

HAMSTERS also highlights the current enabled tasks in the graphical model, using a colour code to help the user understanding the context: green tasks are enabled, while red tasks refer to skipped tasks. A panel shows the in progress scenario (Figure 7).

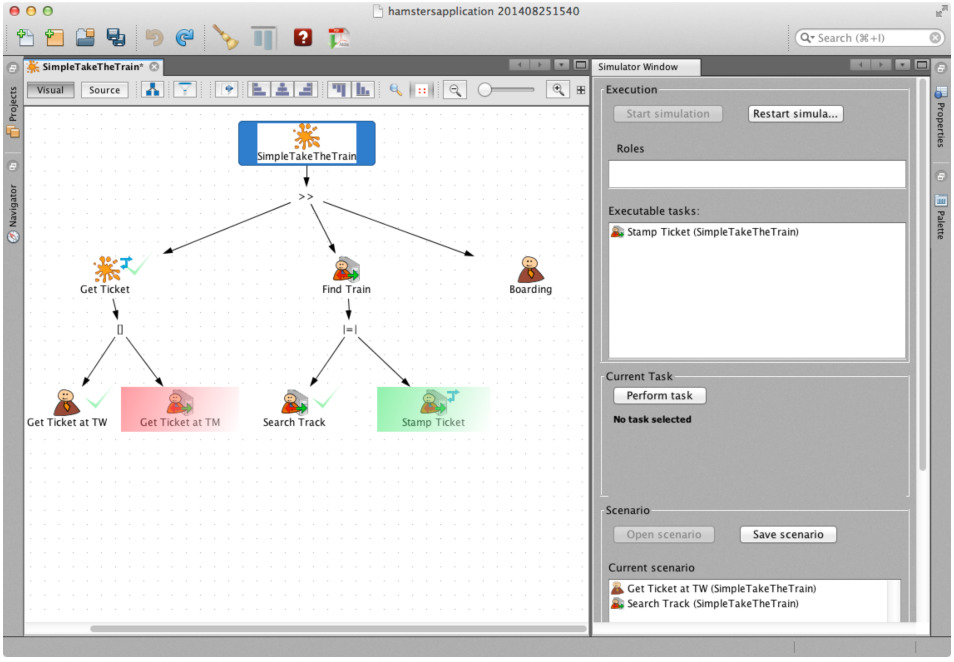

Figure 7: HAMSTERS simulator in action

K-MADe adds information through colour codes about *already done* subtasks (grey), *in progress* tasks (light bue-grey), *passed* tasks (green) and *enabled* tasks (blue). The step in Figure 8 is also during the *Find Train* task, but here, the user decided to pass the *Stamp Ticket* task. Like HAMSTERS, the window includes the scenario in progress. Looking like a debugger, the K-MADe simulator also displays information about the selected task (attributes, and so on), several other information about objects, actors, constraints, etc.

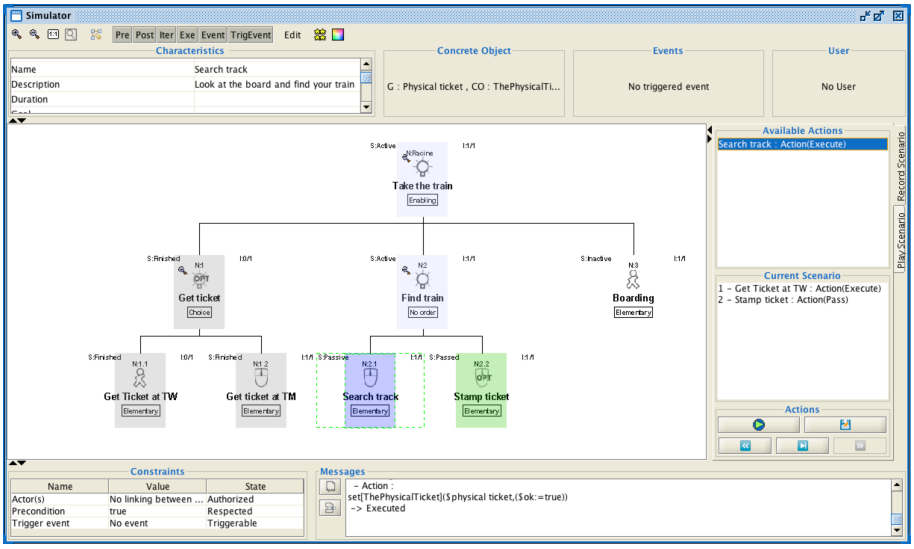

Figure 8: K-MADe simulator in action

The second strategy to help the user understanding task context is proposed by ProtoTask. This tool does not show the tree at all. Instead, it displays relevant information about the current task, and provides a hierarchical view of the scenario, as displayed in Figure 9.

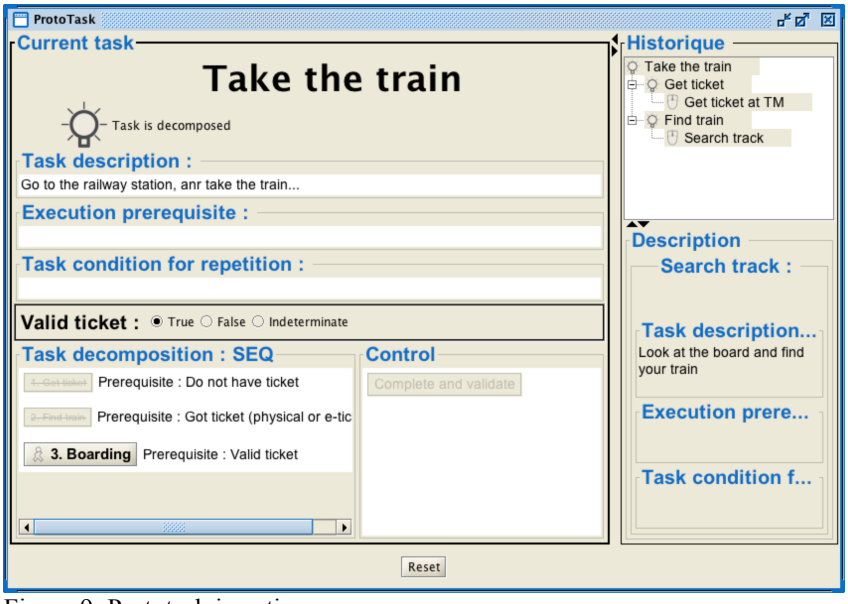

Figure 9: Prototask in action

The main window in Figure 9 focuses on the current task, providing a detailed description (if put into the task model), the different conditions (precondition, iteration condition), and the task decomposition with all enabled tasks. On the top right corner, a navigable scenario is displayed, in a tree-like format, in order to figure out the goal/sub-

goal user strategy. Some experiments showed that this representation is suitable for understanding the global activity [Lachaume, Caffiau, et al. 2012]. In the figure, the step is just after completion of task *Find Train*; the scenario panel shows that the optional *Stamp Ticket* task was not performed.

3.2.3. Summary

Table 1 below summarizes the differences described in this section. Beyond the obvious difference between Prototask and the other three simulators, other differences can be reported. They result from design choices and different policies in the user interface. Note that Prototask is intended to be seen by non-expert users, while the other three systems address mainly task modellers, engineers and domain experts.

LAYOUT	CTTE	HAMSTERS	K-MADe	Prototask
ETS view	simple list of terminal tasks	simple list of terminal tasks	list of terminal tasks with qualification (explicit skip)	list of all sub-tasks and their preconditions
task tree	viewed with colour codes	viewed with colour codes	viewed with colour codes	not visible
scenario	not directly visible during the simulation	visible as a list of terminal tasks during simulation	visible as a list of terminal tasks during simulation	visible as a tree during the simulation
conditions	not directly visible during the simulation	not directly visible during the simulation	not directly visible during the simulation	visible for the current state

Table 1: Summary of differences of visualization

3. 3. BEHAVIOURS

As we described in previous sections, the different simulators lean on the same basic concepts, but differ from each other by several options. The divergences increase when looking at the dynamic behaviour of simulators during simulation. We specifically address four topics. First, we describe the specific case of optional tasks. Then, we focus on task node management and task completion. Next, we address object and pre/post-condition management. After giving an insight on ending model management, we give a summary of all differences between tools.

3.3.1. Optional tasks

The basic principle of task model simulation is to determinate the Enabled Task Sets, i.e. the set of tasks that can be activated at each step of the simulation. This set is calculated according to the semantics of operators. For example, all tasks governed by a ***Choice*** operator are enabled at the same time, while access to tasks governed by a ***Sequential/Enable*** operator is restricted by the order of the sequence, only one task being available at each step. Interactive simulation consists for the user in choosing in the ETS the task to simulate. All simulators work the same way for this point, but differences appear when managing optional attribute and nodes.

In CTTE/HAMSTERS, optional tasks[9] are inserted in the ETS exactly the same way as non-optional tasks. Skipping optional tasks is not a user action. Tasks are skipped because after one step, they are no more in the ETS. For example, Figure 10 shows the "Take the Train" simulation at start.

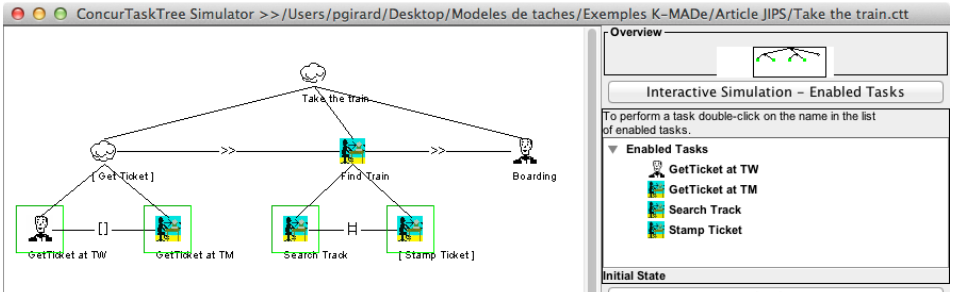

Figure 10: Optional tasks in CTTE

The first two enabled tasks are the terminal tasks of sub-task *Get Ticket*, which is the first task in the first level of sequential decomposition. These two tasks are exclusive (***Choice*** operator). We also can see in the ETS the other two sub-tasks of *Find Train*, which are also enabled. The reason is that *Get Ticket* is an optional task (written in brackets). Implicitly, if the user selects *Search Train* or *Stamp Ticket*, *Get Ticket* is skipped. HAMSTERS and Prototask work the same way.

In K-MADe, optional task management is explicit. The user is requested to explicitly skip the optional task when s/he does not want this task to appear in the current activity (the scenario currently in progress). This can be seen on Figure 11, where the ETS is extracted on the bottom right of the figure. At the start, the user is supposed to get a ticket using one of the two different methods, or to switch directly to the next task. In the last case, s/he must explicitly skip the *Get Ticket* task

[9] We call optional task a task of which the optional attribute is set to true

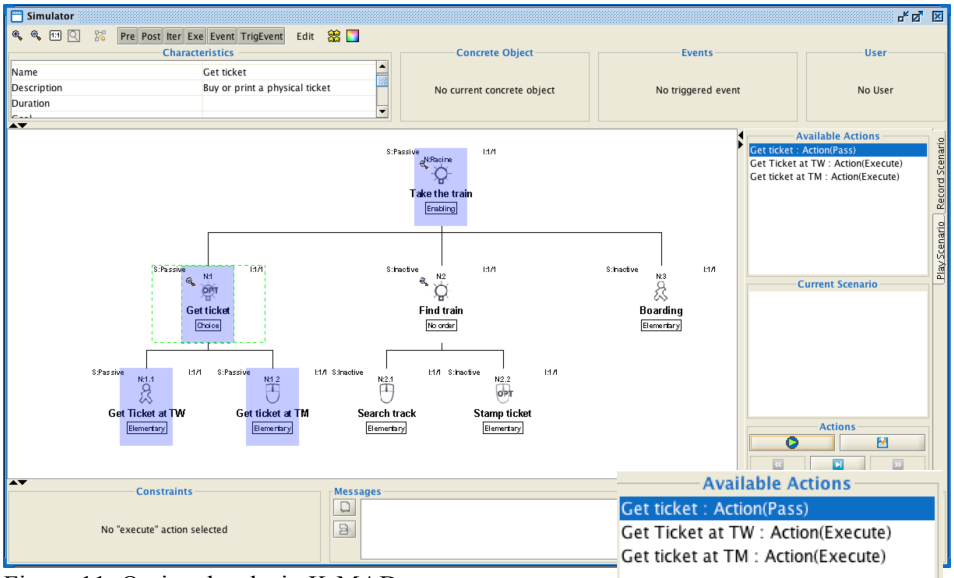

Figure 11: Optional tasks in K-MADe

Figure 12 below shows the situation after skipping the first group of tasks.

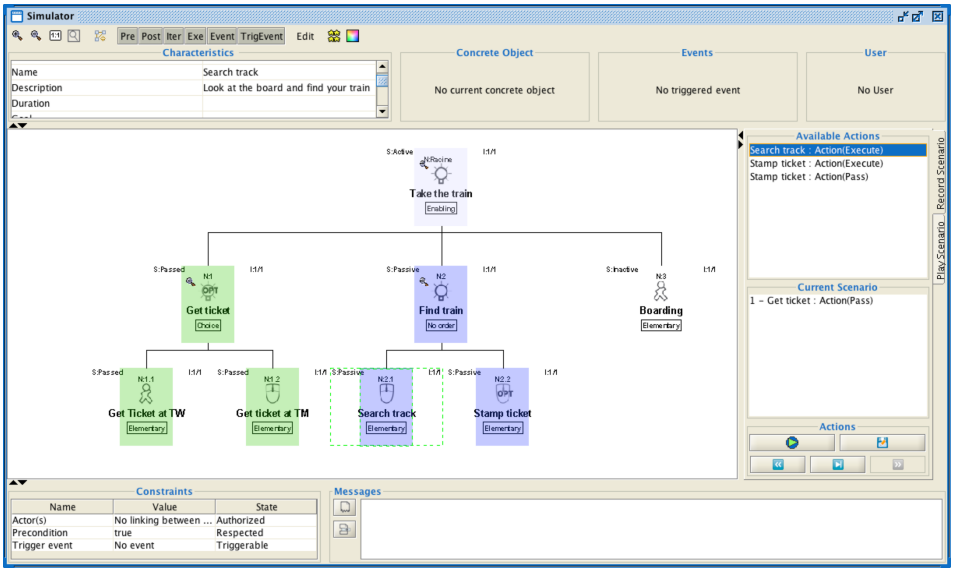

Figure 12: One step further, after skipping the task

3.3.2. Node management and task completion

Enabled Task Sets in CTTE and HAMSTERS only include terminal tasks, i.e. task tree leaves, as introduced in section 3.2. This fact makes it difficult for the user to figure out the whole activity, to know what global task is started, what are the consequences of choosing a subtask, and so on. To solve this problem, tools provide complementary views as described in section 3.2, such as the global tree view. For example, in Figure

13, the context awareness in CTTE is illustrated. Looking at the ETS panel, it is not possible to understand the exact context, which represents the following situation: the *Take the train* task is started, choosing either starts a new subtask, and choosing one of the last two tasks starts a new subtask, *Find train*, making *Get Ticket* unavailable. All this can only be understood through careful analysis of the tree shown on the left. Skipping tasks and ending tasks are all implicit, which make it impossible to have only optional tasks in the ETS.

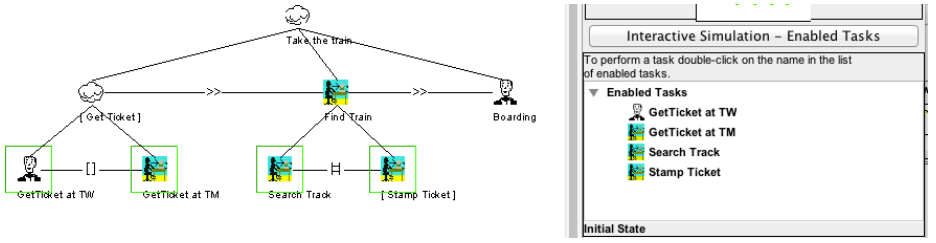

Figure 13: Context awareness in CTTE

K-MADe proposes making the skipping process of all *optional tasks* explicit, what impacts also the node management. Figure 14 shows the K-MADe simulator at the same state as CTTE above. The ETS shows only the first two subtasks, and the option to skip the node task *Get Ticket*. Choosing it enables *Find Train*, but only its two subtasks, because it is not optional itself.

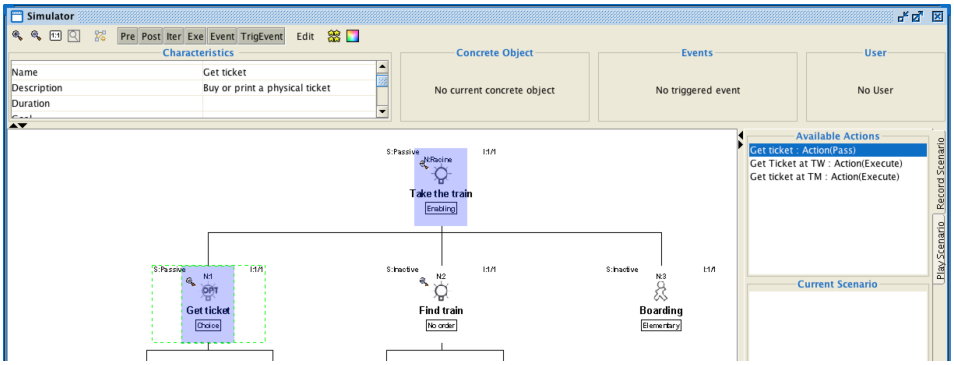

Figure 14: Explicit optional task *skipping*

ProtoTask goes one step further in making explicit the choice of node tasks and the process of ending tasks. Figure 15 shows Prototask after validation of *Get Ticket at TW*; as shown on the top, we are currently in the *Get Ticket* task node, no action is possible with the exception of the final action "***Validate and Terminate"***.

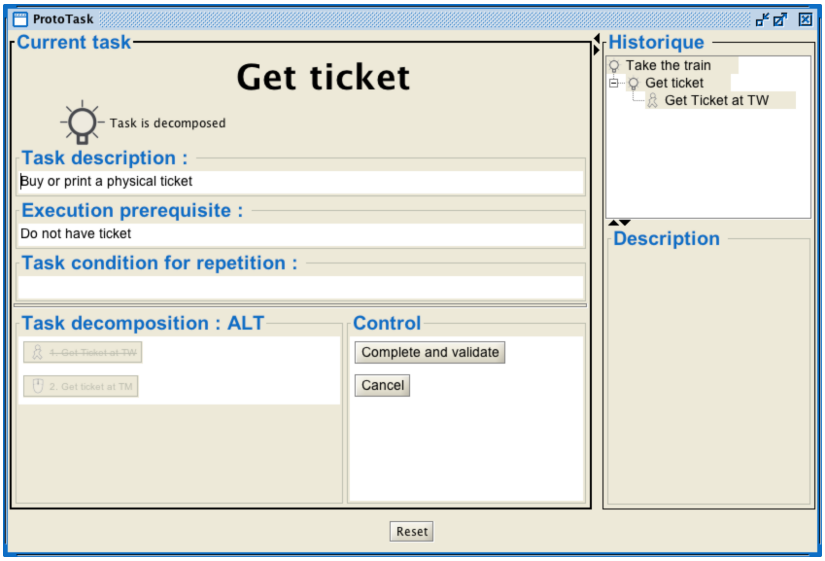

Figure 15: Explicit ending of a task

3.3.3. Object and pre/post-condition management

In recent years, object and pre/post-condition management has become increasingly included in editing tools, and as a consequence, has been better considered in simulators. The increased expressive power given by pre/post-conditions allowed taking into account more realistic activity descriptions. Once again, differences in object management can be observed during simulation. Two different policies are used in the tools. The first one is a human interactive policy, used by CTTE and ProtoTask, and the second one is a process directed policy, used by K-MADe and HAMSTERS.

CTTE and ProtoTask ask the user to personally manage the entire evolution of pre/post-conditions during the simulation. In the control panel of CTTE simulator, several panels allow the exploration and manipulation of objects and pre/post-conditions. Figure 16 shows on the left part the object panel, which gives the description of objects (name, type and value) and their owners (tasks where they are defined), and allows the value to be changed. On the right part, the precondition panel shows the preconditions of every concerned task, with their expression and their current value. We use the word “concerned” to define all possible enabled tasks. The ETS panel only presents enabled tasks, which means tasks that are enabled by the temporal operators and attributes (optional), and whose possible precondition is calculated as true.

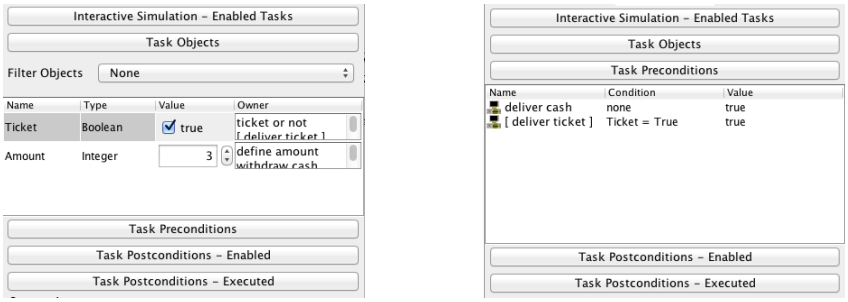

Figure 16: Object (left) and preconditions (right) in CTTE simulator

In the precondition panel, even tasks with false values are shown. In Figure 17, we can see on the left part the only enabled task (*Deliver Cash*), while in the precondition panel (right part), the two potential tasks are displayed, with a "false" value for *Deliver Ticket*.

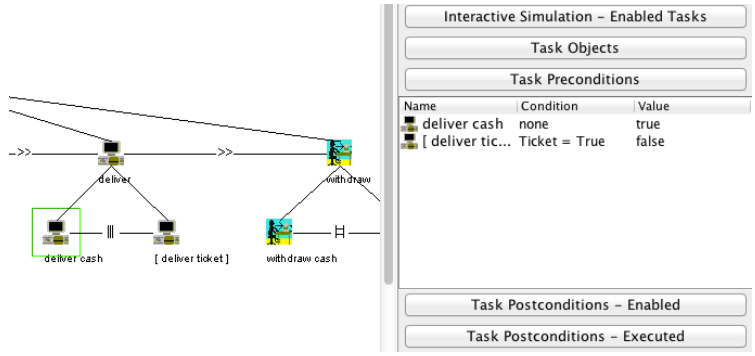

Figure 17: Task with false preconditions in CTTE

In CTTE, using objects and conditions requires the user to manually change the values of objects in order to build scenarios that cover the different options.

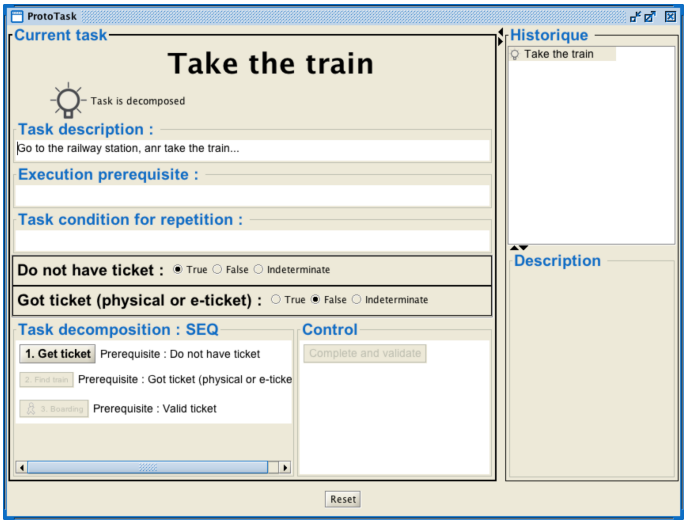

Figure 18: Precondition management in Prototask

ProtoTask uses the same policy, but does not use objects: preconditions are represented by text sentences, which are displayed with True/False/Undetermined status; the user is supposed to give the value s/he wants this precondition to hold for this task in the current scenario. In Figure 18, we can see the situation at the beginning of the "*Take the Train*" activity. In the ETS part, all subtasks of the main task are shown, with their precondition, and only preconditions of concerned tasks are modifiable. Here, the last task (*Boarding*) is not available because the *Search Train* task must be made beforehand.

On the opposite, K-MADe and HAMSTERS do not really allow an interactive modification of object values during simulation. Instead, they provide a way to describe side effects on objects. Tasks are able to define assignations (HAMSTERS) or actions (K-MAD), which allow user to change the values of objects. For example, in K-MADe, *Get Ticket at TW* is supposed to set a *Physical Ticket* boolean value to true. This is done through an action, which is visible on Figure 19a when selecting this action in the ETS panel, while after execution of this task, the feedback messages explain that the action is done (Figure 19b).

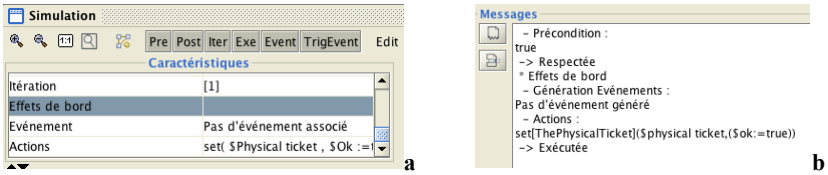

Figure 19: Action management in K-MADe

Thanks to this management, preconditions can be correctly evaluated during the complete simulation.

Note that no simulator manages correctly the precondition and optional attribute association. Optionality is always a user choice. So, it is not possible to state that a task is optional depending on a precondition, i.e. mandatory if the precondition is true, and systematically omitted if the condition is false.

The last difference in managing pre/post-conditions is in the way simulators give access to tasks whose precondition is false. In CTT and HAMSTERS, these tasks are considered as not enabled, which means they are not in the ETS view. In ProtoTask, they are greyed, the same way not enabled tasks are. In K-MADe, such tasks appear in the ETS panel exactly as if they were enabled; if the user tries to simulate them, an error message gives a feedback to the user, explaining that the precondition is not fulfilled.

3.3.4. Managing end of simulation

The last main behavioural difference for common features in simulators can be seen in managing the end of simulations. What does "ending" the activity really mean? Using only temporal operators, the answer is quite simple: when no more tasks are present in the Enabled Task Set, the activity can be considered as completed.

Nevertheless, when adding attributes and preconditions, the problem becomes more difficult. If the last task in the ETS is optional, how should the fact that it is not performed be represented? If the only possible task is guarded by a precondition, and this condition is false, what does it means?

In CTT and HAMSTERS, only enabled tasks, with preconditions set at true, are available in the ETS. So, when the ETS is empty, the activity is supposed to be

completed. There is no way to state that the task is completed even if an optional task is always available. To eliminate this problem of optional tasks, a rule in CTT states that it is not possible to end a model with optional tasks. Nevertheless, the CTTE simulator does not manage this case.

In K-MADe, a specific case for optional tasks was defined. When an optional task is enabled, the user is asked to run it, or to skip it. This is an explicit choice for stating that an optional task is not to be performed.

In ProtoTask, the problem is solved by the specific notion of task completion. The user is asked to assert that a task is completed. So, when a task is optional, the user can choose between this task and the explicit ending of the node. For example in Figure 20, the *Stamp Ticket* task is optional, so the ProtoTask's user can choose between running the task or completing (***Validate and Complete*** button) the *Find Train* task.

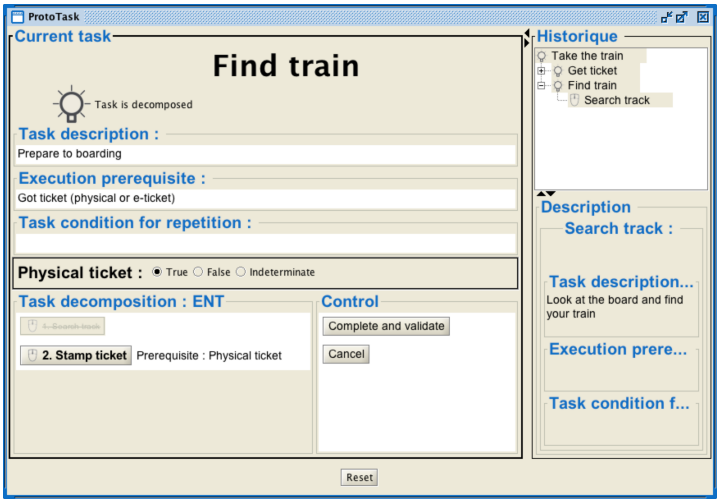

Figure 20: Task completion in ProtoTask

3.3.5. Summary of behavior differences

Table 2 summarizes the differences described in this section. We can see that these differences do not determine two categories of distinct simulators. For example some of them converge for simulators which have very different objectives, such as CTTE and Prototask for the condition management policy.

BEHAVIOUR	CTTE	HAMSTERS	K-MADe	Prototask
optional tasks	implicit skip	implicit skip	explicit skip	explicit skip
node management	none	none	explicit management of optional nodes	explicit management
task completion	implicit	implicit	implicit	explicit control
object/conditions management policy	human interactive	process driven	process driven	human interactive
objects	different predefined object types	Booleans	user-built and predefined object types	no objects
pre/post-condition	pre-defined Boolean expressions	Boolean expressions	free Boolean expressions	textual expressions
side-effects on objects	no	yes (assignations)	yes (actions)	no
control of guarded tasks	proactive (not visible)	proactive (not visible)	post-control (appears enabled)	proactive (appears disabled)
Managing end of simulation	automatic	automatic	controlled for optional ending tasks	controlled

Table 2: Summary of behaviour differences

4. SIMULATOR USAGE

Within the task modelling literature, no comprehensive reviews of the different simulators have yet been compiled. Whilst they appear in some tool descriptions, or in description of case studies, the only article where simulators were compared in terms of usage is [Paternò 2002], which only includes two simulators, CTTE and VTMB (no more available for use by practitioners). Nevertheless, by extracting details from different articles, it is possible to state what kind of simulator's usage has been made. Three main kinds of usage may be noticed: understanding, exploring and explaining, and validating.

4.1. UNDERSTANDING

The first usage of simulators results form the fact that simulators enforce and make explicit the semantics of the task modelling notations: it allows users to understand with no doubt the actual dynamic semantics of the model they are looking at. As we saw in previous sections, some behaviours are not really described in method presentations, and using the simulator is the only way to know what interpretation must be made.

This point makes the choice to use simulators while teaching task modelling obvious. As reported in [Caffiau, Scapin, et al. 2008], using simulation is very important for students to figure out what the task model really models. The coordinated views of the ETS and the current task model are particularly suitable to understand the context of the current state, and to master the different elements of the notation. CTTE, for example, changes the task model display after each task simulation, centring the task

model on the current enabled tasks. Displaying the scenario which is in progress is provided by K-MADe and HAMSTERS, helping users to remember what they did. K-MADe provides a very complete view of the task model, which allows to explore every element of the model. Prototask does not show the task model itself during the simulation. Instead, it makes explicit the goal/subgoal hierarchy, by asking the user to enter a subtask, and by always showing the current scenario, in a tree-like representation. In [Lachaume, Girard, et al. 2012], the authors provide results that proved a high level of user understanding of the task model.

4.2. EXPLORING AND EXPLAINING

The second usage originates from the first goal of simulators: animating the dynamic semantics of models. The result is a visual and dynamic representation of this dynamics, which makes clear the behaviour that underlies the activity.

As pointed out in [Paternò 2002], the simulation may then be the support for a multidisciplinary discussion between people with different background to take design decision at the task level, during early stages of system design. Nevertheless, understanding what happens during simulation requires a minimum knowledge about task modelling; indeed, people involved in this kind of activity must be able to interpret complex tree views, with coded representations. Reported experiments [Paternò et al. 2012; Martinie et al. 2012] confirmed the correctness of this approach.

On the opposite, using traditional task model simulators for interacting with end-users or stakeholders seems more difficult, as established in [Caffiau, Guittet, et al. 2008]. The ProtoTask approach [Lachaume, Girard, et al. 2012], where the task tree is never shown to the user, proved to be efficient in allowing end-users to validate their needs. Improved visualisation of in progress scenarios and explicit usage of task/subtasks decompositions allow end-users to correctly figure out the simulated activity.

4.3. VALIDATING

Validating modelled activities is a crucial challenge. The first need is validating scenarios. Simulators allow users to build scenarios while simulating models. Recorded scenarios may then be checked over the actual systems, or played again on the simulator after changes [Girard et al. 2013]. Nevertheless, this topic has not been yet fully investigated, and we propose new directions in the conclusion.

On the opposite, linking task modelling simulators to system models has been investigated in [Barboni et al. 2010]. The HAMSTERS environment is linked to the PetShop environment [Bastide et al. 2002] to allow co-execution of task and system models. This way, designers are able to validate the actual behaviour of the system compared to task models.

5. CONCLUSION AND PERSPECTIVES

In this article, we propose a comparison of currently available and maintained task model simulators. This review highlights differences in behaviour, which are not necessarily explained in the tool presentations.

Despite the wide range of usages described in section 4, several challenges remain for task model simulators. They can be split in two categories: modelling challenges, and usage challenges.

Task methods incorporate more and more concepts, increasing greatly their expressive power. In the meantime, simulators do not take into account some of them. A first example is parallelism. While parallel operators are generally available in the notations, managing parallelism during the simulation is not really done, and feedback of parallel tasks is very poor. A second example relates to interruptions. Interrupting, and possibly resuming a task is a very complex process, which relates to very different things, such as cognitive aspects for the user, interrupting and resuming processes, and so on. Time is also a topic that is not really taken into account during simulation. Verification of timing issues can be made during simulation, but no tool seems to manage it yet.

In section 3, we explored how the different simulators manage objects and pre/post-conditions. This management is very different from one tool to others. What should really be the place of objects and pre/post-conditions in task models? What should be their role in the simulation? All these points need more investigation. Problems remain in using simulators. The major example relates to scenarios. Current scenarios record only tasks. They do not consider the context of the tasks. Replaying scenarios requires to take care of this context; how is it possible to obtain the same result if the conditions of tasks are different? No simulator takes care of this point at the moment. In the same way, exploring models and modifying parameters during simulation is not really standardised over the different simulators.

Despite academic attempts, task models are not widely used in system design processes. Connections between task models and UML models, or process modelling notation such as BPMN[10] for example, are not really clear. The already mentioned W3C initiative for standardizing task models did not lead to a significant usage of task models in current software engineering methods. Agile methods promote extensive discussion between users (or product owners) and developers using simple descriptions. Despite of the reported experiences with CTTE or Prototask, which seems to demonstrate that this dialogue can be improved using task-modelling tools, task models have not yet found their place in this process.

The last point concerns simulator usability. Many differences between the different tools result from strong design choices and tool policy (as for example explicitly skipping optional tasks, or explicitly activating task nodes). Authors demonstrated the correctness of their choices, but no comparative study has been made between these different tools. More, whereas task modelling comes from ergonomists and psychologists, it seems that advanced tools, and more precisely those with simulators, are mainly used by software engineers and HCI designers. The reason of this situation should be investigated, and comparative studies could bring interesting points about task model acceptance by different publics and task analysis and modelling benefits.

6. BIBLIOGRAPHY

ANNETT, J., 2004. Hierarchical Task Analysis . In D. Diaper & N. Stanton, eds. *The handbook of task analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates, pp. 67–82.

ANNETT, J., DUNCAN, K.D., STAMMERS, R.B. AND GRAY, M.J., 1971. *Task analysis.*, London: Her Majesty's Stationery Office.

[10] http://www.bpmn.org

BARBONI, E., LADRY, J.-F., NAVARRE, D., PALANQUE, P. AND WINCKLER, M., 2010. Beyond modelling: An Integrated Environment Supporting Co-Execution of Tasks and Systems Models. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '10*. New York, New York, USA: ACM Press, p. 165.

BARON, M., LUCQUIAUD, V., AUTARD, D. AND SCAPIN, D., 2006. K-MADe : un environnement pour le noyau du modèle de description de l'activité. In J.-M. Robert & B. David, eds. *IHM'06*. Montréal, Canada: ACM Publishers, pp. 287–288.

BASTIDE, R., NAVARRE, D. AND PALANQUE, P., 2002. A model-based tool for interactive prototyping of highly interactive applications. In *CHI '02 extended abstracts on Human factors in computing systems - CHI '02*. New York, New York, USA: ACM Press, pp. 516–517.

BAUMEISTER, L.K., JOHN, B.E. AND BYRNE, M.D., 2000. A comparison of tools for building GOMS models. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '00*. New York, New York, USA: ACM Press, pp. 502–509.

BIERE, M., BOMSDORF, B. AND SZWILLUS, G., 1999. The Visual Task Model Builder. In J. Vanderdonkt & A. Puerta, eds. *Third Conference on Computer-Aided Design of User Interfaces (CADUI'99)*. Louvain-la-neuve, Belgique: Kluwer Academic Publishers, pp. 245–256.

BIERE, M., BOMSDORF, B. AND SZWILLUS, G., 1999. The Visual Task Model Builder. In *Proceedings of the third international conference on Computer-aided design of user interfaces*. Norwell, MA, USA: Kluwer Academic Publishers, pp. 245–256.

CAFFIAU, S., GUITTET, L., SCAPIN, D.L. AND SANOU, L., 2008. Utiliser les outils de simulation des modèles de tâches pour la validation des besoins utilisateur : une revue des problèmes. In *ERGO'IA*. Biarritz, France, pp. 257–258.

CAFFIAU, S., SCAPIN, D.L. AND SANOU, L., 2008. Retour d'Expérience en Enseignement de la Modélisation de Tâches. In *ERGO'IA*. Biarritz, pp. 135–143.

CARD, S., MORAN, T. AND NEWELL, A., 1983. *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates.

DELOUIS, I. AND PIERRET, C., 1991. Emad : Manuel de référence .

DIAPER, D., 1989. Task Analysis for Knowledge Descriptions (TAKD): The method and an example. In D. Diaper, ed. *Task analysis for human- computer interaction*. Chichester, UK: Ellis Horwood, pp. 108–159.

DIAPER, D., 2004. Understanding Task Analysis for Human-Computer Interaction. In D. Diaper & N. Stanton, eds. *The Handbook of Task Analysis for Human-Computer Interaction*. Mahwah: Lawrence Erlbaum Associates, Inc., pp. 5–48.

GAMBOA, R.F., SCAPIN, D.L., HANSMANN, W., HEWITT, W.T. AND PURGATHOFER, W., 1997. Editing MAD* task description for specifying user interfaces, at both semantic and presentation levels. In M. D. Harrison & J. C. Torres, eds. *Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'97)*. Granada, Spain: Springer-Verlag, pp. 193–208.

GIESE, M., MISTRZYK, T., PFAU, A., SZWILLUS, G. AND DETTEN, M. VON, 2008. AMBOSS: A Task Modeling Approach for Safety-Critical Systems. In P. Forbrig & F. Paternò, eds. *Engineering Interactive Systems (HCSE 2008 and TAMODIA 2008)*. Pisa, Italy: Springer (LNCS 5247), pp. 98–109.

GIRARD, S. ET AL., 2013. *Artificial Intelligence in Education* H. C. Lane, K. Yacef, J. Mostow, & P. Pavlik, eds., Berlin, Heidelberg: Springer Berlin Heidelberg.

HIX, D. AND HARTSON, H.R., 1993. *Developping user interfaces: Ensuring usability through product & process*, Newyork, USA: John Wiley & Sons, inc.

JOHN, B.E. AND KIERAS, D.E., 1996. The GOMS Family of User Interface Analysis Techniques: Comparaison and Contrast. *ACM Transactions on Computer-Human Interaction*, 3(4), pp.320–351.

JOHNSON, H. AND JOHNSON, P., 1991. Task Knowledge Structures: Psychological basis and integration into system design. *Acta Psychologica*, 78, p.3-26.

JOHNSON, P., JOHNSON, H., WADDINGTON, R. AND SHOULS, A., 1988. Task-related knowledge structures: analysis, modelling and application. In *Proceedings of the Fourth Conference of the British Computer Society on People and computers IV*. Cambridge University Press, pp. 35–62.

JOHNSON, P., WILSON, S., MARKOPOULOS, P. AND PYCOCK, J., 1993. ADEPT: Advanced Design Environment for Prototyping with Task Models. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93*. New York, New York, USA: ACM Press, p. 56.

JOURDE, F., LAURILLAU, Y. AND NIGAY, L., 2010a. COMM notation for specifying collaborative and multimodal interactive systems. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '10*. New York, New York, USA: ACM Press, pp. 125–134.

JOURDE, F., LAURILLAU, Y. AND NIGAY, L., 2010b. e-COMM, un éditeur pour spécifier l'interaction multimodale et multiutilisateur. In *Conference Internationale Francophone sur I'Interaction Homme-Machine on - IHM '10*. New York, New York, USA: ACM Press, pp. 225–228.

KIERAS, D.E., 2004. GOMS Models for Task Analysis. In D. Diaper & N. Stanton, eds. *The handbook of Task Analysis*. pp. 83–116.

LACHAUME, T., CAFFIAU, S., GIRARD, P., FOUSSE, A. AND GUITTET, L., 2012. Comparaison de différentes approches de simulation dans les modèles de tâches. In *Ergo-IHM'2012*. Biarritz, France: ACM, pp. 60–67.

LACHAUME, T., GIRARD, P., GUITTET, L. AND FOUSSE, A., 2012. ProtoTask, new task model simulator. In M. Winckler, P. Forbrig, & R. Bernhaupt, eds. *Human-Centered Software Engineering*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 323–330.

LIMBOURG, Q., VANDERDONCKT, J., MICHOTTE, B., BOUILLON, L. AND LOPEZ-JAQUERO, V., 2005. UsiXML: a Language Supporting Multi-Path Development of User Interfaces | UsiXML - USer Interface eXtended Markup Language. In R. Bastide, P. Palanque, & J. Roth, eds. *Engineering Human Computer Interaction and Interactive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 200–220.

MARTINIE, C., 2011. *Une approche à base de modèles synergiques pour la prise en compte simultanée de l'utilisabilité, la fiabilité et l'opérabilité des systèmes interactifs critiques*. Toulouse, France.

MARTINIE, C., PALANQUE, P., NAVARRE, D. AND BARBONI, E., 2012. A development process for usable large scale interactive critical systems: application to satellite ground segments. In M. Winckler, P. Forbrig, & R. Bernhaupt, eds. *HCSE'12 Proceedings of the 4th international conference on Human-Centered Software Engineering*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 72–93.

MORI, G., PATERNO, F. AND SANTORO, C., 2002. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, 28(8), pp.797–813.

PATERNO, F., 1999. *Model-Based Design and Evaluation of Interactive Applications*, Springer.

PATERNO, F., 2002. Tools for Task Modelling: Where we are, Where we are headed. In *TAMODIA '02 Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design*. INFOREC Publishing House Bucharest, pp. 10–17.

PATERNO, F., FACONTI, G.P. AND COMPUTER, P. AND, 1992. On the LOTOS use to describe graphical interaction. In Cambridge University Press, pp. 155–173.

PATERNO, F., MANCINI, C. AND MENICONI, S., 1997. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *IFIP TC13 human-computer interaction conference (INTERACT'97)*. Sydney, Australia, pp. 362–369.

PATERNO, F., MORI, G. AND GALIMBERTI, R., 2001. CTTE: An Environment for Analysis and Development of Task Models of Cooperative Applications. In *ACM CHI 2001*. Seattle: ACM Press.

PATERNO, F., SANTORO, C. AND SPANO, L.D., 2012. Improving support for visual task modelling. In M. Winckler, P. Forbrig, & R. Bernhaupt, eds. *HCSE'12 Proceedings of the 4th international conference on Human-Centered Software Engineering*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 299–306.

SCAPIN, D.L. AND PIERRET-GOLBREICH, C., 1990. Towards a method for task description : MAD. In L. Berliguet & D. Berthelette, eds. *Working with Uisplay units*. Elsevier Science Publishers, North-Holland, pp. 371–380.

SEBILLOTE, S., 1992. Task analysis and formalization according to MAD: Hierarchical task analysis, method of data gathering and examples of task description.

SEBILLOTTE, S. AND SCAPIN, D.L., 1994. From Users' Task Knowledge to High-Level Interface Specification. *International Journal of Human-Computer Interaction*, 6(1), pp.1–15.

STUART, J. AND PENN, R., 2004. TaskArchitect: taking the work out of task analysis. In *Proceedings of the 3rd annual conference on Task models and diagrams - TAMODIA '04*. New York, New York, USA: ACM Press, p. 145_154.

SYSTEMS, I.S.O.I.P., 1984. Definition of the Temporal Ordering Specification Language LOTOS.

TABARY, D. AND ABED, M., 2002. A software environment task object-oriented design (ETOOD). *Journal of Systems and Software*, 60(2), pp.129–140.

TABARY, D., ABED, M. AND KOLSKI, C., 2000. Object-oriented modelling of manual, automatic, interactive task in mono or multi-user contexts using the TOOD method. In Z. Binder, ed. *IFAC 2nd Conference on Management and Control of Production and Logistics (MCPL 2000)*. Amsterdam: Elsevier Science Publisher, pp. 101–111.

TARBY, J.-C. AND BARTHET, M.-F., 1996. The Diane+ method. In J. Vanderdonckt, ed. *Proceedings of the Second International Workshop on Computer-aided Design of User Interfaces (CADUI '96)*. Namur, Belgique: Presses Universitaires de Namur, pp. 95–119.

VAN DER VEER, G.C., 1996. GTA: Groupware Task Analysis - Modeling Complexity. *Acta Psychologica*, 91, pp.297–322.

VAN WELIE, M., VAN DER VEER, G.C. AND ELIËNS, A., 1998. Euterpe - Tool support for analyzing cooperative environments. In *Ninth European Conference on Cognitive Ergonomics*. Limerick, Ireland.

WURDEL, M., SINNIG, D. AND FORBRIG, P., 2008. CTML: Domain and Task Modeling for Collaborative Environments. *Journal of Universal Computer Science*, 14(19), pp.3188–3201.

Thomas Lachaume is preparing his PhD in Computer Science, in the Data Engineering and Human Computer Interaction team of the Laboratory of Computer Science and Automatic Control for Systems (LIAS). He is working more specifically on task models, and task simulators. He developed ProtoTask, a new paradigm of task simulation for task models.

Laurent Guittet is Assistant Professor at the ISAE-ENSMA engineering school, located in Poitiers, France. He belongs to the Data Engineering and Human Computer Interaction team of the LIAS Laboratory. He has been working on software engineering aspects of Human Computer Interaction, and more specifically on architecture models for HCI, HCI for teaching, and task models

Patrick Girard is Professor at the University of Poitiers, France, and head of the HCI group of the Data Engineering and Human Computer Interaction team of the Laboratory of Computer Science and Automatic Control for Systems (LIAS). His main research interests focus on End-User Programming and Task Modelling.

Allan Fousse is Assistant Professor at the University of Poitiers, France, and belongs to the Data Engineering and Human Computer Interaction team of the Laboratory of Computer Science and Automatic Control for Systems (LIAS). Coming from Software Engineering and Data Imaging, he is working now on task modelling and post-WIMP interfaces.