End User Designing of Complex Task Models for Complex Control-Command Systems

Olga Goubali[1(✉)], Patrick Girard[2], Laurent Guittet[2], Alain Bignon[1], Djamal Kesraoui[1], Soraya Kesraoui-Mesli[1], Pascal Berruet[3], Benjamin Morio[1], and Laurianne Boulhic[1]

[1] SEGULA Technologies, BP 50256, 56602 Lanester Cedex, France
{olga.goubali, djamal.kesraoui, soraya.kesraoui, laurianne.boulhic}@segula.fr, bignon.alain@neuf.fr, morio.benjamin56@gmail.com
[2] LIAS/ENSMA, 1 avenue Clément Ader, 86961 Chasseneuil, France
{girard, laurent.guittet}@ensma.fr
[3] Lab-STICC, BP 92116, 56321 Lorient Cedex, France
pascal.berruet@univ-ubs.fr

Abstract. In the design of control-command systems, task models are useful for collecting requirements about the use of the systems. Indeed, task models describe actions the supervisor performs to start, to control and to monitor the system events (alerts, performance messages), to evaluate, to stop a function (if needed), etc. Depending on the state of the supervised system (for example, presence of defects), these tasks can be numerous, repetitive and complicated. This complexity makes it difficult to describe task models, which are therefore complex, but essential for design. Task models are usually described by HMI designers using dedicated modeling tools and can be validated with business experts.

In this paper, we propose a specification process that enables to best capture system expert knowledge and to facilitate obtaining complex task models based on their operational expert knowledge. Our approach aims at formalizing and designing industrial system tasks models in our tool named Prototask Editor User. With our tool, system experts who are not tasks specialists can read and adapt the tasks pattern while simulating, verifying and validating it.

Keywords: Tasks pattern · Complex tasks model · Industrial system · EUD

1 Introduction

In the design of interactive systems, taking into account users through tasks models is inevitable because, for designers, it avoids a lot of errors related to the use of the system. Indeed, tasks models help to understand the human activity and the use of the system to be designed. With this model, HMI designers can express human tasks based on activity analysis to meet design needs. The basic principles of task models do not differ much from one tasks model to another [1]. The use of task models in a multi-disciplinary design has become increasingly popular [2].

M. Kurosu (Ed.): HCII 2019, LNCS 11566, pp. 114–130, 2019.
https://doi.org/10.1007/978-3-030-22646-6_9

Tasks are usually described according to structuring mechanisms. These mechanisms provide temporal operators that allow a hierarchical decomposition of tasks into sub-tasks, with categorization according to the type of tasks. For a more precise description of the activity, it is possible to add attributes to the tasks. These attributes can be optional, iterative, pre-condition and post-condition.

In designing sociotechnical and complex systems, a task leads to a goal (state of the desired system) that the user wishes to achieve by using a procedure that describes the means to achieve that goal [3]. Tasks analysis provides a set of data that can be used in design approaches. It is now accepted that task models are fully integrated into the design process of control-command systems. Faced with the complexity of these systems, where the user tasks must be precise and context-sensitive, the design of task models becomes more time-consuming. In fact, with the actions to be defined, the states of the system to be monitored (alarms, elements, faults, etc.), the possible actions depending on the context, the decision-making etc., the design of tasks models is laborious. In addition to that, the detection and correction of errors during the simulation of these models can delay the design of the control-command system. Indeed, tasks models are usually designed by the HMI designers and then simulated with the business expert until they are validated.

To solve these problems, existing approaches offer pattern-based solutions. Patterns provide a reusable structure for task models allowing designers to focus on the user needs.

Although tasks patterns facilitate tasks models designing, their definition is often not easy [4]. In order to facilitate the design of complex tasks models, we propose in this paper: an approach that allows to formalize and design industrial system tasks pattern; and a tool named Prototask Editor User. Based on the principle of End User Development, our tool contains modules enabling the designer to change the name of tasks, add a basic task, or delete a basic or complex task, in addition of the tasks simulation principles of the tasks simulator named Prototask [5]. In our tool, the adaptation process is performed jointly with the tasks pattern simulation.

To validate our approach, we evaluate the usefulness of on a real case study. We also evaluate our approach with tasks specialists to take into account their needs and the usability of our tool with practitioners' designer. Results are analyzed and detailed in this paper.

2 State of the Art

2.1 Analysis of Operators Activities

The description of user activities on a control-command system in tasks models, is based on the formalization of specifications. This formalization is firstly based on an analysis and a review of the literature [6–8] and secondly on feedback acquired through the analysis of specifications established in real projects and through interviews with expert designers. With this formalization, it is possible to identify necessary information contained in specifications, which must be modeled as tasks. The modeling of

these information in tasks, brings out global recursive actions, necessary to control a control-command system.

The supervision of control-command systems consists in carrying out a certain number of commands which are described as “complex” and classified in four categories [7]. The first category is about monitoring of the system functions. This task is the main one for the system operators and takes an important part of their working time. The second category concerns the manipulation of orders. This task can be performed at any time (for example: when the system is in permanent regime, when faults are detected on the system, etc.). In a functional context, for most control systems, supervisory operations occur when a function starts, when the function stops normally, or when faults are returned [7]. The third category is the diagnosis of defects. This category of tasks occurs when a defect is identified by the supervisory operator. The fourth category relates to administrative tasks (e.g. surveillance tests, verification of technical specifications, journal maintenance and event reports). These four categories of tasks detailed in [9] are closely related in a control room.

Bovell, Carter and Beck presented a flow that shows the connection between the four categories described [7]. However, this flow does not allow to accurately take into account hierarchical organization of operator activities.

Kluge presented a table containing the sequences of actions which are performed by supervisory operators [8]. However, with this table, it is not possible to structure these actions with the necessary precision, to carry out the complex and various operators tasks in exceptional situations [10].

2.2 Designing of Complex Task Models

Typically, supervisory task templates take into account the specificities of each high-level function of control systems. One can imagine the complexity in the design of such models. Several works have attempted to provide a solution to this complexity. Therefore, during the SIGCHI’97 pattern workshop [11], participants considered patterns as a way to solve complexity and diversity problems increasing in the design of HMI. The main purpose of patterns is to create an inventory of solutions to help HMI designers to solve common and difficult design problems. A pattern is defined as a solution description format for recurring design problems [12].

The study of [4] is among the first work that integrated patterns in task modeling. They were motivated by the possibility of reusing good design solutions to solve recurring problems related to dialogue specification, in order to reduce design time. Their work has highlighted the use of task structures to speed up the process of building tasks models and integrating them into the design of large-scale industrial applications.

The PSA Framework [12] uses patterns throughout the software development cycle to extend reusability across all design models. This framework offers a reusability solution and allows designers to capture and propagate information between the proposed patterns (tasks, domain, structure and navigation patterns, etc.). However, the concept of linking patterns in a design is not addressed in this approach [13]. In addition, the approach does not address the transition from task patterns to task models.

Gaffar and his colleagues approach emphasizes another important aspect of the notion of pattern: the combination of patterns [14]. By combining different patterns,

developers can use pattern relationships and combine them to produce an effective design solution. This approach offers a solution and a tool to facilitate the integration and adaptation of patterns into tasks models, then the transition from tasks models to concrete interfaces. In their work, several patterns have been proposed to represent generic tasks models and techniques to transform them into a concrete interface. However, this tool uses dialog patterns in addition to tasks patterns and when the user wants to achieve a single goal, he has to take several actions and make several decisions consecutively. The manipulation of patterns is therefore fastidious [13]. In addition, to be interpreted by the tool, tasks patterns must be expressed in XIML exclusively. To use the tasks models described in CTT, the authors use Dialog graph editor [15] to convert CTT files into XIML files.

The PD-MBUI Framework [16] offers a very interesting approach for instantiating and adapting all models involved in the design process, not just task models. However, this approach is based on a library of formalized and predefined patterns which are not usable in all areas of design. The authors plan to extend the modeling concept into an integrated pattern environment which will support their tool and others based on generalized patterns which will be independent of any platform and programming language.

In [13], a pattern language was constructed for the Smart meeting room domain. This language can assist the designer when he is building models and thus improve the design process. Tasks patterns belonging to this language allow integration of task fragments as a design block within the user's tasks model. The whole methodology is adaptable to the design of similar patterns in other domains. In addition, the consideration of pre-conditions and post-conditions in the definition of visualization constraints related to tasks patterns provides a precision in the description of the latter. However, the lack of tools does not facilitate use of this language.

3 Problem Description

The description that is made of user activities on a control system, in task models is a good illustration of the complexity of these models. Previous work has shown that using task patterns is a good solution.

The simple graphical representation of the supervisory tasks pattern as proposed in [7], presents only the tasks tree, but no information on the decomposition of tasks, operators, implementers, etc. The TPML language and its variants make it possible to represent the patterns graphically and textually, but the instantiation of the patterns is done with specific tools that exploit these languages. A task model that integrates these patterns must be described either in CTT [17], or in XIML. Other tasks modeling languages are not currently taken into account. However, the use of notation such as CTT to structure the complex activity of operators in supervision is difficult [10]. It is preferable to be able to describe patterns in any tasks modeling language. Therefore, the first challenge identified is the definition of complex task patterns. This requires an approach to build the patterns that describe the tasks of an operator in front of a control system. This also requires identification and formalization of the so-called tasks.

On the other hand, current approaches that exploit tasks patterns can reduce the complexity of designing tasks models. These approaches offer tools based on the use of task patterns which are stored in a database, for the description of task models. The limits of these tools are at two levels. First, it's very difficult to imagine all the possible patterns for storing them into a Framework. In fact, patterns designed in a warehouse are not suitable for all areas. This problem was encountered when one was trying to build task models for a "smart environment" for example [13] and a new pattern language adapted to this domain had to be built.

Some of the work presented provides tools for moving from tasks patterns to tasks models. These tools are generally used by HMI designers because they contain concepts that are still complex for end users (expert users). Once the tasks models are achieved, they are simulated with the end users who have operational knowledge, to verify and validate them. During the simulation, detected errors can lead to important modifications in the models. These changes may affect design time of the control system. The second identified challenge is enabling end users to build complex tasks models by using tasks patterns.

4 Proposition

Our approach aims at facilitating the design of complex tasks models. The transition from generic tasks patterns to tasks models is a way to promote reusability and to facilitate the definition of tasks models. Figure 1 describes the process used to define the tasks pattern and to transform it into tasks models. This process is detailed into three phases. During the first phase, we formalize and design the supervision tasks pattern, by analyzing state of the art and functional specifications information. At this stage, we analyze the typical situations which may exist in different applications, and the information that could be used to determine the instances of the patterns. Then, we choose TPML [14] as the representation formalism. This choice allows HMI designers to understand the pattern. This formalism contains a modular section which enables to represent the pattern tree structure. For the graphical representation, we use this formalism through the K-MADe task modeling tool [18], to describe the supervision tasks pattern. The advantage of this representation is to be able to take into account information about tasks decomposition, tasks operators, tasks performers, preconditions, event, etc. at the pattern level.

It would be better to use the pattern directly without going through transforming processes from a pattern language to a task model representation language. However, these processes are unavoidable if we want to go from one task notation to another. In order to facilitate these processes, we formalize and define a generic meta-model so call ATS (Anaxagore Task Structure), which standardizes the information contained in the different tasks notations.

For implementing the model transformations, it is necessary to identify the notation in which the pattern is described. This identification enables to execute the appropriate transformation and to obtain a uniform tasks pattern.

The standardization of tasks patterns facilitates the transition to tasks models, through our developed tool. Our tool is based on the techniques of End User Development since it allows business expert to design tasks models without training.

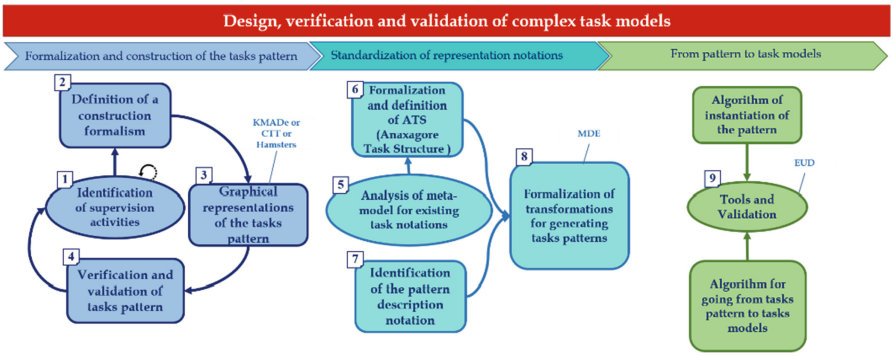


Fig. 1. Our approach

5 Implementation and Results

5.1 Pattern Construction Formalism

The description of the TPML formalism fields (Table 1) to adapt it to our context facilitates the design of the supervisory task pattern. In this formalism, classic fields such as Name, Problem, Context, Solution, and Relationship are described to help the designer understand the pattern. The graphic representation shows the tree structure of the pattern.

Table 1. Representation formalism of tasks patterns based on [19]

Field	Description
Name	Pattern name
Problem	Description of the problem solved by the pattern e.g. it allows designing task models for high-level function of control-command system
Context	Description of precondition that will be verified before applying the pattern
Solution	Description of the problem solution by presenting the procedure to follow
Relation	List of the pattern(s) to which this pattern is linked
Representation	Graphical representation of the solution

5.2 Graphical Representation of Tasks Pattern

We describe tasks pattern with K-MADe tool. The tasks patterns are generic tasks models. Unlike tasks models, task patterns must contain variables that can be adapted

to different contexts of use, allowing it to be specialized in specific tasks models. For the pattern to be as generic and as flexible as possible, the description of the hierarchical structure of the tasks must include, at least, one variable (for example, instruction (n1) on Fig. 2). Variables contained in the pattern are tasks name and tasks descriptions. Generic tasks names are those that change from one function to another.

The complete tasks pattern for describing operators activities on a complex industrial system is presented in Annex 4 of [9].

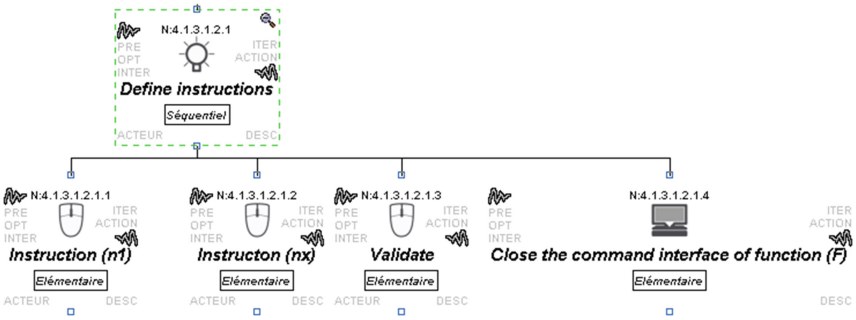


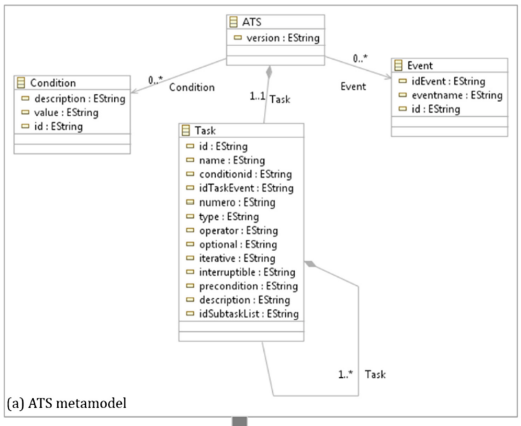
Fig. 2. Extract of tasks pattern

To make our approach generic, we used the PIM (Platform Independent Model) principle: the patterns whose descriptions contain variables can be specified with modeling tools other than K-MAD. HMI designers can represent these patterns with any task modeling notation (KMAD, CTT, Hamsters, etc.). However, to facilitate the exploitation of the patterns in our approach, we defined a phase to uniform the tasks representation notation.

5.3 Standardization of Tasks Modeling Language

To be independent from task modeling notations, an important step before getting tasks models is to transform the tasks pattern written in any task modeling notation into a platform-independent generic notation. For this, we have described a task meta-model (Fig. 3a) named ATS (Anaxagore Task Structure) that gathers all the information contained in all existing task notations.

The current created models transformations enable to convert the pattern described with K-MADe into a generic tasks pattern (Fig. 3b). These transformations will be further completed to take into account other task modeling notations, such as CTT and Hamsters. A new model transformation will only be created for using a new task notation, for the first time.



(a) ATS metamodel

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="ATS">
  <PostCondition idPostCondition="K542" PostConditionName="Marche" value="false"/>
  <Condition description="In monitoring" value="INDETERMINATE" id="K33"/>
  <Condition description="The instruction is validated" value="INDETERMINATE" id="K61"/>
  <Condition description="Fonction in progress" value="INDETERMINATE" id="K476"/>
  <Task id="K19" name="Handling of planned orders" numero="4" type="ABS" operator="ALT"
optional="false" iterative="[1]" interruptible="false" description="" idSubtaskList="K76 K79 K407 ">
  <Task id="K4" name="monitor the system" idPostCondition="K542 " numero="Racine" type="ABS"
operator="PAR" optional="false" iterative="[1]" interruptible="false" description="" idSubtaskList="K7 K186 K16 K19 ">
```

(b) Extract of the task pattern according to the ATS metamodel

Fig. 3. Extract of the task pattern according to the ATS metamodel

5.4 From Task Pattern to Task Models

Once the generic tasks pattern is obtained, it can be reused for designing tasks models of most of control-command systems. Indeed, this pattern can be instantiated to be adapted to the specificities of the function of which one wants to describe the tasks model. The instantiation and adaptation are carried out with the Prototask Editor User tool (Prototask EU). The adaptation process is performed jointly with the simulation of tasks pattern. Different algorithms have been implemented to take into account all task model characteristics [9].

The developed interface gradually displays the tasks pattern loaded in the Prototask EU tool (Fig. 4), to enable the user to adapt it. In fact, the gradual display of tasks is carried out like task simulators. Simulation and adaptation are therefore closely linked.

The proposed interface is in four zones. The main one (frame 1 in Fig. 4) displays the tasks pattern gradually according to tasks hierarchical organization. It contains widgets for changing the name of an active task, changing the description of an active task, deleting an active task, or adding a basic subtask. Frame 2 displays the historic, as the user progresses into the task tree. Frame 3 groups all preconditions and post-conditions defined during the creation of tasks pattern. Finally, Frame 4 appears when the user clicks on the button “Ajouter une nouvelle tâche” (in English “Add a new task”). All required information needed to add a task must then be entered by the user.

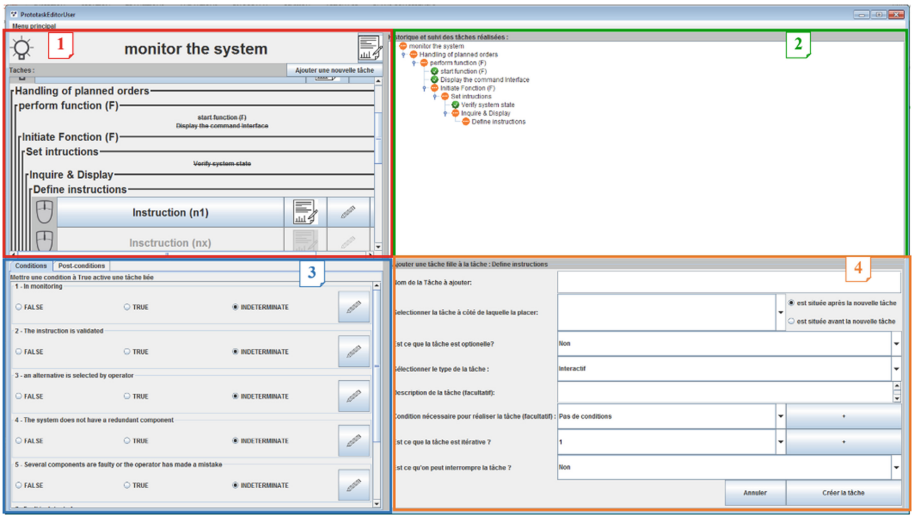


Fig. 4. Prototask Editor User

6 Validation

Our approach and the associated tool were evaluated in three different ways:

- By experimentation, to validate if our approach and tool are helpful to obtain complex tasks model of any control-command system. This validation is carried out by developer teams of our tool who have relied on specifications of seven high-level functions of a control-command system.
- By tasks specialists, who were consulted in unstructured interviews to validate if our approach and tool (Prototask EU) are relevant in designing of complex tasks models. We also validate if the usage of Prototask EU tool facilitates the description of complex tasks models in real project.
- By practitioners, first to validate if end user can use Prototask EU tool to describe tasks models; then to evaluate the actual usability of the interface.

6.1 Experimental Validation: Case Study

To validate our approach, we applied it for designing the tasks models of seven high-level functions of an industrial case study. The latter, is a system for the production, storage and distribution of fresh water, onboard a ship, called EdS (Eau douce Sanitaire in French, sanitary freshwater in English). Here we describe some steps of the adaptation process to present the different Prototask EU widgets and their usefulness. In Fig. 4, we simulated the tasks pattern up to the task “Fill instructions” to illustrate the adaptation to the tasks model design for the transfer function of EdS system.

The “pencil” icon (to edit) and “trash” icon (to delete) are active on achievable tasks. The pencil icon enables the user to modify the task name if needed. The “trash” icon enables the user to delete a task not required in the current tasks model. It is important to note that deleting a complex task removes all subtasks related to it.

The “paper pencil” icon enables the user to modify or enter a description for the active task. Adapting the pattern to obtain the tasks model of the transfer function exploits all widgets to modify preconditions, task names, and delete unnecessary tasks for this function. At the end of this operation, we obtain the tasks model (an extract of this model is shown in Fig. 5) which is verified and validated for the transfer function.

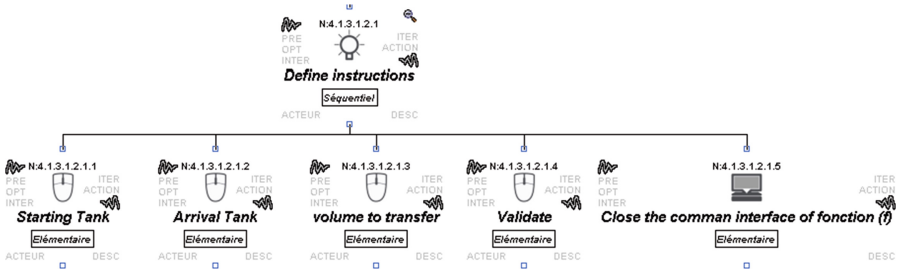


Fig. 5. Extract of the tasks model adapted for the transfer function

The same process of adaptation was followed to obtain all seven high-level tasks models of the EdS system in one hour and ten minutes instead of fifteen hours when using classical task modelling tool. The tasks models obtained this way contain required information for the specification of the EdS functions.

Our tool makes the task tree completely transparent to the expert user (system expert) who has no knowledge of task notations. It therefore gives the user the ability to customize the task pattern while simulating it, and to obtain checked and validated task models.

6.2 Tasks Specialist Validation

To evaluate the usefulness of our tool, we have asked two computer scientists (tasks specialists) for specifying two complex tasks models (composed of 64 tasks and 8 decomposition levels) with our tool and with an existing tool (K-MADe). Each tasks specialist had to specify the model with both tools. Time required for specification of the models with each tool was recorded, and the tasks specialists were asked to answer three questions about their overall impressions.

The results showed that the tasks specialists spent more time to specify the tasks models with the conventional tool (K-MADe) than with our tool (Table 2), five times more on average.

Table 2. Specification times with both tools

	K-MADe	Prototask Editor
Tasks model n°1	Tasks specialist 1: 00:46:24 Tasks specialist 2: 00:52:11	Tasks specialist 1: 00:08:48 Tasks specialist 2: 00:15:01
Tasks model n°2	Tasks specialist 1: 01:12:17 Tasks specialist 2: 00:56:14	Tasks specialist 1: 00:08:26 Tasks specialist 2: 00:16:31

They explained that the use of the pattern in our tool saved them a lot of time because they didn't need to start from the beginning for each model. With K-MADe, they had to rebuild the tree for each model and found it tedious. The integration of the simulation with the conception was also an advantage in terms of specification time. For them, the major advantage of K-MADe is the possibility to see and interact with the tree diagram. In fact, the absence of the diagram in our tool was disturbing for complex tasks models and tasks specialists felt lost in the specification process at some points.

Regarding tasks simulation in the tools, the tasks specialists explained that the frame showing the tasks during the simulation decreases systematically with the increase of the model complexity (number of tasks). Indeed, the number of displayed windows in K-MADe increases with the number of tasks. With Prototask EU (our tool), the number of lines (one for each task) increases with the number of tasks. They mentioned that this problem seems to be a major issue common to all tasks modelling tools in the case of complex tasks models designing. They therefore suggested it would be interesting to improve our tool for trying to solve this problem.

On the other hand, simulation is an unavoidable step in the verification and validation of task models. The tasks specialists appreciated the possibility of simulating the whole model in Prototask EU and not just a single scenario like with conventional task simulators such as K-MADe. For them, the complete simulation of the model offered by Prototask EU allows a considerable gain in terms of quality and completeness of the model. Indeed, they explained that our tool forces them to realize all the scenarios and thus to validate all the model and not only some parts. The tasks specialists mentioned that the Prototask EU does not allow them to deeply modify the tasks tree contrary to K-MADe. Indeed, Prototask EU tool allows the modification and addition of new single task. It does not enable to add a complex task with the choice of tasks operators, tasks types, etc. They suggested to add more functionalities (edition, insertion, deletion of operators) to Prototask EU tool for tasks specialists.

In summary, tasks specialists preferred to use Prototask EU for its ease of use, the completeness of its simulation and also the gain in quality (reduction of errors) and the time saving.

6.3 Practitioner Validation

To identify potential usability problems of our tool, we have solicited five plan designers for individual user tests sessions with our tool. They were men between 25 and 57 years old (32.8 years old on average). They didn't have previous experience in tasks models design and specification.

The user tests consisted of four steps. First, the context of the study was presented to the participants (objective of the study and of the tool, some technical notions about task models that were important for the understanding of the tool, and organization of the test session). Then, they were asked to use the tool to specify two simple tasks models (the first one was composed of four tasks, the second of six tasks, both models with two decomposition levels). The screen and the voice of the participants were recorded with their agreement during this step. After that, we asked them some questions to know their first impressions with the tool. Finally, they completed the Post

Study System Usability Questionnaire (PSSUQ) and a form with personal information such as age or profession.

These user tests identified some usability problems. We identify three minor problems. First, the function of creating a new task is misplaced in the tool. This creates confusion for some elements (for example, placing the new task against existing tasks) and mixes mandatory and optional fields in the same place, which unnecessarily complicates the creation process. We also observed problems of title and placement for some functionalities that were not clear for the participants (e.g. confusion between the description button and the modification button). Furthermore, they pointed out that the historic frame was not interactive. Indeed, they can only view their progression and they wished they could go back to previous tasks through it. We also identifies two major usability problems that we are going to present with further details.

The first one concerns the preconditions frame. During the test sessions, the participants had to activate a precondition for some of the tasks in the models before any modifications. We have reported the number of clicks participants have done for activating the precondition associated with the task they intend to specify, and compared it to the minimal number of clicks needed to activate a precondition with our tool (here, this number is two clicks). The Fig. 6 shows the results. The first time, the participants clicked 8.2 times on average (min = 5; max = 10), that represents 720% of additional clicks. Observation of the participants when they were using the tool showed that they haven't spontaneously thought to activate the precondition of the task the first time. In fact, they were clicking randomly on the interface and reported that they felt lost. Moreover, they had difficulties to identify which precondition is associated with which task. Indeed, four of the participants have first activated the wrong condition. However, for the other occurrences of this type of action, we observed a significant reduction of the number of clicks by the participants, which shows that they learnt how to do this type of action.

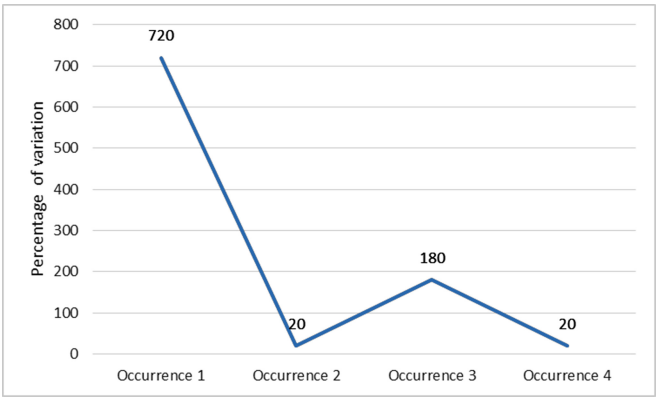


Fig. 6. Percentage of clicks number variations to activate the good precondition compared to minimal number of clicks needed

The second major problem concerns the mode of simulation, which consists in clicking on the task button to go to the next one. We observed that, the first time, participants made 3 clicks in average (min = 1; max = 7) while this action only requires one click the shortest way; that represents 200% of additional clicks (Fig. 7). Indeed, during the tests sessions, the participants didn't understand that they had to click on the task to simulate it and go to the next one. They found it by clicking at different places on the interface. The number of clicks decreased significantly the second time the action was required and reached the minimum number necessary for this action. We can conclude that, even if not very intuitive at the beginning, the minimum sequence of clicks quickly becomes automatic.

Nevertheless, the participants explained that, with this functioning, it is very easy to make an error. Some of them have expressed exasperation reactions after clicking involuntarily on the task button, and others were surprised by the appearance of the task when it is simulated (i.e. the task name is strikethrough). It seems that they thought they made a mistake while this was entirely correct.

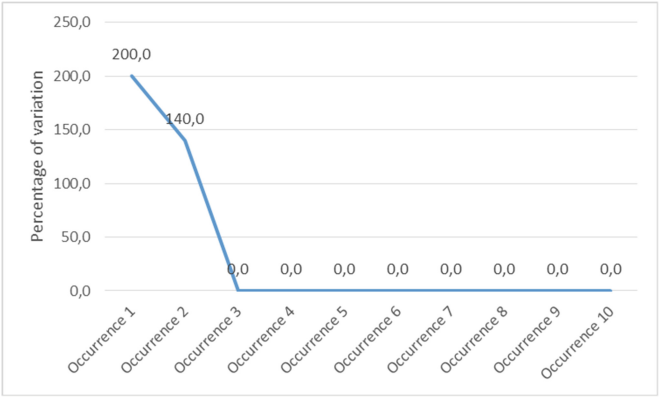


Fig. 7. Percentage of clicks number variations made to simulate the task compared to minimal number of clicks needed

Despite these problems, answers to the PSSUQ showed that our tool has a satisfying overall usability quality (Fig. 8). In this questionnaire, lower is the score, better is the quality of the tool. The overall usability and the other sub-dimensions of the questionnaire average scores are comprised between two and three, which is significantly superior to the mean.

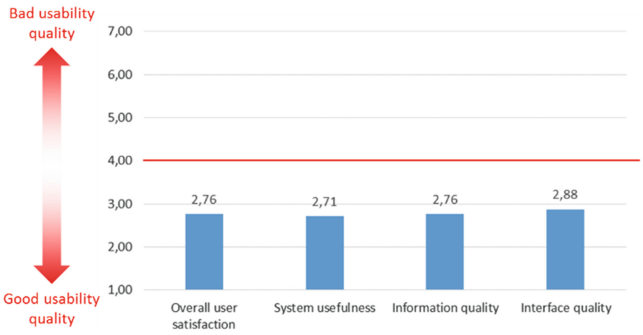


Fig. 8. PSSUQ dimensions average scores

6.4 Discussion

The validation of our tool showed that the presence of the pattern is an important advantage for specifying complex tasks models. We have demonstrated through experimental validation the applicability and the utility of our tool for the specification of complex task models. This validation, carried out on an industrial study, made it possible to specify complex task models in less time than other conventional tools. However, the correction of complex errors detected during tasks patterns simulation or the addition of complex tasks to the tasks pattern or model leads to starting again at the design phase of the pattern and repeating some steps our approaches. This may delay the design of the control system. This problem was also pointed by tasks specialists during their validation. Indeed, they mentioned the limits of our tool since it does not allow them to add complex tasks tree with appropriate tasks operators.

Tasks specialists validation has also shown that our tool is a great help in ensuring quality when specifying complex task models. The simulation of the entire task tree ensures the completeness of the tests and thus guarantees the quality of the validation of the tasks models. Using pattern also reduces design errors and avoids repetitive creation steps for redundant tasks. Tasks specialists also appreciated the integration of simulation into the adaptation phase (design). For them, this avoids interface changes for designing and simulation of tasks models compared to conventional tools. Nevertheless, they explained that they need a tool which takes into account more tasks modeling features such as operators, types for designing complex tasks. The information collected from analysis of tasks specialists validation will be used to propose them another tool which takes into account these features since different end users do not have the same needs.

To take into account end users needs, we have carried out practitioner validation. This validation made it possible to check if users who are not tasks specialists can use the tool for designing their tasks models. The practitioners' validation showed that the usability quality of our tool is satisfying. This is because the participants learned quickly when using the interface for the first time (designing the first task template). The design of the second task model was therefore quick since they understood the operation of the interface. The participants declared that the interface remains a tool of work which they must understand but after the first use it becomes easy. The usability

problems detected were identified during the first use of the interface. The information from this validation will be used to improve the interface and make it more intuitive for the user.

7 Conclusion and Future Works

The presented work provides additional support over existing approaches for designing complex tasks models. Our goal is to reduce designing time and effort while getting verified and validated complex tasks models through an interface which can easily be used by our end users, i.e. non-tasks specialists.

The formalization, construction and use of tasks patterns have enabled us to solve the complexity and diversity problems that are increasing in the description of industrial system tasks models. Indeed, the identification of the structure of the supervisory tasks, based on state of the art and real design projects, led to the construction of a pattern which regroups the operator activities in a control room in a generic way.

The use of proven software for tasks modeling (K-MADe) to design tasks pattern by leaning on the TPML, enabled a better description of these patterns. The implementation of model transformations enables to overcome the constraints related to the choice of task notation. With our approach, the pattern can be designed with any task notation. This step may require large model transformations, but these transformations are only implemented when using a new task notation for the first time.

The implementation of Prototask Editor User tool to support our approach enables system experts (an expert user who are not task specialists) to read and adapt the tasks pattern for getting different tasks models.

The three types of validation carried out demonstrated the usefulness of our approach and the usability of the developed tool. Experimental validation enabled us to get seven complex tasks models of EdS system functions, in one hour and ten minutes instead of fifteen hours when using classical task modeling tool. In addition, our approach can be used in the design of new control-command systems.

Tasks specialist validation has highlighted the benefits of our approach for specifying complex task models. The time required for adaptation is five times less on our tool than on traditional tools. The accuracy and quality of the specified models are also improved by our tool which allows the simulation of the whole model and not only some scenarios. However, tasks specialists have brought out the need for more complex modifications in tasks patterns. Today, to widely modify tasks pattern, tasks specialists need to go back to their task notation tool (e.g. K-MADe), validate a new tasks pattern, and repeat the model transformation phase before using our tool.

Practitioners' validation showed that the usability quality of our tool is satisfying but can be improved to be more intuitive for the users.

In future work, we will solve the various usability problems of our tool and we will propose a tool for tasks specialists to enable them to widely modify task pattern and task models.

Acknowledgements. We are grateful to Line Poinel for her help.

References

1. Lachaume, T., Guittet, L., Girard, P., Fousse, A.: Task model simulators: a review. J. d'Ineraction Pers. **3** (2014)
2. Lewandowski, A., Bourguin, G., Tarby, J.-C.: De l'Orienté Objet à l'Orienté Tâches – Des modèles embarqués pour l' intégration et le traçage d'un nouveau type de composants. Rev. d'Interaction Homme-Machine **8**, 1–33 (2007)
3. Normand, V.: Le modèle SIROCO: de la spécification conceptuelle des interfaces utilisateur à leur réalisation. Joseph-Fourier - Grenoble I (1992)
4. Breedvelt-schouten, I.M., Paternò, F., Severijns, C.: Reusable structures in task models. Des. Specif. Verif. Interact. Syst. **97**, 225–239 (1997)
5. Thomas, L., Patrick, G., Laurent, G., Allan, F.: ProtoTask, new task model simulator. In: Winckler, M., Forbrig, P., Bernhaupt, R. (eds.) HCSE 2012. LNCS, vol. 7623, pp. 323–330. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34347-6_24
6. AFNOR X50-151: Management par la valeur et ses outils, analyse fonctionnelle, analyse de la valeur, conception a objectif designe. French National Standards, Ed.
7. Bovell, C.R., Carter, R.J., Beck, M.G.: Nuclear power plant control room operator control and monitoring tasks. Oak Ridge, TN, July 1998
8. Kluge, A.: Controlling complex technical systems: the control room operator's tasks in process industries. In: The Acquisition of Knowledge and Skills for Taskwork and Teamwork to Control Complex Technical Systems, pp. 11–47. Springer, Dordrecht (2014). https://doi.org/10.1007/978-94-007-5049-4_2
9. Goubali, O.: Apport des techniques de programmation par démonstration dans une démarche de génération automatique d'applicatifs de Contrôle-Commande. ENSMA (2017)
10. Martinie, C.: 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. université de Toulouse (2011)
11. Bayle, E., et al.: Putting it all together: pattern languages for interaction design. A CHI 97 workshop. In: CHI 1997 Workshop, vol. 30, no. 1, pp. 17–23 (1998)
12. Granlund, Å., Lafrenière, D., Carr, D.A.: A pattern-supported approach to the user interface design process. In: HCI International 2001 9th International Conference on Human-Computer Interaction, vol. 1, pp. 282–286 (2001)
13. Seffah, A.: Patterns of HCI Design and HCI Design of Patterns: Bridging HCI Design and Model-Driven Software Engineering. HIS. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15687-3
14. Gaffar, A., Sinnig, D., Seffah, A., Forbrig, P.: Modeling patterns for task models. In: Proceedings of the 3rd Annual Conference on Task Models and Diagrams, vol. 2, no. 1, pp. 99–104 (2004)
15. Wolff, A., Forbrig, P., Dittmar, A., Reichart, D.: Linking GUI elements to tasks - supporting an evolutionary design process. In: 4th International Workshop on Task Models and Diagrams, pp. 27–34 (2005)
16. Radeke, F., Forbrig, P., Seffah, A., Sinnig, D.: PIM tool: support for pattern-driven and model-based UI development. In: Coninx, K., Luyten, K., Schneider, K.A. (eds.) TAMODIA 2006. LNCS, vol. 4385, pp. 82–96. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70816-2_7

17. Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: a diagrammatic notation for specifying task models. In: Howard, S., Hammond, J., Lindgaard, G. (eds.) Human-Computer Interaction INTERACT 1997. ITIFIP, pp. 362–369. Springer, Boston, MA (1997). https://doi.org/10.1007/978-0-387-35175-9_58
18. Baron, M., Lucquiaud, V., Autard, D., Scapin, D.L.: K-MADe: un environnement pour le noyau du modèle de description de l'activité. In: IHM 2006, vol. 133, pp. 287–288 (2006)
19. Sinnig, D.: The complicity of patterns and model-based UI development. Concordia, Montreal, Quebec, Canada (2004)