Rapid Task-Models Development Using Sub-models, Sub-routines and Generic Components

Peter Forbrig[1], Célia Martinie[2], Philippe Palanque[2], Marco Winckler[2], and Racim Fahssi[2]

[1] University of Rostock, Department of Computer Science, Albert Einstein Str. 21, Rostock, Germany
peter.forbrig@uni-rostock.de
[2] ICS-IRIT, University of Toulouse 3, 118, route de Narbonne 31062 Toulouse Cedex 9, France
{martinie,palanque,winckler,fahssi}@irit.fr

Abstract. Whilst task models are perceived as critical artifacts within User Centered Design methods, task models development is often considered as a resource and time consuming activity. Structuring techniques can support handling issues such as reuse and scalability and can improve analysts' productivity and the overall quality of models. In this paper we propose (based on the notation of the HAMSTERS project) several means to structure task models and present how they can be used in order to increase reusability and scalability in task models. Besides sub-models and sub-routines, generic components are described. These mechanisms are duly illustrated within a project for the ground segments of satellite missions. This paper shows, by example, how such elements look like and how both readability and quality of models is improved by their use.

Keywords: Generic components, sub-models, sub-routines, task models.

1 Introduction

It is widely accepted that models help to explore and understand new domains by providing insights into the domain at an abstract level, avoiding going too early into details and supporting communication between the various stakeholders (by providing unambiguous descriptions). For designing interactive systems, task models are a valuable source of knowledge as they represent large quantity of information related to user goals and to the activities to be carried out in order to reach those goals. Task models are particularly useful when describing complex activities for which informal descriptions (e.g. natural language text) are not manageable. Task models contain information that can be used to assess the complexity of activities in terms of number of actions to be performed, knowledge and skills required to carry out activities, information and objects required to accomplish tasks. By providing unambiguous descriptions, task models can be used to check the consistency of the information described and to detect modeling mistakes. Moreover, task models help analysts reason about activities, for example, about possible migration of tasks from the user to the system.

S. Sauer et al. (Eds.): HCSE 2014, LNCS 8742, pp. 144–163, 2014.

Despite all the advantages of task models, if the activities to be represented are complex, the resulting models might become complex too. Complexity in models is a recurrent problem with model-based approaches that might require significant availability of resources which is sometimes perceived as too much effort, too long to produce and not being cost effective enough. This might be true if every model has to be developed from scratch each time a new application is considered and if the modeling techniques are not equipped with adequate tool support. Moreover, beyond tool-support, model complexity is also a concern at the notation level.

In [15], a detailed argumentation is presented about the fact that abstraction and refinement of task models is not sufficient for handling large real world applications. For that two mechanisms were proposed (sub-models and sub-routines) to deal with complexity in task models. In the present paper we revisit these mechanisms and we propose a third one (component) that provides a powerful mean for reusing models parts. These three mechanisms are aimed at supporting rapid task-model development by structuring models and improving reuse of existing models. All these mechanisms are illustrated using the notation HAMSTERS which has been fully integrated in the eponym project and tool support [17]. These mechanisms can be used with other task model notations, but HAMSTERS already has them fully embedded. A full description of other objects (data, information, objects, knowledge...) in a HAMSTERS task model can be found in [16].

2 Three Mechanisms for Supporting Structuring and Reuse in Task Models

This section presents three mechanisms for structuring task models: sub-models, sub-routines and components.

2.1 Sub-models

Sub-models are based on the refinement/abstraction principle and make possible to define elementary reusable bricks in task models. A large task model can thus be decomposed into several duplications of elementary tasks (called sub-models). These sub-models can then be reused (as a kind of “copy”) in various places of the same model and even in other models. Each time one of the attributes of these elementary sub-models is modified, the modification is reflected in all the other “copies” of the same sub-model.

Characteristics

While task notations propose reuse at the class level (an example of such a class being a “motor task type”) the sub-model proposes reuse at the instance level. For example, if in a task model, a task “push button” appears many times (because moving the lever can be performed by users for reaching multiple goals, such as to change gears in a car to reduce or to increase velocity) the sub-model construct makes it possible to handle those instances altogether.

Advantages and Limitations

- The gathering of elementary tasks into a set of which all the elements can be manipulated at once. It reduces viscosity as changing one attribute of a task (e.g. name) is automatically reflected to the entire set it belongs to.
- Explicit representation of identical activities. The sub-models allow analysts to represent in an explicit manner the fact that several task models involve exactly the same elementary tasks.
- This mechanism does not provide support for reducing the size of the models.
- This mechanism provides support for reusing recurrent activities previously recorded. In a given domain (e.g. aeronautics) the activities of the operators are usually identified and codified. Sub-models make it easy to explicitly build a library of elementary tasks that can then be quickly reused while building models that correspond to interactions with new devices or systems.

Description in HAMSTERS

Fig. 1 illustrates sub-models in HAMSTERS (Fig. 2 b).depicts the task types available in the HAMSTERS notation). Copy tasks (such as the bottom left one "Push button" is the same sub model as the task with the same name under the disengage abstract

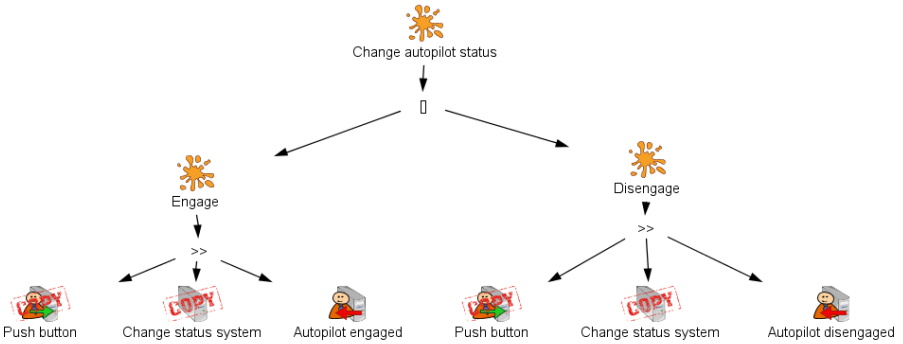
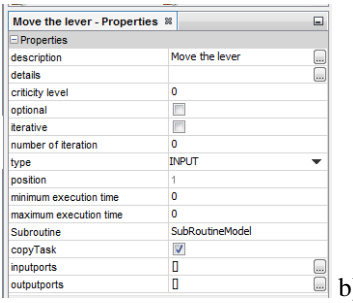


Fig. 1. Sub-models as copy tasks in HAMSTERS

a)



b)

Task type	Icons in HAMSTERS notation
Abstract	Abstract task
System	System task
User	User task, Cognitive task, Perceptive task, Motor task
Interactive	Input task, Output task, Input Output task

Fig. 2. a)Edition of a copy task in HAMSTERS, b)task types in HAMSTERS

task. All the properties of these two leaf tasks are shared and changing the type or name of one is immediately reflected on the other one. In HAMSTERS, "Copy" is an attribute of a task and it can be defined by selecting the checkbox "copy task" as shown in Fig. 2 a).

2.2 Sub-routines

Sub-routines are used to structure task models and to define information passing between task models. This mechanism is similar to procedure calls in programming languages and parameterization of the behavior is possible via input and output parameters.

Characteristics
The sub-routines aim at reusing a sub-tree in a task model. A group of tasks (represented as a tree) might have to be performed in multiple occasions with very little differences which are depending on some values and represented as pre and post conditions. The sub-routine makes it possible to describe recurring behaviors and to describe explicitly the parameters and how they influence the task model behavior.

Advantages and Limitations

- The grouping of a sub-tree of a task model into a sub-routine. It thus makes it possible to reuse the same sub-tree in one or several task models.
- The parameterization of a sub-tree by using parameters (both at input and output level) so that the sub-tree can represent a slightly different behavior. Input parameters are used as preconditions in some of the sub-tasks to describe conditional execution of tasks, while output parameters are used for describing different outcomes when tasks are executed.
- The identification of activities that operators perform in different contexts but are recurrent in their work (or the domain in which they operate). For instance, to report an incident each time it occurs might have a similar structure but it can appear as a sub-goal of another activity (monitoring the system for instance). In such case, incident reporting would not be a full task model but a sub-tree that has to be duplicated in every context where this activity has to be performed.
- This structuring mechanism reduces the models' size but, from the point of view of the HAMSTERS' user, it diminishes the global understanding of the model (the global hierarchical view is split into several models).
- Such structuring mechanism requires skills and knowledge in task engineering, in order to be able to select the sub-tree that should become a sub-routine and to be able to select and describe its parameters.

Description in HAMSTERS
A sub-routine is a group of activities that users perform several times possibly in different contexts which might exhibit different types of information flows. Fig. 3 provides the description of sub-routines. The icons for input and output parameters are filled if values are needed and computed. In the HAMSTERS CASE tool, the sub-routines are stored as task models but are gathered in the project tree under the same grouping called "subroutines" (see Fig. 9 a)). The task models are stored under the grouping called "Roles" because they specify the tasks related to a certain role.

Fig. 3. Representations of sub-routines in HAMSTERS

Fig**. 4** describes how sub-routines can be called within a task model. That model describes tasks for a ground segment application that is currently used to monitor and to control the PICARD satellite that was launched in 2010 for solar observation. More details about the system are discussed in [15]. The model of Fig**. 4** represents the fact that monitoring and controlling the PICARD system is an iterative task. Satellite monitoring and Failure detection and recovery are sub-routines with no input and output parameters. They can be executed in parallel and the second one is optional.

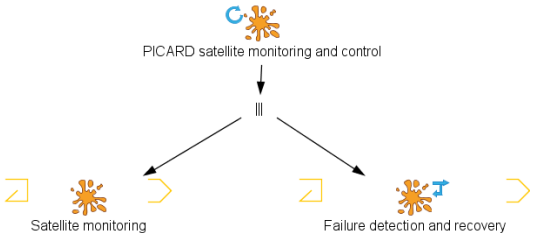


Fig. 4. Description of sub routines within a task model

Fig. **5** describes the content of a subroutine in HAMSTERS. That sub-routine called "IncidentReporting" has an input parameter (incidentLevel) and an output parameter (status). Within the task model the input parameter is used as a precondition (for instance if its value is "low" then the task LightReporting is performed). The output parameter "status" is set to "closed" when task "Register Incident is performed.

Fig. **6** presents the usage of input parameters in the sub-routines. Within the alternative tasks "Voltage failure" and "Signal failure" one can see that failure code must be an input parameter. Depending on its value VOLTAGE_FAILURE or

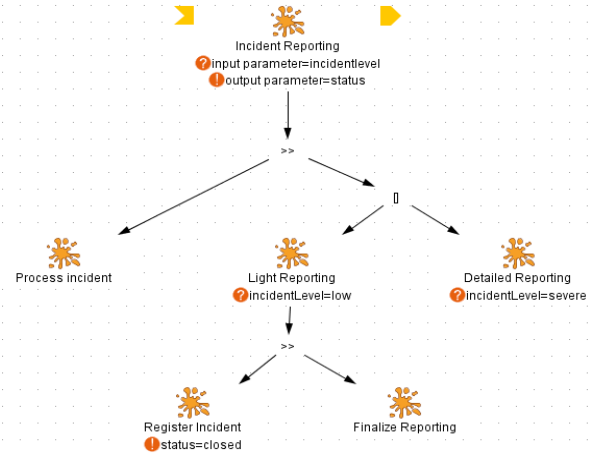


Fig. 5. Parameterization of a sub-routine

SIGNAL_FAILURE the corresponding sub-tree is executed. Both sub-trees have a very similar structure. There might be the chance to specify them with generic specifications. This will be explained within the following sub-section.

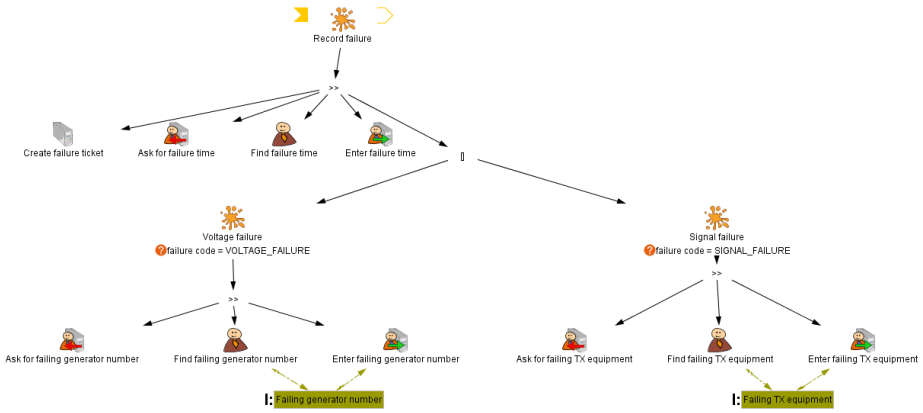


Fig. 6. Task model corresponding to the sub-routine "Record failure" (taken from [15])

Having a look at the graphical specification of the task "Record failure" (Fig. 6) one can see some further features offered by the HAMSTERS notation that have been introduced in [16]. The rectangles with an "I:" at their left describe the information required to perform the tasks that are connected to them. For example, the human task "Find failing generator number" produces the information "Failing generator number" which is then consumed by the interactive input task "Enter failing generator number".

2.3 Generic Components

The third mechanism for reuse is based on the notion of reusable components. The concept of sub-routines provides parameters for tuning the reuse of task models, but does not provide mechanisms for defining generic features of sub-models. Fig. 7 presents a generic task component that provides a general solution for both sub-trees of the tasks "Voltage failure" and "Signal failure" in Fig. **6**.

Characteristics

The goal of using generic components for task models is to allow for reuse of modeling efforts in a more general way than sub-routines.

Advantages and Limitations

- The grouping of a collection of sub-trees of task models into a component. It thus makes it possible to reuse similar sub-trees in one or several task models.
- Identification of repetitive activities within different contexts and context-dependent presentation of the models.
- Adaptation of the components can be performed based on the parameters during design time and during runtime.

- This structuring mechanism reduces the models' size but, from the point of view of the HAMSTERS' user, it diminishes the global understanding of the model (the global hierarchical view is split into several models).
- Such structuring mechanism requires skills and knowledge in task engineering, in order to be able to select the sub-tree that should become a generic component and to be able to select and describe its parameters.

Design time instantiation means that a designer of a new model uses generic components to build a new task model. Generic parameters are substituted and the resulting instances are inserted into the main model.

Description in HAMSTERS

The component of Fig. **7** is generic in such a way that names of sub-tasks and information (notation element tagged with an "I:" and representing information produced and/or required to accomplish a task [16]) are based on the generic parameters.

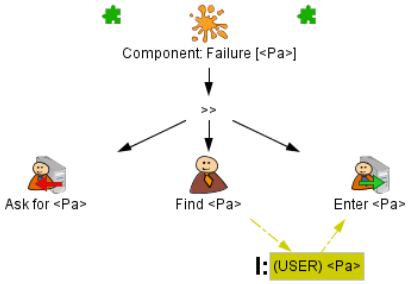


Fig. 7. Generic task component for "Failure"

This parameter can be substituted during design time, when a larger task model is built. It can also be substituted during runtime while executing the different parts of a model. In this case, a generic component instance is instantiated with the values of the generic parameters that are provided interactively.

Assuming that the generic parameter <Pa> gets the value "failing generator number" during design time, the task model of Fig. **8** is the result of building an instance of the task component of Fig. **7**.

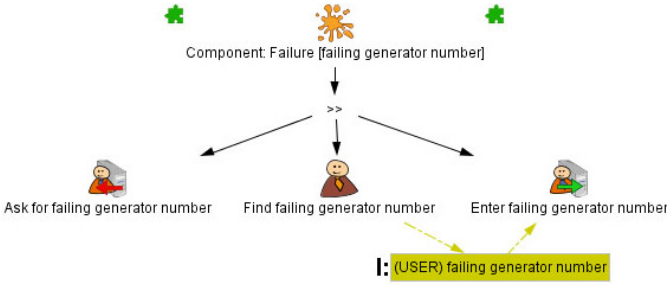


Fig. 8. Instance of component "Failure" with parameter value "failing generator number"

2.4 Patterns and Task Patterns

Patterns have a long history both in the area of software engineering (e.g. [11] for the definition of the Model-View-Controller pattern and [10] for the first collection of patterns). They have been reused from practice in the area of buildings architecture where they were introduced by C. Alexander et al. [2].

The essence of patterns is that they represent reusable solutions to a recurring type of problem. For instance, in the area of buildings, steps organized in groups are a pattern for making it possible for people to go from a level to another one (higher or lower).

This concept has already been applied to task analysis and even task modeling. In such case they are mainly used (as for software engineering) to define a generic solution to a given problem. Paternó et al. in [5] have proposed a pattern called "Multi-values input Task" where a set of iterative tasks are interrupted by a termination task. Such construction deserves the term of patterns as they are supposed to help the person in charge of modeling reuse this solution (a combination of the iterative operator with the interruption one) to describe users' tasks. Other approaches have also been proposed in [8], [9], [19] or [24] in order to provide reusable elements corresponding to solutions to recurring problems. The structuring mechanisms presented in this paper are of a different nature. Indeed, they are introduced for addressing scalability and modifiability of task models. Indeed, we don't present how they help modeling user tasks but instead how they can be used for structuring task models in order to ease their understanding, the modification and their reuse in other contexts. These two approaches (structuring mechanisms and patterns) are even orthogonal. Indeed, one could define design patterns without any structuring concepts (as in [19] for instance) or use structuring mechanisms for tasks models without addressing the aspect of patterns to solve recurring problems (as in [15] for instance). Of course, those two concepts could be integrated but this is beyond the scope of this paper.

3 Process of Identifying Reusable Components

Sub-models and sub-routines can be identified based on knowledge of the application domain. Sub-models are created for modification purposes mainly. This can be considered as grouping on the lexical level. On the contrary sub-routines can be considered as specifications at the level of semantics of the domain. They are used for structuring reoccurring activities with and without parameters related to functions or procedures.

Generic components of models are tools for reuse. They are on a different "direction" of abstraction. They or their instances can be on the one hand side part of sub-models or sub-routines. On the other hand they can contain sub-models and sub-routines.

From our point of view generic components can be considered to be specifications on the syntactic level. It seems to us that they only can be developed based on existing sub-models and sub-routines. This was at least the case in our project and is conform to the definition of design patterns in the object-oriented world.

Sub-models and sub-routines were already available while the task models for satellite observation were designed. In this way those opportunities for reuse already found their way into the models. We did not find further opportunities for those language elements after reviewing the resulting models.

However, the idea of generic components was not available at the beginning of the project. The identification of reusable generic components was based on carefully studies of the designed models. As a result several generic components were identified. Their instances had to be applicable at least at two different places of the task models.

At the beginning components with only one generic parameter were identified. Their instantiation was thought to be at design time. Later components with more parameters were specified that allowed more complex manipulations of task trees during instantiation.

The extension of the HAMSTERS environment by runtime instantiation of generic components was a result of intensive discussions after different complex components were specified. Short examples extracted from the case study of the controllers' activities performed during the PICARD satellite mission and are presented in this paper.

4 The Three Structuring Mechanisms in the Hamsters Case Tool

This section presents how the structuring mechanisms can be used in the HAMSTERS CASE tool [12].

4.1 Editing

Creating a new component is performed the same way as creating a task model. The difference is that the file containing the description of the generic component belongs to the "Component" folder in the project tree (as illustrated in Fig. **9**a)). When building a task model, a generic component can be added by a drag and drop of a "Component Task" from the Palette (Fig. 9b)).

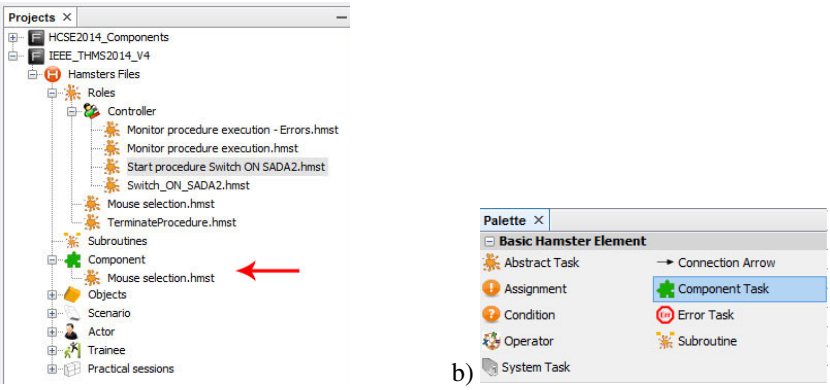


Fig. 9. a) Representation of the generic components directory in the project tree b) Representation of the palette in HAMSTERS

In order to instantiate a new instance of a component, the generic component task has to be selected and a right click has to be performed on this selection (illustrated in Fig. 10). After the pop-up menu displays, the user has to select the "Instantiate Component" item, which opens a dialog (illustrated in Fig. 11).

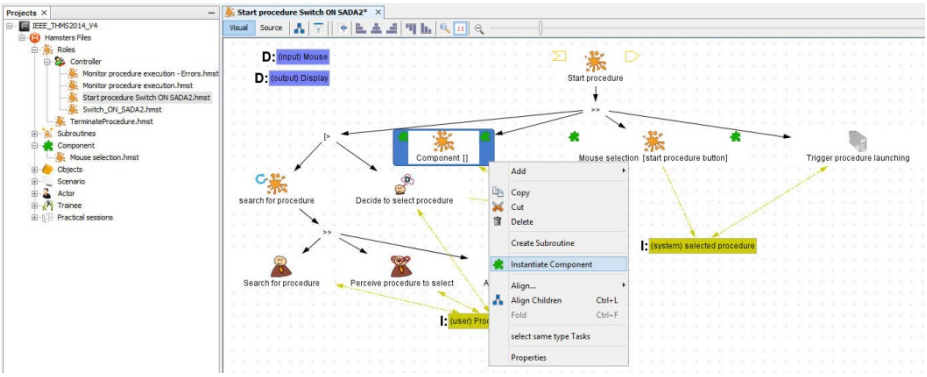


Fig. 10. Instantiation of a generic component

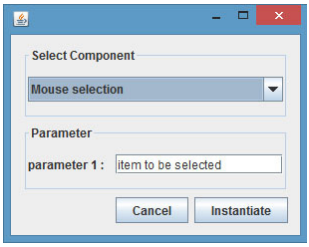


Fig. 11. Selection of the parameters for the instance of the generic component

This dialog window provides support to select the targeted generic component, and to enter the parameters. Once the “Instantiate” button has been pushed, a new instance is created (illustrated in Fig. 13) in the project tree (illustrated in Fig. 12).

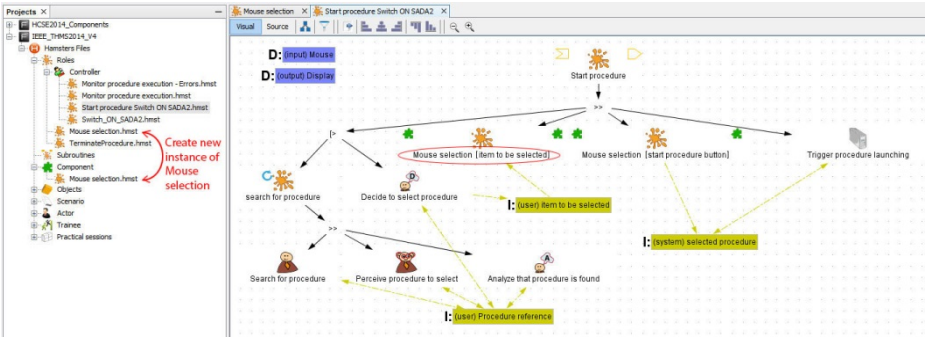


Fig. 12. New instance of the “Mouse selection” generic component in the current project

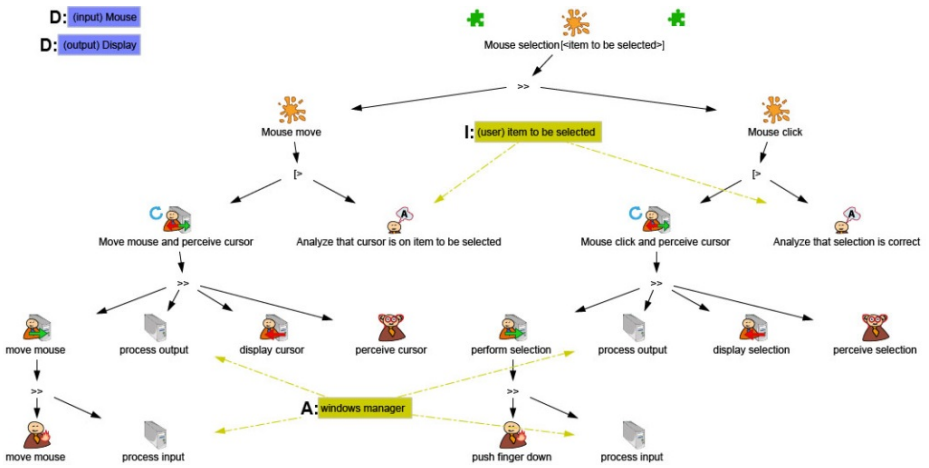


Fig. 13. Instance of "Mouse selection" generic component

4.2 Simulation

The HAMSTERS CASE tool provides support for executing task models. HAMSTERS provides support for editing and simulating task models related to collaborative activities [14]. Task models can be associated to roles and a main task model (corresponding to the highest goal in the hierarchy) has to be defined for each role in the project. The simulator executes all the task models referenced in each role's main task model as well as in its sub-models, sub-routines and generic components in a recursive way. Fig. 14 depicts the execution of a sub-routine and its generic components.

5 Case Study

This section presents a case study from space segments and aims at illustrating the various mechanisms introduced above. The context of ground segment operations as well as operators' main activities have been presented in [15]. This case study is used to compare the benefits of the use of the proposed mechanisms for structuring and reuse over large and complex task models. Fig. **15** presents a Bird's eye view on the task model of the PICARD satellite monitoring and control activities.

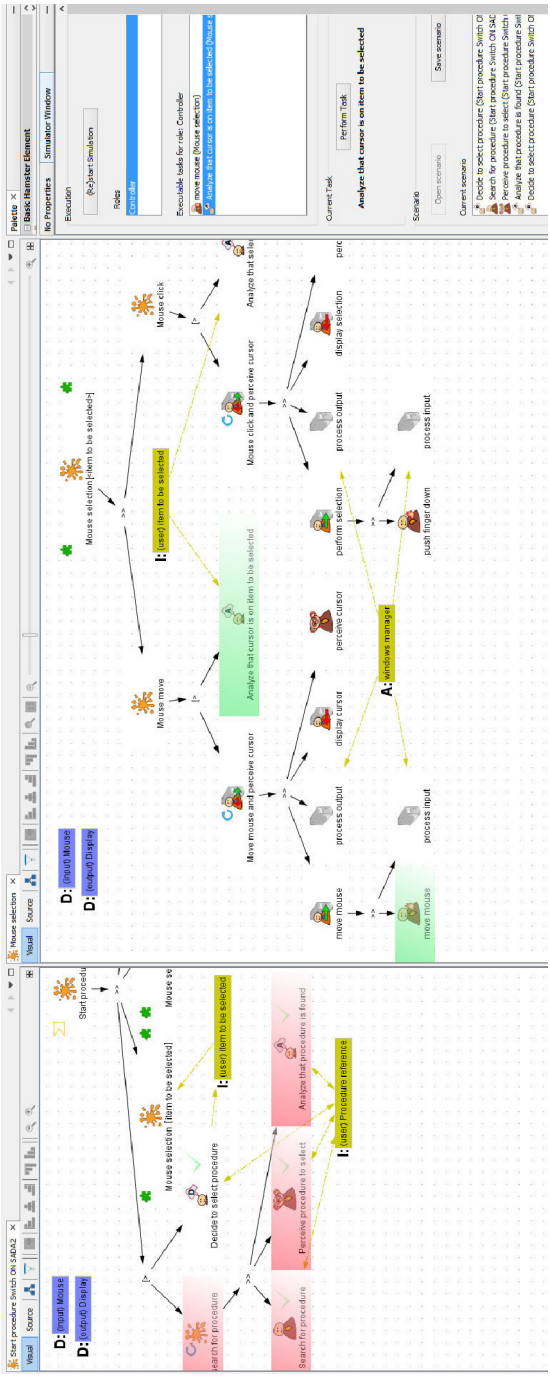


Fig. 14. Simulation of the models in HAMSTERS

Fig. 15. Bird's eye view on the task model without structuring mechanisms

Assuming that the generic parameter <Pa> gets the value "failing Tx equipment" during design time, the task model of Fig. 16 is the result of building an instance of the task component of Fig. **7**.

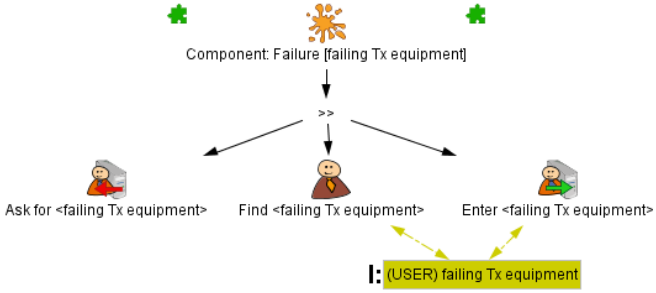


Fig. 16. Instance of component "Failure" with parameter value "failing Tx equipment"

The original model "Record failure" that is presented by Fig. **6** can be reformatted by using the introduced generic component (illustrated in Fig. 17). The generic component is applied and instantiated twice.

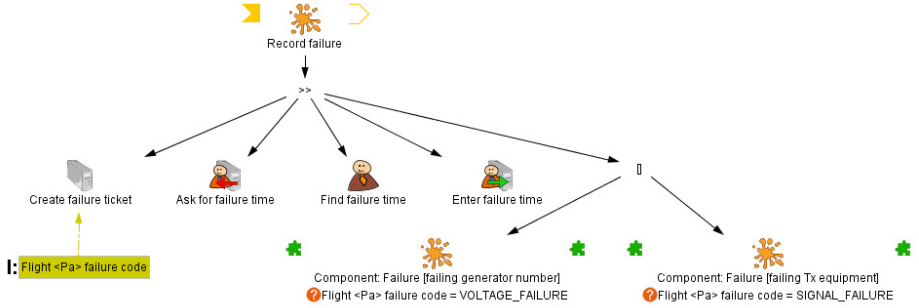


Fig. 17. Sub-routine "Record failure"

Avoiding Mistakes by Applying Components

Additional investigations of specified models within the PICARD project showed further opportunities of using generic components. It also demonstrated that such

reusable elements can help to avoid modeling mistakes. Within the following model of sub-routine "Failure detection and recovery" in Fig. 19, another potential generic component was identified. However, slight differences of two sub-trees existed. The two related tasks are "Generator voltage issue" and "Communication failure 2nd try" (in Fig. 15). Indeed, there is a sub-task "Create failure token" that is part of the second task but not of the first one. Nevertheless, it came out that this sub-task was only forgotten to be modeled in the first case and should be added there as well.

By a detailed analysis it was realized that the differences were based on minor modeling mistakes and the identified potential component was applicable twice. In this way the careful studying of the possible application of reusable components helps to identify modeling problems as well. The generic component identified from the model in Fig. 19 is presented in Fig. 18. The rewritten model using the component is presented in Fig. 20.

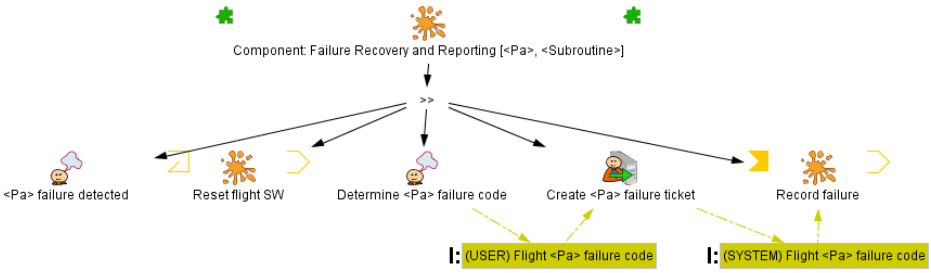


Fig. 18. Generic component "Failure Recovery and Reporting"

An instance of the specified reusable component "Failure Recovery and Reporting" with parameters *voltage* and *Reset flight SW* is presented in Fig. 19.

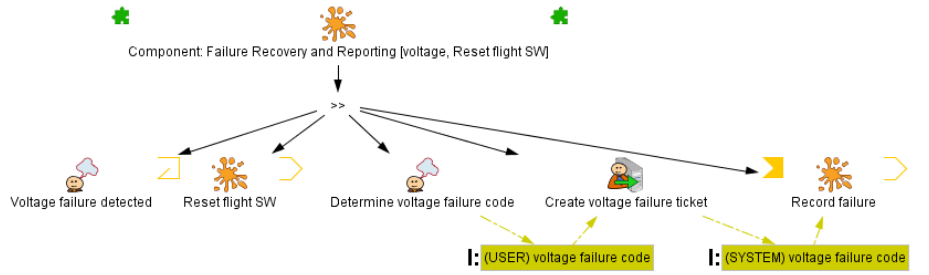


Fig. 19. Instance of component "Failure Recovery and Reporting"

The model presented in Fig. 20 is more compact than the previous version which was not using the generic component structuring mechanism.

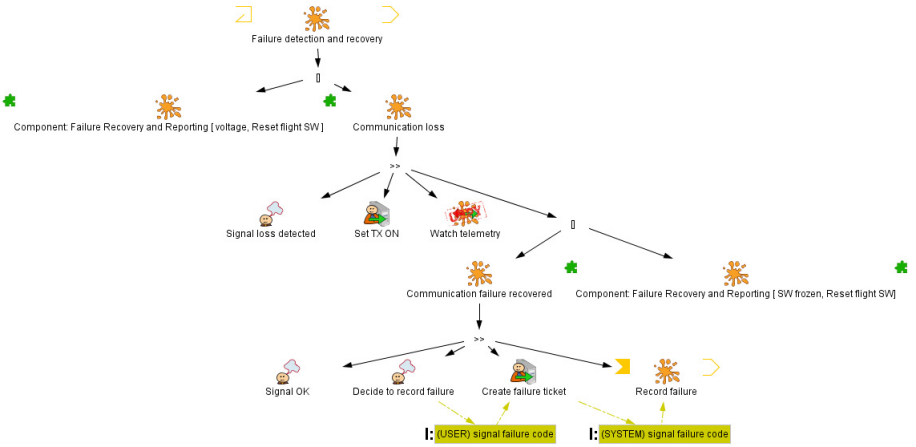


Fig. 20. Sub-routine "Failure detection and recovery" with usage of components

This approach provides support for avoiding the minor modeling mistake of missing out a specific task that is recurrent in the models. This can happen if a modeler has to specify the same type of sub-model several times.

In existing task modeling CASE tools there are no features for reusing models in such a way. We argue for implementing generic components in modeling tools like in HAMSTERS. The provided examples are represented with HAMSTERS. However, the discussed principles can be applied to different notations as well. The presented components are still very simple.

Runtime Instantiation

We already mentioned in section 4 the different options of generic component instantiation during design time and runtime. This is related to the different kinds of execution of decisions during design time or during runtime. In case of instantiating the above generic component during design time the contained decisions can already be executed.

However, a combined strategy could be used as well. For example, in Fig. 18, the parameter <Pa> could be instantiated with the value "voltage" and the parameter instantiation for <subroutine> could be postponed to runtime. In this way the value of the parameter <subroutine> has to be provided interactively during runtime.

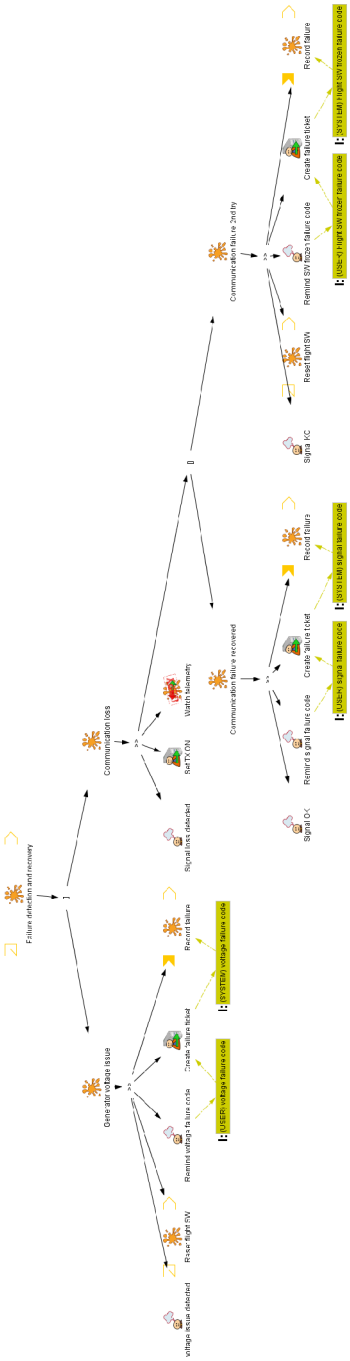


Fig. 21. Sub-routine “Failure detection and recovery”

Benefits and Concerns

This section summarizes quantitative and qualitative assessment of the proposed structuring approach.

Table 1. Quantitative comparison of the approach

Structuring methods	Number of tasks	Number of operators	Reduction percentage
Model without structuring mechanisms	59	13	Reference model for calculus
Sub-models and subroutines	45	13	24% less tasks than in model without structuring mechanisms
Structuring with sub-models, subroutines and components	35	7	41% less tasks and 46% less operators than in model without structuring mechanisms

Quantitative Analysis

Table 1 presents statistics about the size of models in terms of number of tasks and number of operators. It highlights the fact that the number of tasks to be edited highly decreases when using the proposed structuring mechanisms.

Qualitative Analysis

The case study shows that the proposed structuring mechanisms provide support for representing complex set of activities related to large scale systems. The components are reused several times in a set of models related to a project but they also can be inserted to a Palette of components and be reused from one project to other ones (as presented in section 4). However, such structuring mechanisms require skills and knowledge in task engineering in order to take in hand the modeling environment and to use the structuring mechanisms.

6 Related Work

This work is mainly focused on structuring mechanisms at notation level for dealing with complexity in large task models and for supporting reuse of task model elements. Structuring mechanisms should be adapted to the appropriate level of abstraction supported by the notation and its inner construct. For example, most of task model notations such as UAN [6], CTT [21], MAD [22] and AMBOSS [1] provide hierarchical task decomposition, which enforces the abstract/refinement mechanism for dealing with complex models [5]. They also include the operator iterative tasks (symbol T* in CTT) so that repetitive tasks can be drawn once even if they are executed many times in a row. That operator is mainly used for describing the behavioral aspect of the task but additionally makes it possible to significantly reduce the size of a model if such an operator was not available. Therefore, structuring mechanisms require appropriate support at the notation level to be fully support.

An important issue that must be considered when deciding the structure of task models is its potential for reuse [5]. The reuse of software artifacts has been the subject of research for many years in the software community and it has been proved that it might reduce time for development and cost, and increase the quality of software systems. In more recent years structuring mechanisms have been proposed to support the reuse of snippets of models such as use cases [4] and Petri Nets [13].

As far as task models are a concern, some tasks (such as login into systems) remain structurally similar even when reused in different applications. This feature has been introduced in notations like CTT [20] so that some generic tasks can be used as building blocks that can be integrated along the modeling process. Concerned by the reusability of tasks models, Gaffar et al. [9] have investigated structuring mechanisms around the notion of patterns to be used in task models. They propose a method and a tool to model generic task patterns as building blocks that can be instantiated and customized when modeling real-life socio-technical systems. One of the advantages of task patterns with respect to building blocks is the fact they provide more flexibility to adapt the specification to very specific needs. Nonetheless, all these solutions for reusing generic tasks and task patterns are limited to isolated models, lacking of a notational support to describe how such snippets of models are articulated once they are integrated into larger task models.

7 Summary and Outlook

This paper has presented three structuring mechanisms for notations dedicated to task modeling. These mechanisms are sub-models, sub-routines and generic task components. While the first two ones were already described in [16], the third one is new. It was developed while studying the models that were developed in the context of the PICARD case study for possibilities of reuse.

The specified generic task components helped to improve the readability of the models but also to improve their quality. It made it possible to avoid minor modeling mistakes. These mistakes could have been identified with reviewing methods of course as well.

We have shown how generic components can be used and how they can be instantiated during design time and during runtime. Both concepts are helpful for modeling and executing specifications. It is even possible to allow both strategies for one component. Some (not all) parameters can be instantiated during design time and the rest during runtime.

While this paper focuses on the HAMSTERS notation and tool, the presented ideas can be used within other task modeling frameworks.

Sub-models, sub-routines and generic task components structuring mechanisms provide support for managing complex and numerous tasks in all task modeling environments. In the context of safety-critical systems such elements are of primary importance as the resilience of the entire socio-technical system has to be assessed prior to deployment.

References

1. Amboss, http://wwwcs.uni-paderborn.de/cs/ag-szwillus/lehre/ws05_06/PG/PGAMBOSS
2. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A pattern language: towns, buildings, construction. Oxford University Press (1977)
3. Barboni, E., Ladry, J.-F., Navarre, D., Palanque, P., Winckler, M.: Beyond modeling: an integrated environment supporting co-execution of tasks and systems models. In: Proc. of the 2nd ACM SIGCHI Symp. on Engineering Interactive Computing Systems (EICS 2010), pp. 165–174 (2010)
4. Bonilla-Morales, B., Crespo, S., Clunie, C.: Reuse of Use Cases Diagrams: An Approach based on Ontologies and Semantic Web Technologies. IJCSI 9(1(2) (January 2012) ISSN (Online): 1694-0814
5. Breedvelt, I., Paterno, F., Sereriins, C.: Reusable structures in task models. In: Harrison, M.D., Torres, J.C. (eds.) DSVIS 1997, pp. 251–265. Springer (1997)
6. Diaper, D., Stanton, N.A. (eds.): The Handbook of Task Analysis for Human-Computer Interaction, 650 p. Lawrence Erlbaum Associates (2004)
7. Dittmar, A.: More precise descriptions of temporal relations within task models. In: Paternó, F. (ed.) DSV-IS 2000. LNCS, vol. 1946, pp. 151–168. Springer, Heidelberg (2001)
8. Forbrig, P.: Interactions in Smart Environments and the Importance of Modelling. Romanian Journal of Human - Computer Interaction 5, 1–12 (2012)
9. 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 (TAMODIA 2004), pp. 99–104. ACM, New York (2004)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Object-oriented Software. Addison-Wesley (1995)
11. Goldberg, A., Robson, D.: Smalltalk-80: the Language and Its Implementations. Addison Wesley (1983)
12. HAMSTERS, http://www.irit.fr/ICS/hamsters/
13. Huber, P., Jensen, K., Shapiro, R.M.: Hierachies in Coloured Petri Nets. In: Rozenberg, G. (ed.) APN 1990. LNCS, vol. 483, pp. 313–341. Springer, Heidelberg (1991)
14. Martinie, C., Barboni, E., Navarre, D., Palanque, P., Fahssi, R., Poupart, E., Cubero-Castan, E.: Multi-Models-Based Engineering of Collaborative Systems: Application to Collision Avoidance Operations for Spacecrafts. In: Proc. EICS, pp. 45–55 (2014)
15. Martinie, C., Palanque, P., Winckler, M.: Structuring and Composition Mechanisms to Address Scalability Issues in Task Models. In: Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., Winckler, M. (eds.) INTERACT 2011, Part III. LNCS, vol. 6948, pp. 589–609. Springer, Heidelberg (2011)
16. Martinie, C., Palanque, P., Ragosta, M., Fahssi, R.: Extending Procedural Task Models by Explicit and Systematic Integration of Objects, Knowledge and Information. In: Proc. ECCE, articleno. 23, pp. 1–10 (2013)
17. Navarre, D., Paternó, F., Santoro, C., Feige, U.: A tool suite for integrating task and system models through scenarios. In: Johnson, C. (ed.) DSV-IS 2001. LNCS, vol. 2220, pp. 88–113. Springer, Heidelberg (2001)
18. Navarre, D., Palanque, P., Ladry, J., Barboni, E.: ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM TOCHI 16(4), 1–56 (2009)

19. Palanque, P., Basnyat, S.: Task Patterns For Taking Into Account In An Efficient and SystematicWay Both Standard And Erroneous User Behaviours. In: IFIP 13.5 Working Conf. HESSD, pp. 109–130. Kluwer Academic Publishers (2004)
20. Paternò, F.: CTTE: An Environment for Analysis and Development of Task Models of-Cooperative Applications. ACM CHI 2001 Extended Abstracts (2001)
21. Paternò, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proc. of Interact 1997, pp. 362–369. Chapman & Hall (1997)
22. Scapin, D.L.: K-MADe. In: COST294-MAUSE 3rd International Workshop, Review, Report and Refine Usability Evaluation Methods (R3 UEMs), Athens (March 5, 2007)
23. Sinnig, D., Wurdel, M., Forbrig, P., Chalin, P., Khendek, F.: Practical Extensions for Task Models. In: Winckler, M., Johnson, H. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 42–55. Springer, Heidelberg (2007)
24. Zaki, M., Wurdel, M., Forbrig, P.: Pattern Driven Task Model Refinement. In: International Symposium on Distributed Computing and Artificial Intelligence, DCAI 2011, Salamanca, Spain, April 6-8 (2011)