Getting SW Engineers on Board: Task Modelling with Activity Diagrams

Jens Brüning, Anke Dittmar, Peter Forbrig, Daniel Reichart

University of Rostock, Department of Computer Science Albert-Einstein-Str. 21, 18059 Rostock, Germany
{jens.bruening, anke.dittmar, peter.forbrig, daniel.reichart}@informatik.uni-rostock.de

Abstract. This paper argues for a transfer of knowledge and experience gained in task-based design to Software Engineering. A transformation of task models into activity diagrams as part of UML is proposed. By using familiar notations, software engineers might be encouraged to accept task modelling and to pay more attention to users and their tasks. Generally, different presentations of a model can help to increase its acceptance by various stakeholders. The presented approach allows both the visualization of task models as activity diagrams as well as task modelling with activity diagrams. Corresponding tool support is presented which includes the animation of task models. The tool itself was developed in a model-based way.

Keywords: HCI models and model-driven engineering, task modelling, UML

Introduction

Model-based software development has a long tradition in HCI. Approaches like Humanoid [22], Mecano [20], or TRIDENT [4] aim to provide designers with more convenient means to describe user interfaces and to supply corresponding tool support. The main idea is to use different models for specifying different aspects which seem to be relevant in user interface design. Because of the dominant role of task models the terms model-based design and task-based design are often used interchangeably (e.g. [5], [25]). In this context, task analysis provides "an idealized, normative model of the task that any computer system should support if it is to be of any use in the given domain" [14]. In other words, it is assumed that parts of task knowledge of users can be described explicitly, for example, in terms of task decomposition, goals, task domain objects, and temporal constraints between sub-tasks. It is furthermore assumed that task models can be exploited to derive system specifications, and particularly user interface specifications, which help to develop more usable and task-oriented interactive systems (e.g. [26], [19]).

However, with the emergence of MDA [17] the model-based idea is often related to the object-oriented approach. Many supporters of MDA are not even aware of the origins. It also was recognized that some software engineers have problems to accept the value of task modelling. This might be the case because task diagrams are not part of the Unified Modeling Language [24]. Taking into account that object-oriented techniques covering all phases of a software development cycle are currently the most successful approaches in Software Engineering it might be wise to integrate taskrelated techniques and tool support in order to transfer knowledge and experience from HCI to Software Engineering.

UML offers activity diagrams to describe behavioural aspects but does not prescribe how they have to be applied during the design process. They are often deployed to describe single steps of or an entire business process. We suggest to use activity diagrams

for task modelling. Although a familiar notation does not guarantee that software engineers develop a deeper understanding of user tasks it might be a step in the right direction.

The paper shows how CTT-like task models can be transformed into corresponding activity diagrams. In addition, transformation rules to some extensions of CTT models are given. This approach has several advantages. First, task analysts and designers can still apply “classical” task notations but transform them into activity diagrams to communicate with software developers or other stakeholders who prefer this notation. However, it is also possible to use activity diagrams from scratch to describe tasks. Second, activity diagrams which are structured in the proposed way can be animated. We will present corresponding tool support. Third, the comparison of task models and activity diagrams enrich our understanding of the expressiveness of both formalisms. Activity diagrams, which are structured like task models, represent a subset of all possible diagrams only. However, their simple hierarchical structure might also be useful for specifying other aspects of systems under design in a more convenient way. On the other hand, elements as used in activity diagrams (e.g. object flows) might stimulate an enrichment of current task model notations.

The paper is structured as followed. Sect. 2 presents a short introduction to task modelling and to activity diagrams as well as related work. An example is introduced, which is used and extended throughout the paper. Transformation rules for CTT-like task models and their application to the example model are given in Sect. 3.1 and 3.2. In Sect. 3.3 we discuss extensions to CTT models and their transformation. Tool support for the suggested approach is discussed in Sect. 4. Model-based development ideas were used to implement tools for handling task models and their transformation into activity diagrams as well as for animating these activity diagrams by using a taskmodel animator which we developed earlier in our group (e.g. [10]). A summary is to be found in Sect. 5.

Background

This section begins with a short overview of task modelling concepts and with an introduction of CTT as well-known task notation within the model-based design approach of interactive systems. Then, activity diagrams in UML 2.0 are introduced. Af-ter discussing related work we argue why activity diagrams seem to be a good starting point for integrating task modelling into the object-oriented design approach.

2.1 Task Modelling

The underlying assumption of all theories about tasks is that human beings use mental models to perform tasks. Task models -as cognitive models -are explicit descriptions of such mental structures. Nearly, if not all task analysis techniques (with HTA [1] as one of the first approaches) assume hierarchical task decomposition. In addition, behavioural representations are considered which control the execution of sub-tasks. TKS (Task Knowledge Structure) [14] is one of the first attempts to formalize the hierarchical and sequential character of tasks in order to make task models applicable to system design [26]. CTT (Concur Task Trees) [19] is the perhaps best known approach within the model-based community (HCI) today -thanks to corresponding tool support like CTTE [7]. In [16] a comparison of task models is to be found. The example model given in Fig. 1 shows the decomposition of task *Manage Goods Receipt* into the sub-tasks of receiving, checking, and processing single items and so on. In addition, temporal operators between sibling tasks serve to define temporal relations between sub-tasks. CTT supports the following operators (T1 and T2 are sub-tasks of T).

T1 >> T2 enabling T1 [] T2 choice
T1 ||| T2 independent concurrency T1 [=] T2 order independency
T1 [> T2 disabling/deactivation T1 |> T2 suspend-resume
[T1] optional task T1* iteration

Hence, a task model describes a set of possible (or rather, planned) sequences of basic tasks (leaves in the task tree) which can be executed to achieve the overall goal. For example, the model T = (T1 ||| T2) >> T3 would describe the sequences <T1,T2,T3> and <T2,T1,T3> (with T1,T2,T3 as basic tasks).

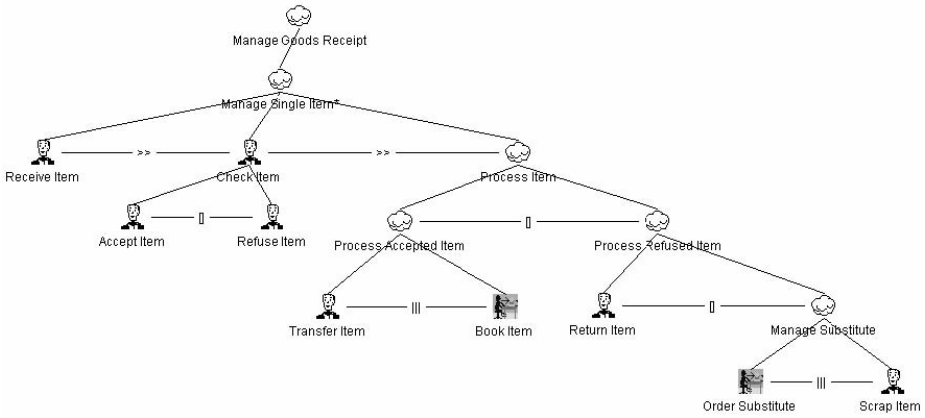


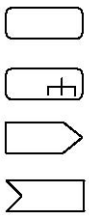
Fig. 1. CTT model of task *Manage Goods Receipt*.

Most task-based approaches do not support a formal description of the task domain, though often aware of the need for it. In Sect. 2.4, we use TaOSpec (e.g. [9]) to enrich our example task model by task-domain objects which help to describe preconditions and effects of sub-tasks in a formal way.

2.2 Activity Diagrams in UML 2.0

An activity diagram is a graph consisting of action, control, and/or object nodes, which are connected by control flows and data flows (object flows). UML 2.0 offers a variety of notations. We mention only some of them, which are used in the context of this paper. For more information we refer to [24].

Actions are predefined in UML and constitute the basic units of activities. Their executions represent some progress in the modelled system. It is distinguished between four types of actions.



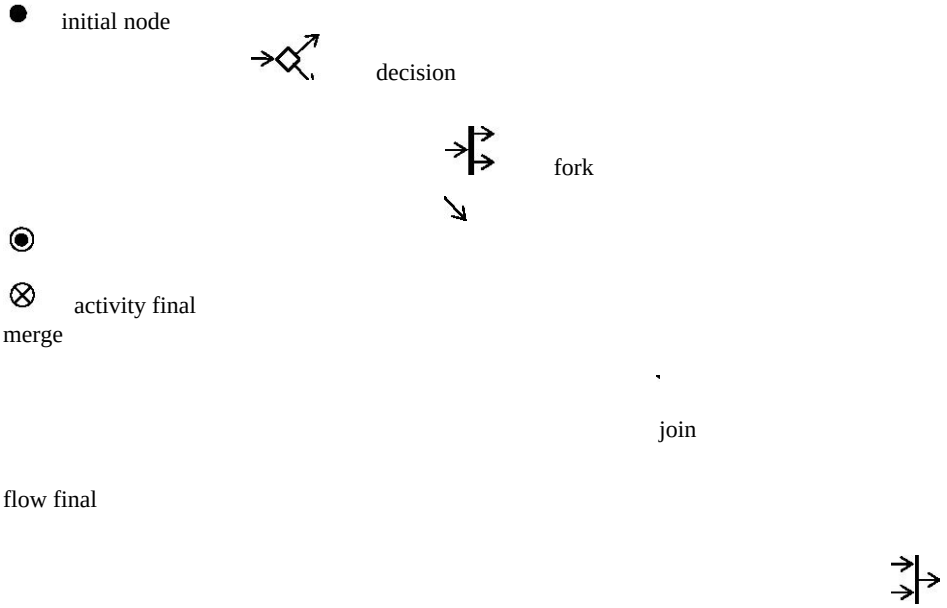
CallOperationAction: invokes user-defined behaviour

CallBehaviorAction: invokes an activity

SendSignalAction: creates an asynchronous signal

AcceptEventAction: accept signal events generated by a SendSignalAction An *activity* describes complex behaviour by combining actions with control and data flows. CallBehaviorActions allow a hierarchical nesting of activities.

Control nodes serve to describe the coordination of actions of an activity. Following node types are supported.



2.3 Related Work

The need for integrating knowledge from both the HCI field and from Software Engineering was seen by others. However, most of the current work in model-based design is concentrated on providing alternatives for task trees in CTT notation in form of structural UML diagrams (e.g. [18], [3], [2]). [15] suggests a mapping from tasks specified in a CTT model to methods in a UML class diagram. We believe that activity diagrams are a more appropriate UML formalism for integrating task-based ideas into the object-oriented methodology. They allow behavioural descriptions at different levels of abstraction. Hence, it is possible to specify task hierarchies and temporal constraints between sub-tasks in a straightforward way. Furthermore, activity diagrams are already used to specify workflows and business processes. Relations between workflows and task modelling are explored e.g. in [23], [6], and [21].

Transformation from Task Models to Activity Diagrams

In this section a transformation from task models into activity diagrams is presented. It preserves the hierarchical structure of models by deriving corresponding nested activities. Each level in a task hierarchy is mapped to one activity (that is called through a

CallBehaviorAction in the level above) as illustrated in an abstract way in Fig. 2.

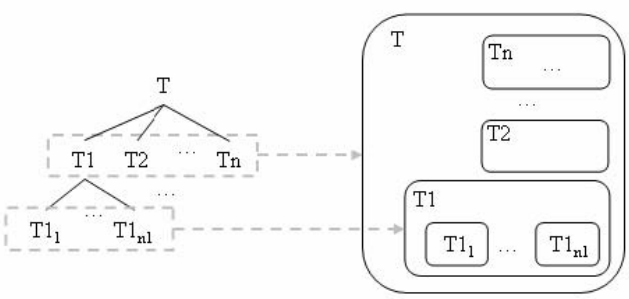


Fig. 2. The transformation preserves the hierarchical structure of the task model.

For each temporal operator mentioned in Sect. 2.1, a transformation rule is defined in the figures below. The left parts of the figures show task structures which are relevant for the transformation rules. On the right side, corresponding activity diagram fragments are given. For reasons of simplicity, the labels for CallBehaviorActions were omitted. The consideration of a node and its direct sub-nodes within a task tree is sufficient for most of the rules. The dashes above and below the nodes indicate the context. So, rules are applied to those parts of an actual task tree, which match the structures given on their left sides.

3.1 Transformation Rules for CTT-like Tasks

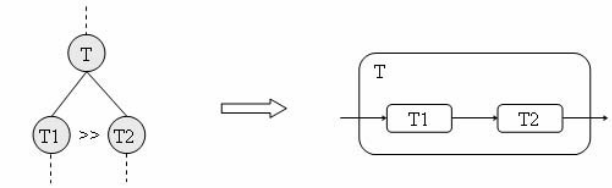


Fig. 3. R1: Enabling relation as activity diagram.

Fig. 3 shows the enabling operator that describes that task T2 is started when task T1 was finished. In the activity diagram, this case is modelled with a simple sequence of activity T1 followed by T2.

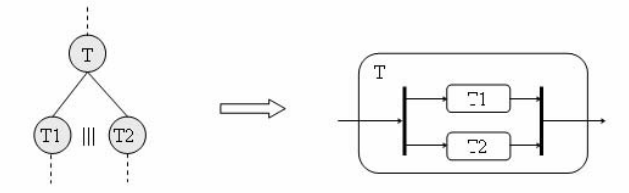


Fig. 4. R2: Concurrency relation as activity diagram.

In Fig. 4, the |||-operator is used to express the concurrent execution of tasks T1 and T2. In a corresponding activity diagram fragment the fork node (first bar) and the join node (second bar) are applied to describe this behaviour.

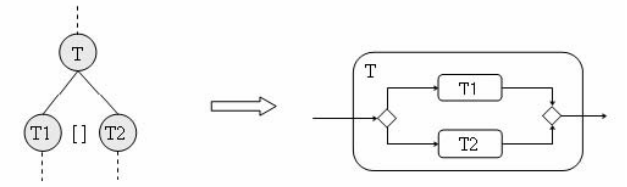


Fig. 5. R3: Choice relation as activity diagram.

Fig. 5 deals with the alternative operator. So, either task T1 is allowed to be executed or task T2. In the corresponding activity diagram part, this operator is realised by using a decision and a merge node. Guards could be attached to the arrows behind the decision node to specify which of the tasks should be performed next.

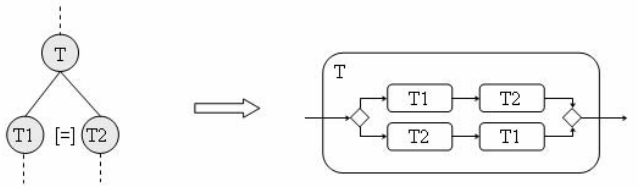


Fig. 6. R4: Order independent relation as activity diagram.

The order independent operator is handled in Fig. 6. There, either task T1 and after that task T2 is executed or first T2 and then T1. In the corresponding activity diagram fragment the situation is modelled with two sequences, a decision and a merge node. It should be mentioned that a problem may arise if more than two activities are executed in an independent order because the number of possible sequences in the activity diagram is growing very fast. In such cases, the readability could be improved by using a stereotype as also proposed below for task deactivation.

Iteration is a special kind of temporal relation. It is not specified between two or more different tasks but can be considered as a feature of a task itself. Fig. 7 shows two ways how activity diagrams can express iteration. We prefer version b) because of better readability (particularly in cultures where people read from left to right).

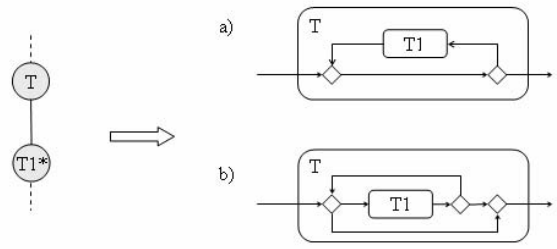


Fig. 7. R5: Iteration in task models and activity diagrams.

In order to find a mapping from task deactivation to a proper activity diagram fragment we have to go somewhat deeper into the semantics of task models. Assume that task T2 specifies a set of sequences of basic sub-tasks (leaves in a task tree) beginning with either $T2i_1$,$T2i_2$, … or $T2i_n$. This set is finite and is called enabled task set – ETS
(e.g. [19]). Thus, ETS(T2)={$T2i_1$,…,$T2i_n$}. T2 deactivates task T1 if one of the basic sub-tasks in ETS(T2) is accomplished. The execution of T1 is stopped and T is continued

by performing the rest of T2.

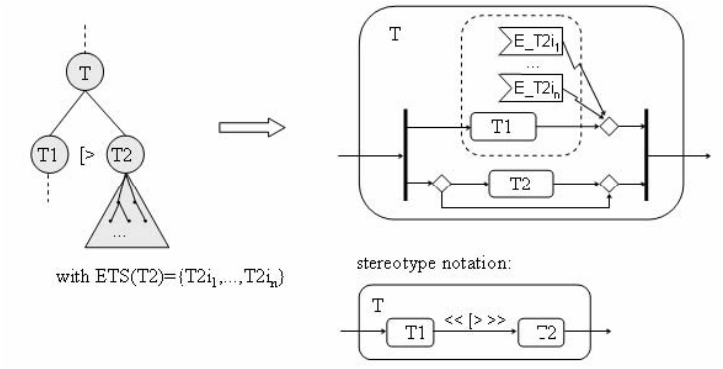


Fig. 8. R6: Deactivation in task models and activity diagrams.

To understand the transformation rule depicted in Fig. 8 we first need to look at Fig. 9a) where an activity diagram for modelling basic sub-tasks is presented. Basic subtask T is mapped to a sequence of a CallOperationAction with the same name and a SendSignalAction E_T which notifies that T was performed. The stereotype <<complex action>> specifies that no other action can be performed between T and E. In the diagram fragment in Fig. 8 AcceptEventActions for accepting signals sent by actions which correspond to the basic tasks in ETS(T2) are used in combination with an interruptable region (denoted as dotted box) to describe a possible deactivation of T1. However, to keep the diagrams readable we suggest to use the notation with stereotype << [> >> as shown down right in Fig. 8.

A transformation for the suspense-resume operator is not proposed. It would require a kind of "history-mode" for activities as known, for example, for states in state charts.

Transformation of sibling sub-tasks

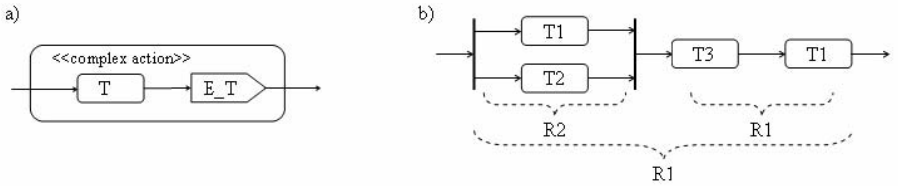
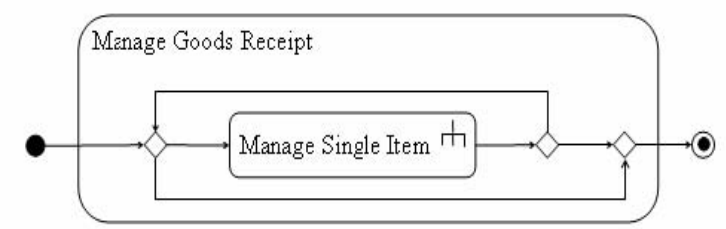


Fig. 9. a) Basic sub-task T as activity diagram, b) Transformation of sibling sub-tasks.

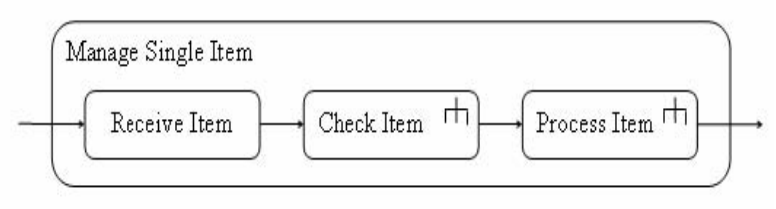
Parent nodes with at most two sons are considered only in the transformation rules. However, all sibling sub-tasks of a task at the hierarchical level n have to be mapped to nodes of a corresponding activity diagram at refinement level n. Hence, a multiple application of rules at the same level of the hierarchy is necessary as indicated in Fig. 9b) for sibling sub-tasks T1, T2, and T3 with temporal constraints (T1 ||| T2) >> (T3 >> T1). Here, rule R2 is applied to (T1 ||| T2), R1 to (T3 >> T1), and then rule R1 again to the intermediate results. Take note that a combined application of rules at the same refinement level is possible because the activity diagram fragments in all rules have exactly one incoming control flow and one outgoing control flow.

3.2 Transformation of an Example Task Model

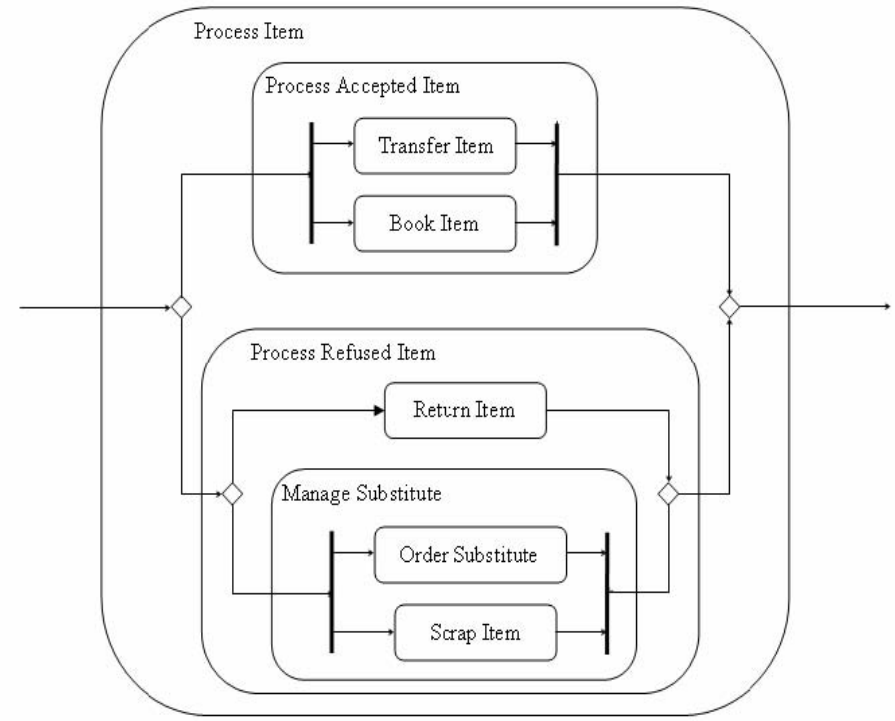
We will now transform the sample task model of Sect.2.1 into an activity diagram to show how transformations work. The task model is about managing incoming items in an abstract company. First, we need to model the begin and end node of the UML activity diagram and then the root task in between these nodes which is shown in Fig.
10. The root task is *Manage Goods Receipt* and is refined by the iterative sub-task *Manage Single Item* (rule R5).



Sub-task *Manage Single Item* is divided into three sub-tasks which are related by the enabling operator in the task model. The double application of the transformation rule R1 in Fig. 3 results in the activity diagram shown in Fig. 11.



The task *Receive Item* is a leaf in the task model of section 2.1. Thus, a diagram as to be seen in Fig. 9a) has to be created. The refinement of activity *Check Item* is shown in Fig. 12. Sub-tasks *Accept Item* and *Refuse Item* are basic ones in the task model and have to be refined similarly *Receive Item.*

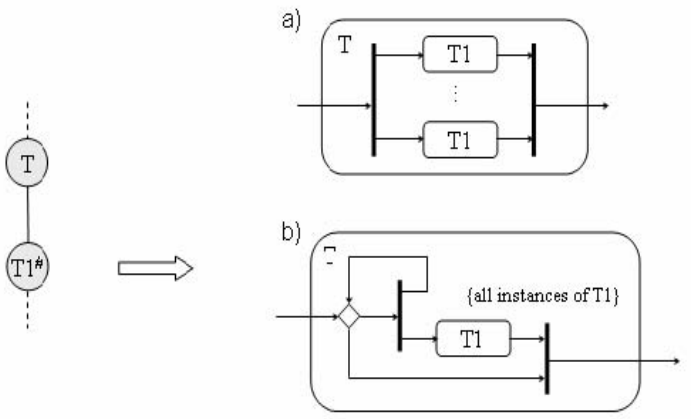


Now, only the task *Process Item* is left for continuing the transformation process and refining the activity diagram. In Fig. 13, the result of this is shown. There, two choice and concurrency operator transformations are used to get this final result for the activity *Process Item*. To get the whole activity diagram as the result of the transformation process the diagram of Fig. 11 should replace the activity of the same name in Fig. 10. The same has to be done with the diagrams of Fig. 12 and Fig. 13 in Fig. 11.

3.3 Handling of Extensions to CTT-like Task Models

3.3.1 Additional Temporal Operators

Our experiences in case studies raised the need for another kind of iteration. In the example, it could be more comfortable for users if they can manage a second, third or more items independent from the first item. Unlike with the normal iteration, in this kind of iteration one can start the next loop before finishing the first one. We call it *in-stance iteration* (denoted by $T^{\#}$). In Fig. 14a), a first idea of a corresponding activity diagram is drawn. There, any number of activity T1 can be started parallel. Unfortunately, the dots between the activities of T1 are not allowed to be used in UML. So we had to search for another solution.



In Fig. 14b), it can be seen how instances of T1 are created. The choice/merge node creates a new instance of T1 if it chooses the arrow in the middle. Then, after the fork node has been passed activity T1 begins and at the same time the token comes back to the choice node. In the same way, any number of new instances of T1 can be created. After sufficient activities of T1 are started the choice node takes the lower arrow. Unfortunately, there is currently no way how the instances of T1 can be caught. In this figure, the functionality of waiting for all the instances finishing is modelled with the discriminator *{all Instances of T1}* associated to the join node.

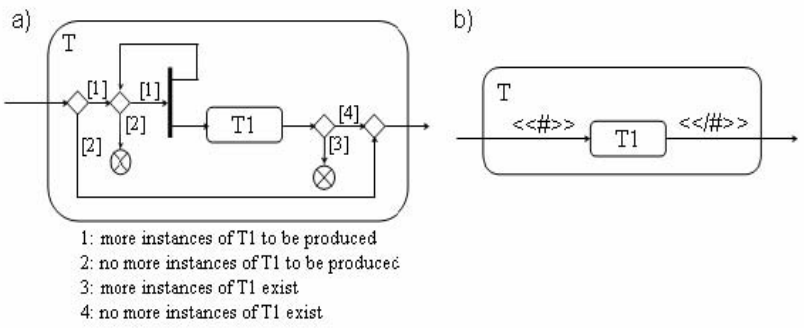


Fig. 15a) depicts an alternative activity diagram for instance iteration. In addition, we suggest stereotypes again to describe instance iteration in a convenient way. This possibility is shown in Fig. 15b). The stereotype **<<#>>**indicates that any numbers of the following task T1 can be created. With the other stereotype **<</#>>** it is said that all of the started tasks T1 must be finished before T is finished.

3.3.2 Task-Domain Objects and Object Flows

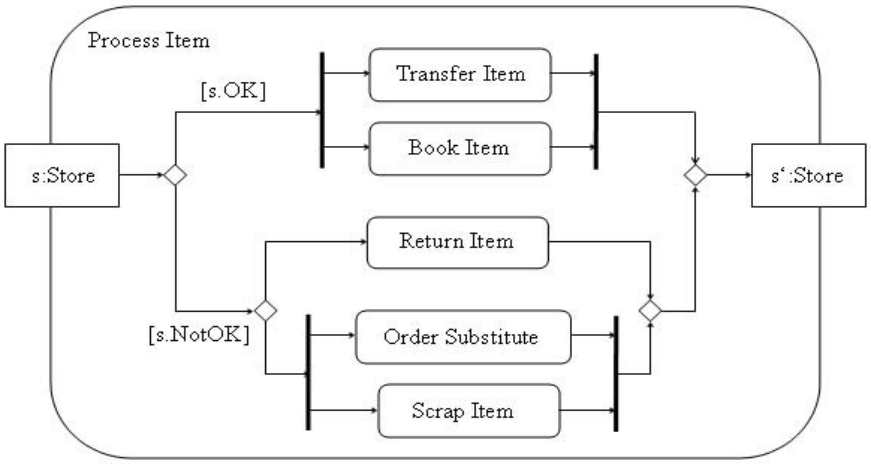
With TaOSpec we developed a specification formalism in our group, which allows to describe tasks as well as task-domain objects in a formal way. In addition, preconditions of sub-tasks and their effects on task-domain objects can be specified. In [8], we have shown that such a hybrid notation often leads to more concise and, possibly, to more natural descriptions than pure temporal notations (like CTT models) or pure state descriptions. In Fig. 16, a more concise TaOSpec fragment, which corresponds to sub-task *Process Item* in Fig. 1 is given (for more details on TaOSpec see e.g. [9]).

```
...
Process_Item($s:Store)[PRE $s.NotOK] =
   ( Order_Substitute() ||| Scrap_Item($s))
   [] Return_Item($s),

Process_Item($s:Store)[PRE $s.OK] =
   Transfer_Item($s) ||| Book_Item($s).

Scrap_Item($s:Store) =
    setAttr($s,"scrapped",$s.$scrapped+1)
    >> getAttr($s,"item",$w)
    >> setAttr($w,"status","scrapped"),
...
```

```
ELEMENT Store
 USES Item
 ATTR
  $transferred: int,
  $scrapped: int,
  $returned: int
 ADDATTR
  $item: Item
 STATES
  OK = $item.OK,
  NotOK = $item.NotOK
```

```
ELEMENT Item
 ATTR
  $okay: bool,
  $status: string
 ADDATTR
  $invoice: string
 STATES
  OK = $okay==true,
  NotOK = $okay==false
```

Activity diagrams not only allow control flows but also object flows. TaOSpec elements can be mapped to objects. In addition, implicit objects flows in TaOSpec (via parameters in a task model) become explicit object flows in activity diagrams. In Fig. 17, an activity parameter is used to describe a *Store*-object. Guards reflect the preconditions specified in Fig. 16. In comparison to Fig. 13 this specification is clearer.



4 Model-Based Development of Tool Support

4.1 General Development Approach

After several years of individual software development we recently used the MDA approach [17]. Using the technology offered by Eclipse [11] and several related frameworks we specify our metamodels and generate main parts of our tools. In other words: we apply model-based techniques to create model-based development tools.

Meta Model

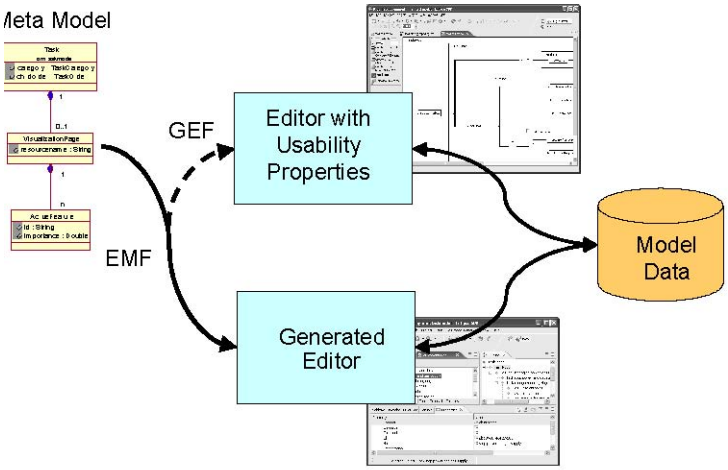


Fig. 18: General approach for model-based development

Based on a meta model and the eclipse modeling framework [12] an editor can be generated that allows the manipulation of corresponding models. In general, such generated EMF-based editors are not very user friendly. For hierarchical models it is acceptable because the model is represented in an appropriate way.

Alternatively, the graphical editing framework [13] offers a technology to develop editors that fulfil the usability requirements better. These editors can work on the same data as the generated editor. In this way, test data can be edited with the generated editor and visualised with the developed one until the full functionality of the user friendly editor is available.

Fig. 18 gives an overview of the general development process. According to the left hand side it is necessary to model the domain of the models. In our case this is a specification for task models.

It is our idea that each task has to be performed by a person or system in a specific role. A task changes the state of certain artifacts. In most cases this will be one object only. Other objects support this process. They are called tools. Both, artifacts and tools are specified in a domain model.

The general idea of hierarchical tasks and temporal relations between tasks are of course true for our models as well.

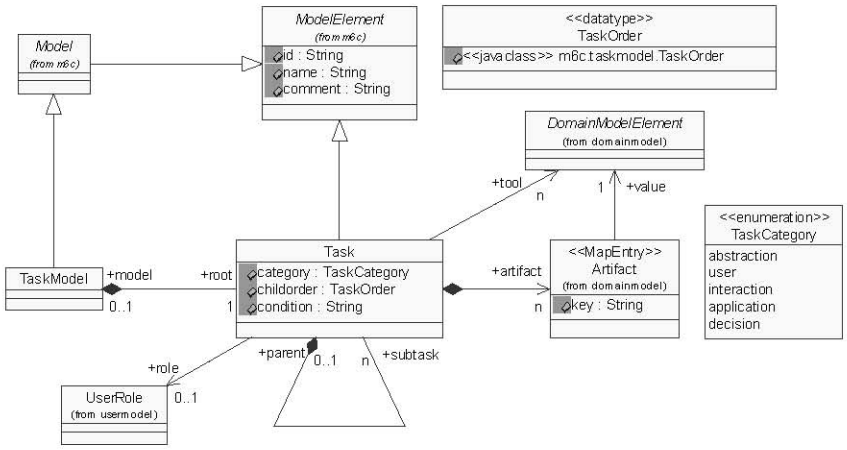


Fig. 19: Meta model for task models

Fig. 19 shows a part of the meta model for hierarchical task models. The parent subtask-relation at the bottom of the class diagram allows the specification of hierarchical task structures. A task model consists of exactly one root task and each task has either a relation to a parent task or is the root task of the model. A task has relations to a user role from a user model and artifacts and tools from a domain model.

A special class *TaskCategory* specifies as enumeration the different types of tasks as they are already known from concurrent task trees [7].

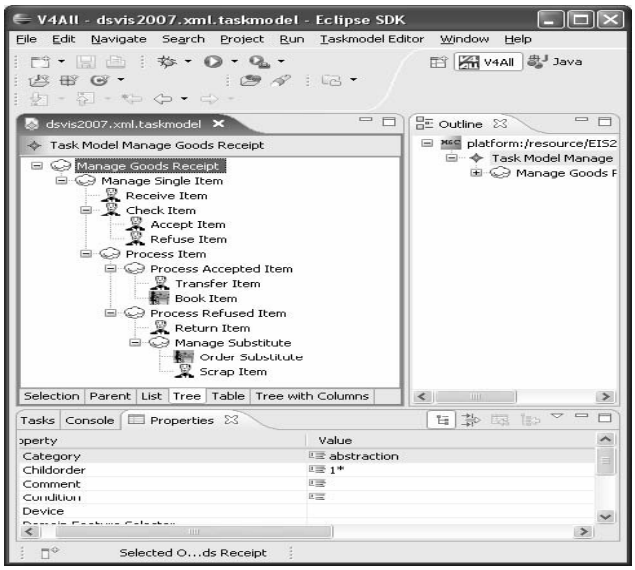
The temporal relations like enabling, order independence, concurrent, choice, suspend-resume and deactivation are represented by a class *TaskOrder*. This class allows the specification of temporal relation of sub-tasks by regular expressions. This gives a little bit more freedom in describing the dynamic behaviour of task models. It allows specifications that are difficult to visualize like ((a >> b >> c) [ ] (b >> c >> d)).

Using EMF [12] the meta model in Fig. 19 is sufficient to generate automatically software for an editor of the corresponding models. This editor of course does not fulfil a lot of usability requirements but it allows specifying models in detail. Fig. 20 gives an impression how the user interface of such a generated editor looks like.

In the central view one can see the tree of model elements, in this case the task hierarchy. New model elements can be inserted via context menu and copied or moved by drag&drop. In the bottom view every attribute of the currently selected model element is shown and can be manipulated via text or combo boxes depending on the corresponding data type in the meta model.

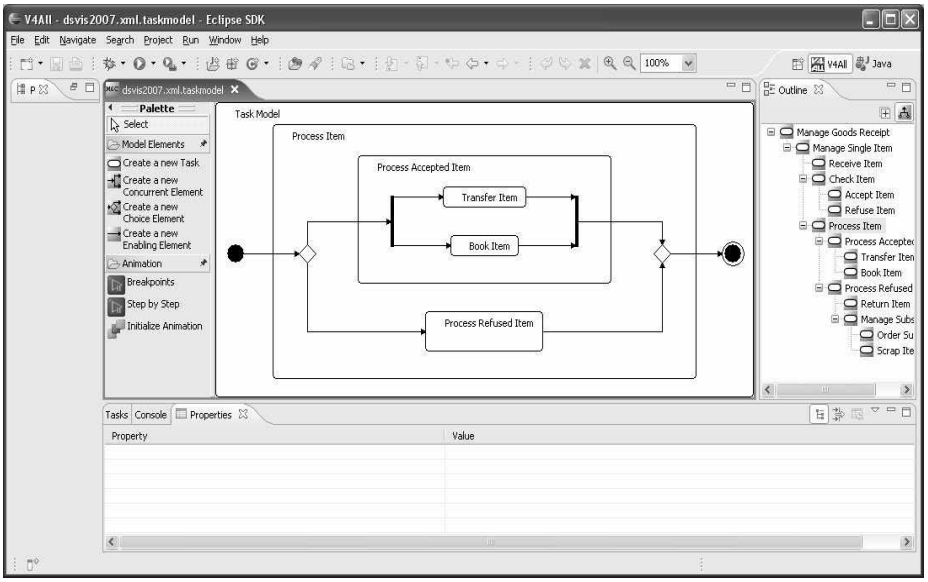
4.2 Tool Support for the Transformation from Task Models into Activity Diagrams

Based on the meta-task model and the generated software an own structured editor for activity diagrams was developed using GEF [13]. It is called structured, because it does not allow the drawing of individual nodes and connections like most editors, but allows only the insertion of complex structured model parts like sequences of “enabling tasks”, “choice elements” and “concurrent elements”. More or less only “subtrees” are allowed to be inserted. The underlying model is the same as for the generated task-model editor. The visual representation is generated automatically from this model and an explicit model to model transformation is not necessary.



In this way a lot of mistakes regarding the creation of activity diagrams can be omitted. This consequence is already known from structured programming that is an improvement over programming with “go to”. Drawing lines can be considered as programming “go to”.

Fig. 21 gives an impression how the user interface of the structured activity diagram editor looks like. On the left hand side one can see the palette of possible operations allowed for a diagram. After selecting one operation and moving the mouse over the diagram certain interaction points will be visible signalling places, where the operation could be performed. After selecting one of these interaction points the editor asks for an integer value, which represents the number of task in a sequence, the number of choices or the number of order independent elements.



According to the answer of the user the corresponding elements are immediately inserted and the diagram is drawn again.

Interactively one can decide how many levels of detail one would like to see. It is also possible to have a look at a special lower level only. This is possible because the task hierarchy is presented as task tree as well. This is cointained in the outline view on the right hand side of the user interface. By selecting a tree node the corresponding sub-diagram is shown in the main view.

GEF uses the model-view-controller pattern to ensure consistency between model and its appropriate visualisation. Here, the model is a task model matching the meta model, the view consists of geometrical figures and labels and the controller defines, which interactions are possible and how model changes are reflected in the corresponding view.

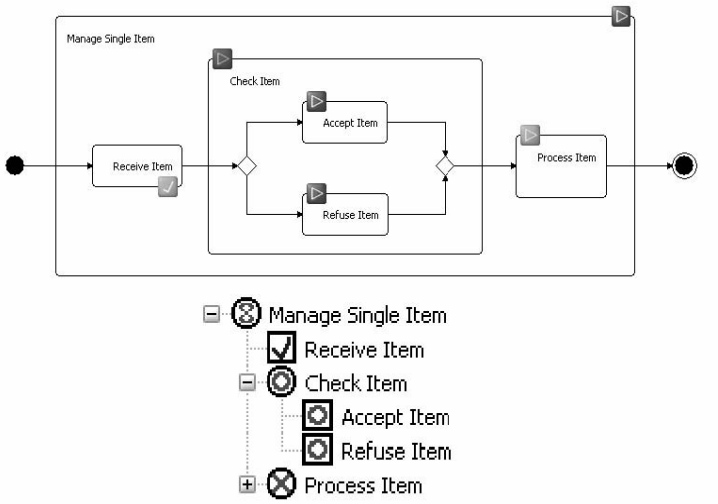
4.3 Animation of task models and corresponding activity diagrams

To validate the behaviour of activity diagrams (or rather their underlying task models) an animation tool has been developed.

Fig. 22 contains an example of an animated activity diagram and the corresponding visualization of the tasks model.

At the current point of execution task *Receive Item* is already performed. It is now the decision to activate *Accept Item* or *Refuse Item*. Both are enabled. *Process Item* is disabled because *Accept Item* or *Refuse Item* has to be performed first.

Animation can be performed with activity diagrams on different levels of detail. Each task containing several subtasks can be collapsed and the user can animate just a part of a complex model. It is possible to automatically proceed the animation until a decision has to be made or to run the animation step by step. In the first mode one can additionally set breakpoints to stop the process at a specific point.



Summary and Future Work

The paper discussed how task modelling concepts can be made more attractive for the software engineering community. It was proposed to present task models as activity diagrams -a notation most developers are familiar with. We suggested a "taskoriented" development of activity diagrams and we defined corresponding mapping rules. Although this restricts the variety of possible activity diagrams we believe that a more systematic methodology helps to come to more reasonable models. Temporal relations available in task models but missing in UML were also represented by stereotypes.

Tool support was suggested that allows to derive and to edit activity diagrams in this structured way. An animation of models helps to evaluate the requirements specifications and to get early feedback from users.

From our point of view structured activity diagrams could play a similar role to activity diagrams as structured programs to programs.

In the future an adequate modelling method has to be elaborated, which allows to develop workflows and task models of current and envisioned working situations as well as system models in an intertwined way. Currently, there exists no satisfying approach in both communities. Activity diagrams in UML 2.0 may be useful to integrate the object-oriented design of interactive systems and the task-based design approach.

References

1 Annett, J., Duncan, K.D.: Task analysis and training design. In Occupational Psychology, Vol.41. (1967)

2 Bastide, R., Basnyat, S.: Error Patterns: Systematic Investigation of Deviations in Task Models. In: Coninx, K., Luyten, K., Schneider, K. (eds.), Proc. of TAMODIA'2006. (2006)

3 Van den Bergh, J.: High-Level User Interface Models for Model-Driven Design of Context-Sensitive User Interfaces. PhD thesis, Universiteit Hasselt. (2006)

4 Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Vanderdonckt, J.: Computer-Aided Window Identification in TRIDENT. INTERACT'95, Chapman-Hall. (1995)

5 Bomsdorf, B.: Ein kohärenter, integrativer Modellrahmen zur aufgabenbasierten Entwicklung interaktiver Systeme. PhD thesis, Universität Paderborn. (1999)

6 Bruno, A., Paternò, F., Santoro C.: Supporting interactive workflow systems through graphical web interfaces and interactive simulators, In: Dix, A., Dittmar, A. (eds.), Proc. of TAMODIA'05. (2005)

7 CTTE: http://giove.cnuce.cnr.it/ctte.html (read: 20.09.06)
8 Dittmar, A., Forbrig, P.: The Influence of Improved Task Models on Dialogs. In Proc. of CADUI'04. (2004)
9 Dittmar, A., Forbrig, P., Heftberger, S., Stary, C.: Support for Task Modeling -A "Constructive" Exploration. In: Proc. EHCI/DSV-IS'2004. (2004)
10 Dittmar, A., Forbrig, P., Reichart, D., Wolff, A. (2005). Linking GUI Elements to Tasks – Supporting an Evolutionary Design Process. In: Dix, A., Dittmar, A. (eds.), Proc. of TAMODIA'05. (2005)
11 Eclipse Homepage. http://www.eclipse.org/ (read: 20.09.06)
12 Eclipse Modeling Framework Homepage. http://www.eclipse.org/emf/ (read: 20.09.06)
13 Graphical Editing Framework Homepage. http://www.eclipse.org/gef/ (read: 21.09.06)
14 Johnson, P.: Human computer interaction: psychology, task analysis, and software engineering. McGraw-Hill Book Company. (1992)
15 Limbourg, Q.: Multi-Path Development of User Interfaces. PhD thesis, Université catholique de Louvain. (2004)
16 Limbourg, Q., Pribeanu, C., Vanderdonckt, J.: Towards Uniformed Task Models in a Model-Based Approach. In: Proc. of DSV-IS2001, LNCS 2220, (2001)
17 http://www.omg.org/mda/ (read: 20.09.06)
18 Nunes, N.J., e Cunha, J.F.: Towards a UML profile for interaction design: the Wisdom approach. In Evans, A., Kent, S., Selic, B. (eds.), UML, LNCS 1939. (2000)
19 Paternò, F.: Model-Based Design and Evaluation of Interactive Applcations. Springer-Verlag. (2000)
20 Puerta, A.: The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. In Vanderdonckt, J. (ed.), Proc. of CADUI'96. (1996)
21 Stavness, N., Schneider, K. A.: Supporting Workflow in User Interface Description Languages. Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, AVI 2004. (2004)
22 Szekely, P., Luo, P., Neches, R.: Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. CHI'92. (1992)
23 Trætteberg, H.: Modelling Work: Workflow and Task Modelling. In: Proc. of CADUI'99. (1999)
24 http://www.uml.org (read: 20.10.06)
25 Vanderdonckt, J., Puerta, A.: Preface -Introduction to Computer-Aided Design of User Interfaces. In: Proc. of CADUI'99. (1999)
26 Wilson, S. Johnson, P. Markoplous, P.: Beyond hacking: A model based design approach to user interface design. In: Proc. of BCS HCI93. Cambridge University Press. (1993)

Questions

Morten Borup Harning:
Question: How can you, by simply showing how to represent task models using activity diagrams, convince developers that focusing on task modeling is important?
Answer: Both the tool support and the extended notation makes it easier to use activity diagrams to represent task models. This is achieved using tool animation; making it easier to specify temporal issues, iteration and so on in a way that is not currently possible. Of course, the tool can be used to specify system oriented designs, but so can CTT.

Daniel Sinnig:
Question: How does the tool enforce the building of activity diagrams that represent task models? Designers don't usually build task diagrams.
Answer: The tool does not force them but it helps them to understand what is represented in the task model.

Philippe Palanque:
Question: As task models are supposed to be built by Ergonomists or human factors people, why do you think software engineers would use the tool?
Answer: There are 2 main levels: the analysis level and the design level. Our work is supposed to support the design level. The initial task model coming from analysis of work has to be done first. The idea is to support the Software engineers so that they understand the analysis of work and to exploit it during design.

Prasun Dewan:
Comment: Your activity diagrams describing constraints on concurrent user actions remind me of an idea in concurrent programming called path expressions which describe constraints on procedures that can be executed by concurrent processes. To address the previous question about the usefulness of these diagrams, path expressions provide a more declarative explanation of the constraints than something low level like semaphores. It seems activity diagrams have a similar advantage.

Question: You might want to look at path expressions to see if you can enrich activity diagrams. Path expressions can be used to automatically enforce the constraints they describe during application execution. Can activity diagrams do the same?
Answer: They can be used to automate animations of the constraints.

Jan Gulliksen:
Question: Would you use these diagrams with end users and do you have experience with this?
Answer: Yes, the simulation tool is aimed at that. And it is very easy for them to understand the models thanks to execution of the models.