From Task to Dialog model in the UML

Jan Van den Bergh and Karin Coninx

Hasselt University, transnationale Universiteit Limburg,
Expertise Centre for Digital Media
Wetenschapspark 2
3590 Diepenbeek
Belgium
{jan.vandenbergh,karin.coninx}@uhasselt.be

Abstract. Many model-based approaches for user interface design start from a task model, for which the ConcurTaskTrees notation is frequently used. Despite this popularity and the importance that has been given to a close relation with UML, no relation has been established with UML state machines, which have been shown to be useful for the description of the behavior of user interfaces. This paper proposes a semantic mapping of tasks and all temporal relations of the ConcurTaskTrees to UML state machines which forms the basis for a compact dialog modeling notation using UML state machines. The proposed approach uses a UML profile to reduce the visual complexity of the state machine.

1 Introduction

The ConcurTaskTrees notation (CTT) [12] is one of the most popular notations for hierarchical task modeling used in academia for model-based design of user interfaces. Since the Unified Modeling language(UML) [11] is one of the most established modeling notations for software models, several approaches have been presented to integrate the ConcurTaskTrees notation into UML.

Nunes and e Cunha[10] made a mapping to UML class diagrams as part of the Wisdom notation. They mapped each task in the task model to a UML class. The relations between parent and child tasks are represented using aggregation relationships while the relations between siblings are represented using constraints. All task categories are represented using the same task symbol.

Nobrega et al. [9] present a different approach which emphasizes the fact that tasks in the ConcurTaskTrees notation represent activities. Therefore, they show tasks using the UML notation for actions. They also propose new symbols for the temporal operators of the ConcurTaskTrees. All changes they proposed were made to integrate the ConcurTaskTrees notation both visually and semantically into the UML.

In earlier work [16] we opted to extend the class diagram to represent the CTT but to keep the appearance closer to the original. This resulted in some notable differences with the approach presented in [10]: The relations between tasks are represented by stereotyped associations and each task category keeps its original symbol (and properties), thus keeping the look of the model closer to the original CTT specification.

A close relationship of the CTT with UML state machines has not been established. It has however been shown that (this type of) hierarchical statecharts can effectively be used to describe [3] and generate user interfaces [1, 14]. In this work, we extend the state-of-the-art by presenting a semantic mapping of the dynamic aspects of the task model and exploiting this mapping for a compact, powerful dialog modeling notation using UML state machines.

The mapping between CTT and UML state machines is established by giving a behavioral specification for a task using UML state machines and is discussed in section 4 after a short introduction of both notations. This specification is then used in section 5 to express the behavioral semantics of *all* temporal operators. These specifications are used to derive a dialog model from a CTT model. A UML stereotype is used to reduce the visual complexity of the model. The paper is concluded by a discussion of related work and conclusions.

2 ConcurTaskTrees

The ConcurTaskTrees notation [12] is a hierarchical task modeling notation that has a tree-based structure. All nodes in the tree are tasks. There are four task categories, each having its own symbol: abstraction (, an abstract task has subtasks of at least two different task categories), interaction (, a task performed through interaction of a user with an application), user(, a task performed by the user without interaction with the application) and application (, a task performed by the application). Siblings in the tree are connected using temporal operators derived from LOTOS [5]. Section 5 discusses these operators into more detail.

Fig. 1 shows a CTT example. It specifies how a user can check the availability of a hotel room. First the user specifies the start and end date of the stay or the start and duration. Next, the user specifies the room type. The application then checks the available hotel rooms (during that period, `Perform Query`) and shows the available rooms to the user. The user can then refine the selection of available rooms by adapting the period and the room type until the refinements are submitted. The check for availability can be cancelled at any time (`Close Availability`).

3 UML State Machines

UML state machines are an object-based variant of Harel statecharts [2]. Both have the advantage over other forms of statecharts and state transition networks that they support concurrent states. This means that when a UML state machine is executed, it can be in multiple states at a given moment in time. Furthermore Harel statecharts as well as UML state machines allow hierarchical composition of states.

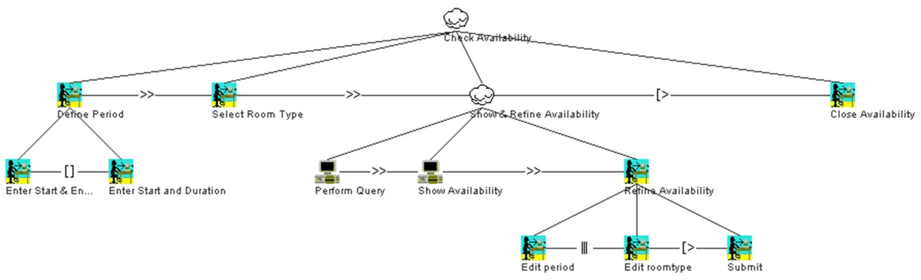


Fig. 1. Example of a CTT specification

Fig. 2 gives an overview of the relevant symbols in this notation. The initial pseudo state and the final state respectively mark the start and the end of a composite state or state machine. An exit point can be used to mark an alternative end point (e.g. due to abnormal behavior). A fork symbol can be used to specify that a single state is followed by two or more concurrent states. A join allows to do the opposite. A choice pseudo state can be used to specify multiple alternative next states. Finally, a small black dot (not shown in Fig. 2 is the symbol for a junction which allows to merge or split transitions (displayed as arrows). For a detailed discussion of UML state machines we refer to the UML Superstructure specification [11].

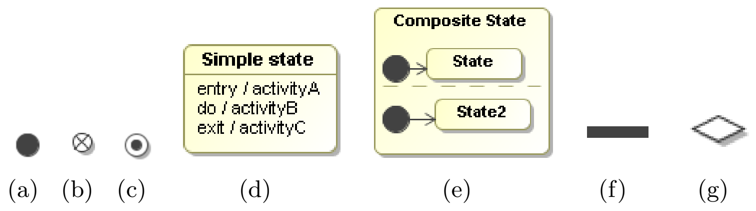


Fig. 2. UML state machine symbols: (a) Initial pseudostate, (b) Exit point, (c) Final state, (d) State with specification of behavior on entry, during and on exit of state, (e) Composite State with two regions specifying concurrent behavior, (f) Fork/join (g) Choice pseudostate

UML state machines have no direct formal mapping, although partial mappings are already specified to stochastic petrinets for UML state machines with the UML realtime stereotype extensions applied [15].

4 Tasks in UML

As mentioned in the introduction, different representations of the CTT in UML have been proposed. Fig. 3 shows the CTT task model of Fig. 1 using the Wisdom notation, proposed in [10]. It clearly shows that each task is represented by the same icon. This icon is related to the stereotype[1]. The task categories are represented by tagged values (shown between brackets such as *application* in Fig. 3) unless they are interactive tasks. Constraints are used to denote the temporal relations. When no constraints are applied between the parent-child relationships of two sibling tasks, these tasks are executed in concurrency.

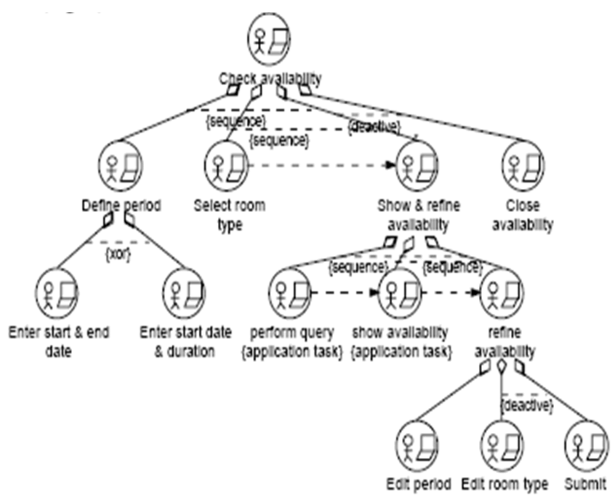


Fig. 3. The example of Fig. 1 using Wisdom notation [10]

In earlier work [16] we presented an alternative representation in UML with a notation that is closer to the original. In fact, the CTT model shown in figure 1 can be modeled using the CTT UML profile[2]. Some differences arise when a task is optional or iterative. These properties are modeled as tagged values, which can be shown when desired.

As both of these approaches use classes to represent tasks and classes can own state machines, a state machine is a natural choice for specifying the behavior of a task. The state machine specification thus complements the specification using classes and gives more details about the behavioral properties of the tasks, while class diagrams can be used to specify the structural properties.

To describe the behavior associated with a task, we model the different states of execution. At the highest level we discern two stages: *active* and *inactive*. These stages can on their turn be subdivided in different states. A task can be considered to be inactive, when it is not yet activated, when it has been

[1] Stereotypes are a light-weight method to extend UML metaclasses.

[2] The profile is made in MagicDraw and available at `http://research.edm.uhasselt.be/~jvandenbergh/cup/ContextualConcurTaskTrees.mdzip`.

successfully *completed* or when it has been *aborted*. The *active* state can be subdivided into two states: *executing* and *sleeping*.

Fig. 4 shows the state machine that can be associated with a task `T1`. Whenever the state `T1` is activated (T1 being the name of the task), the task is considered *active*. The inactive states are depicted in Fig. 4 for completeness. They will not be depicted in further diagrams. On entry and exit of the states `T1` and `Executing`, an activity is specified. These activities broadcast an event, which can trigger state changes for other tasks. All these events have an attribute that specifies the source of the event. On entry of the state `T1` an event, *activated*, is broadcasted indicating that `T1` is *active*. When the actual execution of a task starts an event, *started*, is broadcasted. When an event *stopped* is sent, the task is no longer executing, but the execution might be resumed. The event *ended* indicates that the task has become inactive.

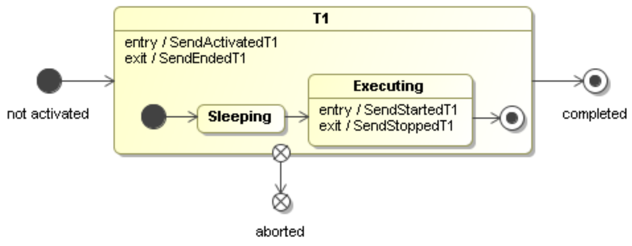


Fig. 4. Task states

The exact meaning of these states depends on the task category and on the platform and context in which the interaction is taking place. Table 1 shows a possible mapping for the states of an interaction task to a concrete context: desktop interaction using a multi-window desktop such as MS Windows or MacOS. The states of an application task presenting data to the user can be described in a similar manner. When the application task is not directly represented on the screen the states might be mapped to the states of the thread or process that executes the task. Giving a concrete description of the states for a user task in general is not as straightforward, although it should be easy to do for user tasks that involve physical activity on a case by case basis.

5 CTT Task Relations and UML State Machines

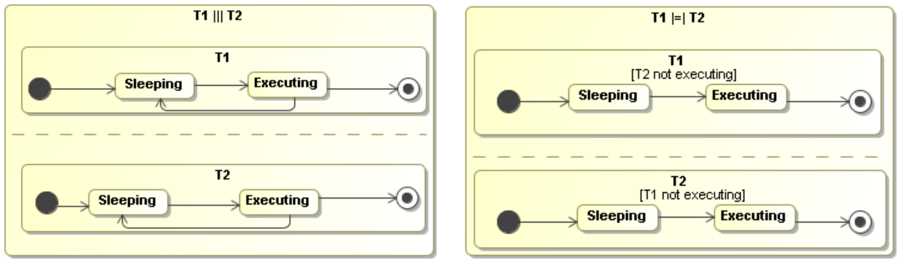
In this section we discuss how the task representation introduced in section 4 can express the temporal relations of the CTT. For each temporal operator a diagram is discussed that shows the application of the operator to two tasks.

concurrency The concurrency operator (`|||`) expresses that two task can be executed in any order and can interrupt each other. It has a straightforward

state	description
active	the window containing the user interface controls associated to the task is shown on the screen
sleeping	the user interface controls associated to the task are disabled or not visible
executing	the user interface controls associated to the task are enabled and visible

Table 1. States of task execution for an interaction task on a pc using a graphical user interface

mapping to UML state machines when the state machine definition in Fig. 4 is used. Fig. 5(a) shows that two parallel tasks can be represented by embedding each task representation in a separate region of a complex state.



(a) Concurrency (b) Order Independence

Fig. 5. Concurrency and order independence

order independence When the order independence operator (`|=|`) is used between two sibling tasks `T1` and `T2`, these tasks can be executed in any order but not concurrently. This means that when `T1` is executing, `T2` cannot start execution and vice versa. When one of the tasks is completed, the other can start executing. Fig. 5(b) also clearly shows that the two tasks cannot interrupt each other.

suspend/resume The suspend/resume operator (`|>`) suspends one task while another one is executing. Using the terminology introduced in section 4 this means that for the expression `T1 |> T2`, T1 cannot be in the state `executing` when T2 is in that same state. This is reflected in the diagram in figure 6. The transition from *sleeping* to *executing* is only possible for task T1, when task T2 is not in the executing state. A transition from *sleeping* to *executing* of the task T2 triggers the inverse transition of task T1.

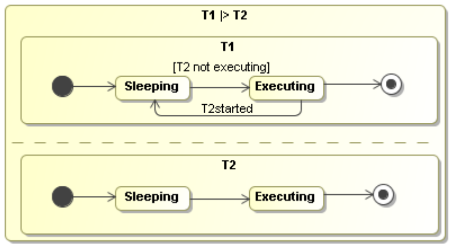
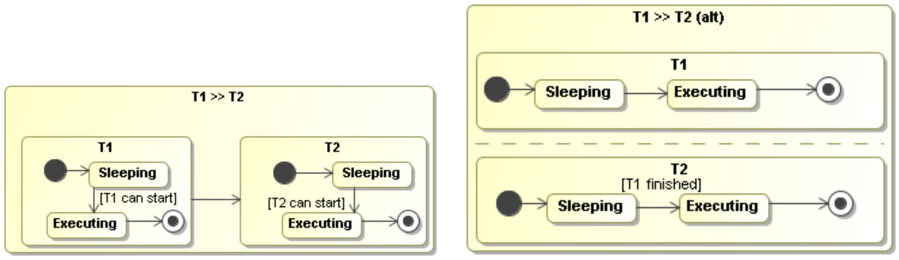


Fig. 6. Suspend/Resume

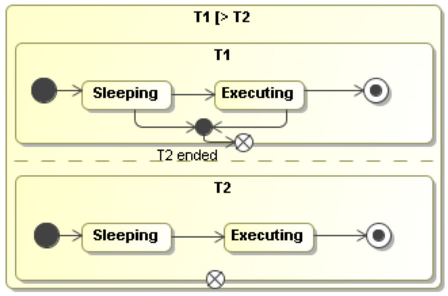


(a) (b)

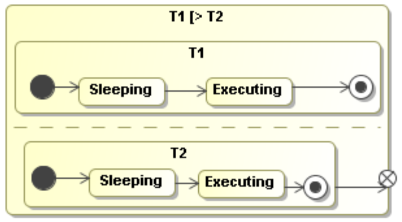
Fig. 7. Enabling

enabling The main property of the enabling operator is that the tasks that are its operands are executed one after the other. In terms of the proposed state machine for tasks, this means that there is only a constraint on the order of the `executing` state. Fig. 7 thus shows two different state machines that satisfy that constraint. Fig. 7(a) corresponds to the situation where only the tasks that belong to the same enabled task set (ETS) are presented in the user interface. An ETS is a set of tasks that are logically enabled to start their performance during the same period of time [12]. Fig. 7(b) corresponds to a situation where two ETSs are merged into a single task set. [13] proposes some heuristics for when such a merge can be desired. Note that unlike many dialog models, Fig. 7(b) still shows that T2 cannot be executed until T1 is finished. This ensures that the dialog model is consistent with the task model, even when ETSs are merged.

deactivation The deactivation operator (`[>`) can be used to let one task interrupt the execution of another task and prevent further execution of that task. Fig. 8 shows what this means in terms of our UML state machine representation. `T1 [> T2` means that when T2 ends execution, T1 immediately becomes inactive. Note that both diagrams in Fig. 8 result in the described effect. The approach in Fig. 8(a) can be extended to work when `T2` has subtasks (although the first subtask of `T2` should be used in this case to be compliant with the CTT specification, while the approach in Fig. 8 cannot but offers a simpler syntax instead.



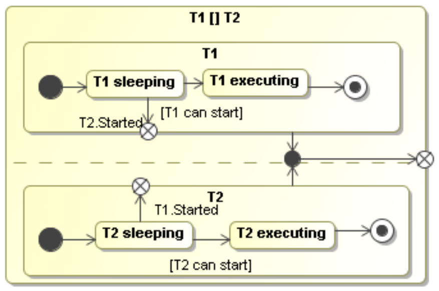
(a)



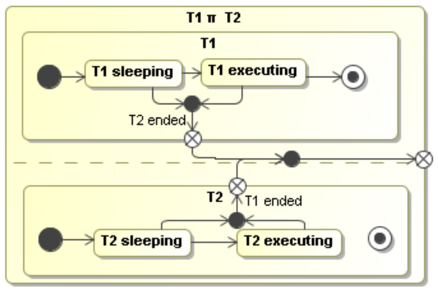
(b)

Fig. 8. deactivation

choice The choice operator (`[]`) offers the option to choose between two tasks of which only one may be completed. As soon as one of the tasks starts execution, the other tasks become inactive. This prevents that more than one task is executing at the same time. This type of choice is called a "deterministic choice" in [4]. The same article also describes a non-deterministic choice. This latter type of choice only allows one task to complete; the other options will not become inactive until one of the tasks has been completed (see Fig. 9(b)).



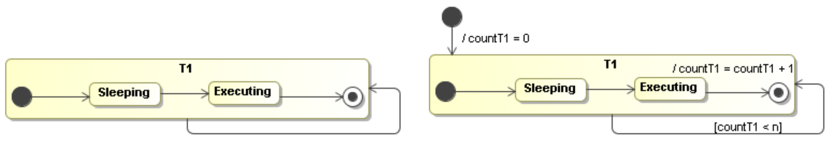
(a) deterministic



(b) non-deterministic

Fig. 9. Choice

task iteration The two possibilities that are offered by the CTT notation for the expression of iterating task can be expressed as is shown in Fig. 10. Both diagrams using UML state machines clearly show the semantics of the iteration operators in the CTT; the repeatable task has to be completed before another iteration of the task can be repeated.



(a) infinite iteration (b) finite iteration

Fig. 10. Task iteration

optional The state machine representation of an optional task contains an additional transition from the sleeping state to the final pseudo state. The transition is triggered by the completion of a task that triggers a transition to another enabled task set.

6 Towards a dialog model

The previous section showed that it is possible to combine the UML state machine description of two (or more) sibling tasks. To create an effective dialog model, however, a complete task model has to be converted into a UML state machine. In this section, we demonstrate by example that it is possible to do so for the task model in Fig. 1.

6.1 The CTT Example

Fig. 11 shows the UML state machine corresponding to the CTT task model in Fig. 1. The hierarchy from the CTT model is preserved in this example. This is however a choice made, not an obligation. Furthermore, one-to-one mapping might not always be possible in the general case but no definitive claims can be made about this without further investigation and in some cases it may be desirable not to copy all abstraction levels from the CTT to the state machine.

We can see that the top-level state is split into four concurrent regions, each corresponding to a direct child of the top-level task in the CTT. The fact that their are four concurrent regions is caused by the choice to use the mapping in Fig. 7(b) for the enabling operator between the tasks `Define period` and `Select room type`, and the tasks `Select Room type` and `Show and refine availability`. For the enabling operator between the tasks `Perform query` and `Show Availability`, the other option was chosen, resulting in a sequence of states within the first region of the task `Show and refine availability`.

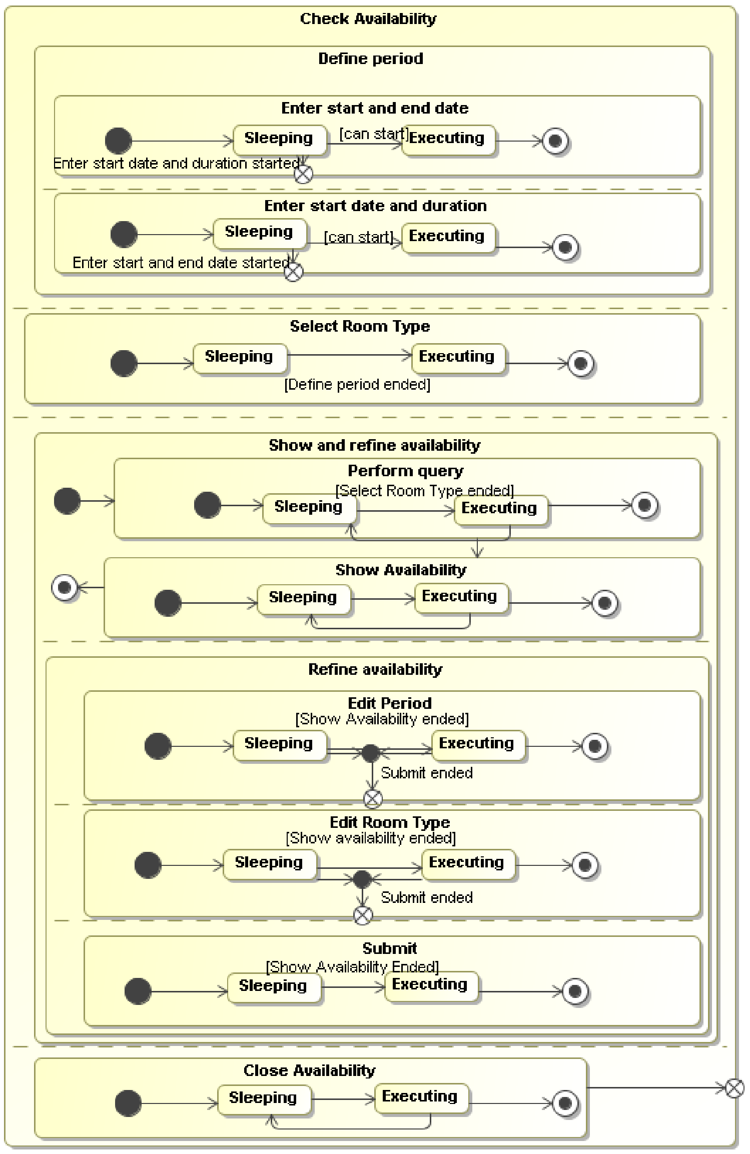


Fig. 11. UML state machine corresponding to Fig. 1 and Fig. 3

When observing the diagram, we can also see the two options to indicate the deactivation operator. The option in Fig. 8(a) is applied for the task `Submit`, while the other option is applied for the task `Close Availability`. It is clear that the choice for the second option reduces the complexity of the overall diagram.

The example in Fig. 11 shows that it is possible to use the proposed notation to map the behavior of the CTT to the UML state machines. The resulting notation is however too complex to quickly understand the meaning of the diagram. To resolve this issue we opted to define a UML profile, which should reduce the complexity of the notation. The profile is discussed into more detail in the following section.

6.2 Simplified notation for a dialog model

Fig. 12 gives an overview of the UML profile we defined to simplify the notation for interactive use of the notation. The profile consists of a single stereotype, << `task`>> for the `State` metaclass. When the UML-profile is applied to a UML state machine, the stereotype should be applied to all instances of the metaclass `State`, i.e. all states in the diagram.

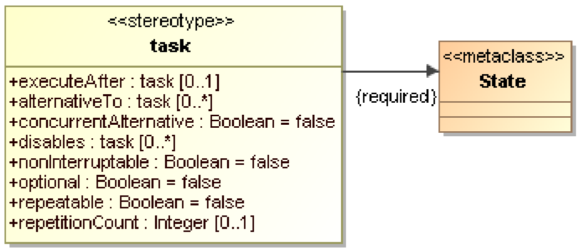

<<stereotype>>
task
+executeAfter : task [0..1]
+alternativeTo : task [0..*]
+concurrentAlternative : Boolean = false
+disables : task [0..*]
+nonInterruptable : Boolean = false
+optional : Boolean = false
+repeatable : Boolean = false
+repetitionCount : Integer [0..1]
{required}
<<metaclass>>
State

Fig. 12. Proposed stereotype for the *State* metaclass

Seven tagged values are defined within the stereotype, which relate to the different temporal operators: `executeAfter` specifies the task after the completion of which the current task can start executing. It is used in case an overlap in the *active* state of the two tasks is desired as is specified in Fig. 7(b). `alternativeTo` allows to specify an alternative task. The collection of tasks specified by this tagged value contains one or more tasks when the corresponding task in the CTT is an operand of the choice operator. `concurrentAlternative` is set to `true` in case of non-deterministic choice. `disables` is a non-empty collection of tasks when the corresponding task in the CTT is the right-operand of a deactivation operator and the mapping to Fig. 8(a) is chosen. `nonInterruptable` is `true` when the corresponding task in the CTT is an operand of the operator order independent. `optional` is `true` when the operator optional is applied to the corresponding task in CTT. `repeatable` is set to true when the corresponding task is repeatable. `repetitionCount` can be used to set the number of repetitions.

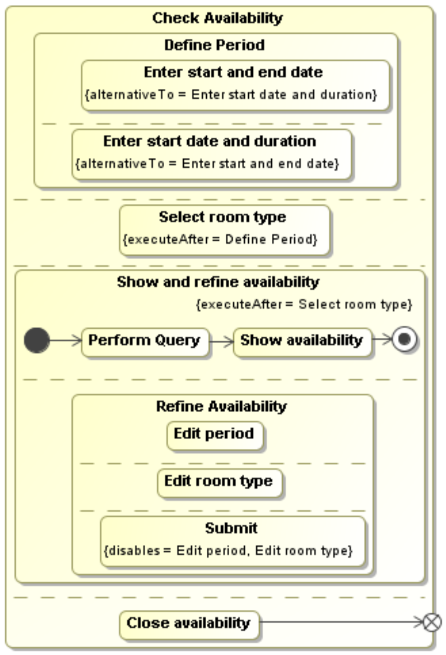

Check Availability
Define Period
Enter start and end date
{alternativeTo = Enter start date and duration}
Enter start date and duration
{alternativeTo = Enter start and end date}
Select room type
{executeAfter = Define Period}
Show and refine availability
{executeAfter = Select room type}
Perform Query
Show availability
Refine Availability
Edit period
Edit room type
Submit
{disables = Edit period, Edit room type}
Close availability

Fig. 13. Simplified notation of Fig. 11 using the stereotype << `task`>>

Fig. 13 shows the simplified version of the state machine in Fig. 11. This diagram is clearly more readable, because the added complexity of the substates and associated transitions of the task state *active* is removed from the diagram. Since all states have the stereotype << `task`>> applied to them, this stereotype is not shown in the diagram. For those states that have a tagged value that contains one or more values, the name of the tagged value as well as the values are shown below the state name between parentheses. For states whose corresponding task is `optional` or `nonInterruptable` or `repeatable` only the name of the tagged value will be shown. Note that there is no such task in this example.

Taking into account the concrete semantics for the task states presented in section 4 and table 1, we can state that we can consider the diagram in Fig. 13 to be a high-level dialog model. A complex state can correspond to a single dialog or window or a part thereof. Fig. 13 can thus describe the dynamic composition of a single window.

7 Related work

One can find several approaches to define the semantics of the temporal operators of the CTT in literature. Some provide an informal definition of the temporal operators such as Mori et al. [7]. They also present an algorithm to transform the CTT to a set of enabled task sets (ETSs). Mori et al. [8] also propose an abstract user interface model that contains a dialogue model description. This notation is based on task sets and transition tasks.

A more formal definition of the CTT is given by Luyten al. [6] who use these definitions to define an alternative transformation algorithm from the CTT to a set of ETSs. They do not give semantics of the temporal operators except that two of them cause transitions: the enabling and deactivation operators.

Both aforementioned approaches do not support nested states, which means that merging task sets creates inconsistencies between the task model and abstract user interface model.

Nobrega et al. [9] provide a mapping of the CTT to UML 2.0. They define the semantics of most of the operators by defining a mapping to UML 2.0 activity diagrams. In contrast to this work, they do not provide a definition for the suspend/resume operator. They do propose an extension to UML, with a hierarchical task notation, which reuses as much symbols of UML as possible for the newly introduced concepts. This notation is, however, not used to derive further specifications, such as a dialog or abstract user interface model.

Elkoutbi et al. [1] propose a semi-automated approach to derive interactive prototypes from scenarios specified using UML use cases, class diagrams and collaboration diagrams. This approach uses statecharts as an intermediate step to specify the behavior of the interactive prototype. Their approach shows the capabilities of the UML state machines (with nested states) as a specification language that can be used for generation of user interface prototypes.

8 Conclusion

In this paper we proposed a general description of the task execution cycle using UML state machines. We described the influence of the temporal operators on this description. An example that combined the states for a complete task model into one stage machine demonstrated the complexity of the notation for larger compositions. We thus proposed an abbreviated notation for this integrated notation using a small UML profile. This profile adds extra semantics to the states, which can be used to generate the complete specification. The support for nested states offers enhanced expressiveness over other solutions such as state transition networks.

The usage of UML enables the application of proven transformation tools to be applied on the models to generate dialog models at different levels of abstractions and adapted to different contexts of use. Further exploration of this route is planned as future work. Building on the work of [15] would allow to exploit all formal work done on petri nets.

Acknowledgements Part of the research at EDM is funded by EFRD (European Fund for Regional Development), the Flemish Government and the Flemish Interdisciplinary institute for Broadband Technology (IBBT).

References

1. Mohammed Elkoutbi, Ismaïl Khriss, and Rudolf Keller. Automated prototyping of user interfaces based on uml scenarios. *Automated Software Engineering*, 13(1):5–40, January 2006.
2. David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
3. Ian Horrocks. *Constructing the User Interface with Statecharts*. Addison-Wesley Professional, 1999.
4. Quentin Limbourg. *Multi-path development of User Interfaces*. PhD thesis, Université Catholique de Louvain, 2004.
5. L. Logrippo, M. Faci, and M. Haj-Hussein. An Introduction to LOTOS: Learning by Examples. *Computer Networks and ISDN Systems*, 23(5):325–342, 1991.
6. Kris Luyten, Tim Clerckx, and Karin Coninx. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In *Interactive Systems: Design, Specification, and Verification*, volume 2844 of *LNCS*, pages 203–217. Springer, 2003.
7. Giulio Mori, Fabio Paternò, and Carmen Santoro. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, 28(8):797–813, 2002.
8. Giulio Mori, Fabio Paternò, and Carmen Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Sofware Engineering*, 30(8):507–520, August 2004.
9. Leonel Nobrega, Nuno Jardim Nunes, and Helder Coelho. Mapping concurtasktrees into uml 2.0. In *Proceedings of DSV-IS 05*, 2005.
10. Nuno Jardim Nunes and João Falcão e Cunha. Towards a uml profile for interaction design: the wisdom approach. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard.*, volume 1939 of *LNCS*, pages 101–116. Springer, October 2000.
11. Object Management Group. *UML 2.0 Superstructure Specification*, October 8 2004.
12. Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.
13. Fabio Paternò and Carmen Santoro. One model, many interfaces. In Christophe Kolski and Jean Vanderdonckt, editors, *CADUI 2002*, volume 3, pages 143–154. Kluwer Academic, 2002.
14. Stefan Sauer, Marcus Dürksen, Alexander Gebel, and Dennis Hannwacker. Guibuilder - a tool for model-driven development of multimedia user interfaces. In *Proceedings of the MoDELS'06 Workshop on Model Driven Development of Advanced User Interfaces*, volume 214 of *CEUR-WS.org*. http://CEUR-WS.org/Vol-214/, 2006.
15. J. Trowitzsch and A. Zimmermann. Using uml state machines and petri nets for the quantitative investigation of etcs. In *valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, page 34, New York, NY, USA, 2006. ACM Press.
16. Jan Van den Bergh. *High-Level User Interface Models for Model-Driven Design of Context-Sensitive Interactive Applications*. PhD thesis, Hasselt University (transnationale Universiteit Limburg), October 2006.