Journal of Universal Computer Science, vol. 14, no. 19 (2008), 3188-3201
submitted: 23/7/08, accepted: 29/10/08, appeared: 1/11/08 © J.UCS

CTML: Domain and Task Modeling for Collaborative Environments

Maik Wurdel
(University of Rostock, Rostock, Germany
maik.wurdel@uni-rostock.de)

Daniel Sinnig
(Concordia University, Montreal, Canada
d_sinnig@encs.concordia.ca)

Peter Forbrig
(University of Rostock, Rostock, Germany
peter.forbrig@uni-rostock.de)

Abstract: A precise model of the behavioral dynamics is a necessary precondition for the development of collaborative environments. In this paper we present a specification framework for collaborative environments. In particular we highlight the interplay of task specifications and domain models. The framework consists of two components: A formal specification language (called CTML) and the tool CTML Editor and Simulator. CTML has a precisely defined syntax and semantics and is designed to model actors, roles, collaborative tasks and their dependency and impact on the domain. The CTML Editor and Simulator is an Eclipse IDE for the interactive creation and simulation of CTML specifications.

Keywords: Task Modeling, Domain Modeling, Collaborative Environments
Categories: D.2.1, D.2.2

1 Introduction

In a collaborative environment tasks are barely carried out in isolation, but have to synchronize with other users' tasks. Cooperation among actors is needed to accomplish personal as well as team goals. Some tasks cannot be started while others are still in progress. Imagine the following scenario of a conference session:

The session chair Dr. Smith introduces herself and defines the topic of the session. Afterward she gives the floor to the first speaker who sets up her equipment, the laptop switches to presentation mode and the speaker starts with the talk. During the presentation the audience accesses additional information related to the talk using their personal devices. After finishing the talk the chairman asks for questions from the plenum which are answered by the speaker. Eventually the chairman closes the talk and announces the next one. Subsequent talks are given in the same manner until the chairman encourages an open discussion, wraps up the session and finally closes it.

By analyzing the scenario one can detect several important aspects to consider. The scenario includes multiple actors (Dr. Smith, each presenter and the listeners), whose behavior is characterized by the role they are fulfilling (chairman, listener, presenter). Key domain concepts are the conference session itself, as well as the interactively given presentation. Additionally each presenter is equipped with a notebook which is used to show the slides after connecting to the projector. Personal devices are employed to support the tasks the actors are currently performing.

Task performance of users is usually embedded in a social context consisting of team and individual goals [Schmidt, 98]. Team goals, such as "performing the conference session" can be decomposed into individual goals like "giving my talk" and "manage the session"; these goals, however, cannot be accomplished autonomously but require cooperation.

In this paper we present a specification framework for collaborative environments. The framework consists of two main components: (1) A formal task-based specification language CTML and (2) an interactive editor and simulator. CTML is able to model dependencies between users, devices and domain objects based on team and individual goals defined at design time. The editor and simulator tool provides software engineers with a tool environment to create and test the specification.

The framework will be particularly useful for the analysis and specification of interactive support for collaborative environments. A CTML specification can serve as a formal (analysis) model capturing the behavioral dynamics involved in collaborative environments. Based on the analysis CTML model, a requirements CTML model is developed, which captures the envisioned interactive tool support for the collaborative environment. For each model, the accuracy and/or usability can be validated by using the CTML Simulator. Based on our experience we found that simulation is a very helpful validation device. It helps to gain understanding about the collaborations involved as well as triggers invaluable feedback from stakeholder's about the elicited requirements.

Preliminary results towards the development of the framework were reported in [Wurdel, 08a; Wurdel, 08c]. While [Wurdel, 08c] focuses on the definition of syntax and semantics of CTML, [Wurdel, 08a] defines a development methodology and a suite of refinement relations for CTML models. The primary purpose of this paper is to provide a comprehensive overview of the specification framework. Moreover, a domain model component is added to the framework, which allows the specification of preconditions and effects based on the objects of the domain.

The remainder of this paper is structured as follows: In the next section we cover related approaches in task modeling and highlight their features and limitations. Next, in Section 3 we distill a set of requirements that should be addressed by our specification framework for collaborative environments. Section 4 introduces the collaborative task modeling language (CTML). The usage of CTML within the development lifecycle and its corresponding tool support, namely the CTML Editor and Simulator, is presented in Section 5 and 6. Finally we conclude and provide an outlook to future avenues.

2 Background and Related Work

In this section the reader is reminded of core concepts involved in classical task modeling. We examine existing approaches in collaborative task model modeling and review commonly used formalisms and relevant related work.

Task analysis and task modeling is a well-established research field in HCI. Various notations for task modeling exist. Among the most popular ones are GOMS [John and Kieras, 96], HTA [Annett and Duncan, 67], CTT [Paterno, 99], and WTM [Bomsdorf, 07]. Even though all notations differ in terms of presentation, level of formality and expressiveness, they assume the following common tenet: tasks are performed to achieve a certain goal. Moreover, complex tasks are decomposed into more basic tasks until an atomic level has been reached. Within the domain of HCI, CTT is the most popular notation, as it contains the richest set of temporal operators and it is supported by a tool, CTTE [Mori, 02], which facilitates the creation, visualization and sharing of task models. A comprehensive overview on CTT can be found in [Paterno, 99].

In order to support the specification of collaborative (multi-user) interactive systems, CTT has been extended to CCTT (Collaborative ConcurTaskTrees) [Mori, 02]. Similar to the corporative task modeling language presented in this paper, CCTT uses a role-based approach. A CCTT specification consists of multiple task trees. One task tree for each involved user role and one task tree that acts as a “coordinator” and specifies the collaboration and global interaction between involved user roles. Main shortcoming of CCTT is that the language does not provide means to model several actors, who simultaneously fulfill the same role.

Bomsdorf [Bomsdorf, 07] as well as Klug and Kangasharju [Klug and Kangasharju, 05] introduced an extension to task models where a task is not regarded as an atomic entity (like in CTT) but has a complex lifecycle, modeled by a so-called task state machine. The former approach does not consider a temporal operator as state chart whereas the latter does not consider abortion or skipping of tasks. In this paper we adopt the same idea for the definition of CTML but try to overcome the limitations mentioned above. Additionally, we integrate instruments to model collaboration. We believe that in this way tasks become more expressive which makes them suitable as runtime specifications and allow for more realistic model simulations.

Tasks are always performed within a certain context or environment and hence their interplay with the environment should also be taken into account. This issue was first tackled by Dittmar and Forbrig [Dittmar and Forbrig, 03] who proposed modeling the execution environment in accordance with the task specification. The environment captures the domain entities which are manipulated, created or needed for the performance of a certain task. Unfortunately the approach by Dittmar and Forbrig is not very well integrated with standard software engineering models. We try to overcome this shortcoming by proposing UML class diagrams [UML, 07] as a notation to model the environment (domain) model while instances of the model are represented using UML object diagrams [UML, 07].

In [Molina, 08], Molina et al. propose a generic methodology for the development of groupware user interfaces. The approach defines several interrelated models capturing roles, tasks, interactions, domain objects and presentation elements. Even though, the models cover all important aspects involved in groupware user interfaces, they are only used at the analysis stage. Subsequent development phases (e.g. requirements or design) are not covered. The methodology is not assisted by a tool which would facilitate the creation and simulation of the various models. In particular, the latter is an important shortcoming since the animation of models is an important technique to obtain stakeholder's feedback. The same disadvantages apply to the work of [Penichet, 08] where a modeling approach for collaborative systems is presented. The approach only covers the analysis phase. An execution semantics for the various models is not defined.

According to [Garrido and Gea, 02], the most important aspects for the development of interactive systems for collaborative environments are user groups, roles and tasks. In their approach, groups and roles are modeled using a state charts whereas the definition of a tasks is specified by activity diagrams. As semantic domain Petri nets [Petri, 62] have been chosen, which allow the animation of the models as well as the verification of properties. The behavioral specification of this approach is sound but lacks the integration with the domain model which is an important aspect to consider when developing those systems. A development methodology has not been defined.

3 Requirements of a Formal Specification Framework

In this section, we compile a set of requirements that are particular to a specification framework for collaborative applications. The requirements are summarized in the following six categories:

1. *Task-Driven Methodology.* The concept of a "Task" is central to collaborative environments. Typically various actors collaborate and interact with each other by sharing, synchronizing on, and triggering common and related tasks. Therefore, we believe that a specification framework for collaborative environments should be built around the concept of a task. It should furthermore intrinsically support well-known task related concepts such as decomposition into subtasks and temporal constraints.
2. *Modeling Cooperation.* A single task model specifies synchronization of tasks within this isolated task model. (e.g., a certain task 'A' has to be performed before task 'B' can be started). In collaborative scenarios the inter-dependencies among tasks are more complex and more expressive synchronization constructs are needed to coordinate task performance. Examples of such constructs are preconditions and effects. The former denote additional constraints defined over the state of the collaborative environment whereas the latter define state modifications as a result of task executions.
3. *Modeling the Domain.* The execution of tasks may depend on the states of domain objects (precondition). Furthermore, the execution of tasks may manipulate the state of domain objects (effect). Such domain objects (and their interrelations) are specified in the domain model. Incorporating the domain objects into task modeling enriches the framework and makes it much more expressive.

4. *Formal Syntax and Semantics.* In order to make effective use of the task specification language, a precisely defined syntax and semantics is needed. An underlying formal model will not only rule out ambiguities but is also an obvious precondition for sophisticated tool support.
5. *Support for Iterative and Incremental Development.* In general, software development consists of a series of transformations in which models are iteratively refined [Larman, 04; Sommerville, 06]. Modeling collaborative environments is no exception to this rule. Often a coarse grained, even incomplete specification is successively transformed into more complete fine-grained specifications. With each transformation step it is important to ensure that the resulting model is a valid refinement of the based specification.
6. *Tool Support.* Another key requirement is tool support that assists developers in handling collaborative task model specifications. In particular, tools can facilitate the actual creation of the collaborative model, perform automated refinement checks, and simulate/animate the specification.

All before-mentioned requirements are tackled by our specification framework. In the next section we introduce the collaborative task modeling language which intrinsically addresses requirements 1.- 4. In Section 5 we introduce the CTML Editor and CTML Simulator which is used to create and animate collaborative task specifications (6). Section 6 proposes an iterative lifecycle of CTML specifications (5).

4 The Collaborative Task Modeling Language

In this section we present the collaborative task modeling language (CTML). We first define the syntax of CTML, explain its design rationale and provide an example. Then we present the semantics of collaborative task expressions and collaborative task models. For the sake of brevity, only semi-formal definitions are given in this paper. A more rigorous language specification of CTML can be found in [Wurdel, 08c].

4.1 Syntax and Design Rationale

The design of CTML is based on three fundamental assumptions:

1. In limited and well-defined domains the behavior of an actor can be approximated through her role.
2. The behavior of each role can be adequately expressed by an associated collaborative task expression.
3. The execution of tasks may depend on the current state of the environment (defined as the accumulation of the states of all available domain objects) and in turn may lead to a state modification.

Based on these assumptions we define a collaborative task model as a tuple consisting of a *set of actors*, a *set of roles* and a *set of collaborative task expressions* (one for each role) and a *set of domain objects* defined by a *domain model*. Each actor belongs to one or more role(s).

Each collaborative task expression has the form of a task tree in a CTT-like [Paterno, 99] notation. Each task is attributed with a (unique) identifier, a precondition and an effect. *Preconditions* add additional execution constraints to a task as a task may only be performed if its precondition is satisfied. Conditions can either be defined over the domain model, denoting the system state, or the state of other tasks, which potentially may be part of another task expression. An *effect* denotes a state change of the system or environment as a result of a task execution. Both, preconditions and effects are needed to model collaboration and synchronization across collaborative task expressions. They also denote the binding to the domain.

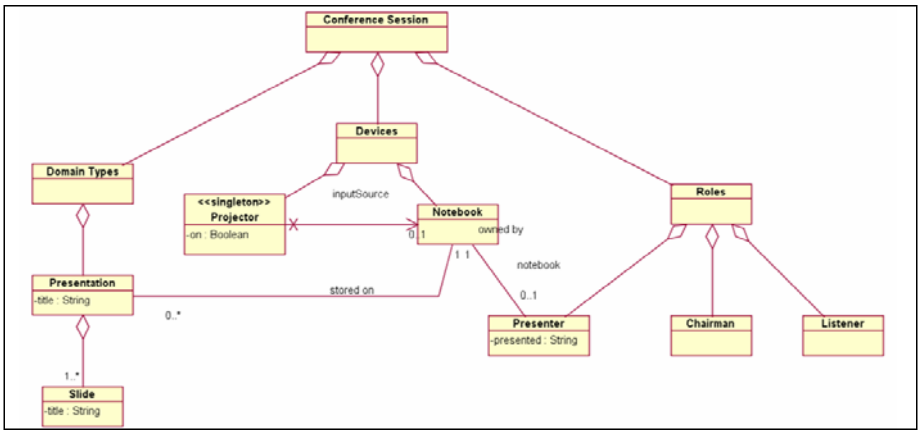


Figure 1: Domain Model for "Session" Scenario

The *domain model* captures domain specific concepts and associates them to relevant roles in the collaboration. It is represented using an UML class diagram. Instances of the class diagram can be loaded during animation (see Section 5) as concrete configurations of the environment. These can be visualized by UML object diagrams.

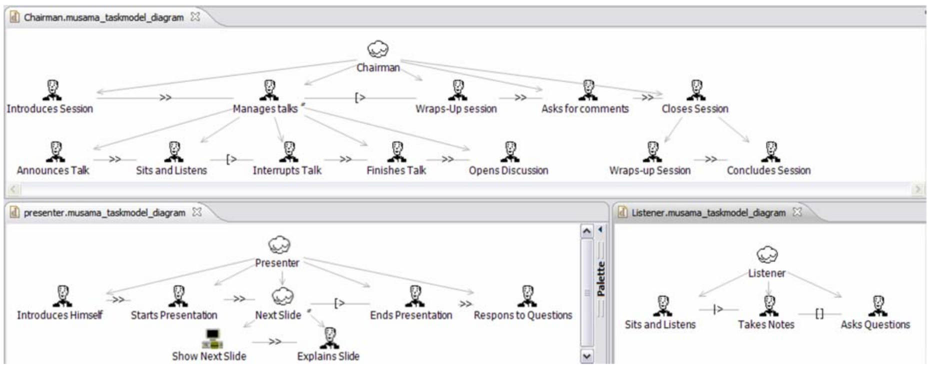


Figure 2: Specification of the “Session” Scenario Using the CTML Editor

Figure 1 and Figure 2 depict specifications for the introductory example of Section 1. In particular Figure 1 shows the domain model whereas Figure 2 illustrates the collaborative task expressions for the involved roles for the given scenario. The collaborative task model was interactively created using the tool CTML Editor (will be presented in Section 5). For the sake of readability for each task only the hierarchical breakdown and temporal relations are shown. Preconditions and effects have been omitted.

4.2 Semantics

We start with defining the execution semantics of a single collaborative task expression. The execution order of the tasks is determined by the following three criteria: (1) The defined temporal operators, (2) the task-subtasks decomposition, and (3) the preconditions defined for each task.

In order to illustrate the interplay of all three criteria, let us consider the lifecycle of a generic task. As depicted in Figure 3 each task starts in the state *disabled*. Upon receiving the message "enable" a task moves from state *disabled* to *enabled*. If, and when, an "enable" message is sent depends on the super-ordinate temporal operator as well as the task state of the sibling tasks. Table 1 gives an intuitive definition of the semantics of all temporal operators defined in CTML. Note that most of the operators (except for instance iteration) have similar counterparts in CTT [Paterno, 99]. Upon receive of message "start" an enabled task starts executing by transiting into state *running*, given that its precondition is satisfied. In state *running* the task executes its predefined action (denoted by "do/action") and its effect to the environment becomes externally visible. A successful run of the task is denoted by the "end" transition to state *completed*.

At any time a task may be prematurely aborted, as a result of the disabling operator (see Table 1 for details). A task that is *enabled*, or already *running* can be *suspended* upon receive of the "suspend" message. Once a task is *suspended* it returns back to its previous state when it receives "resume".

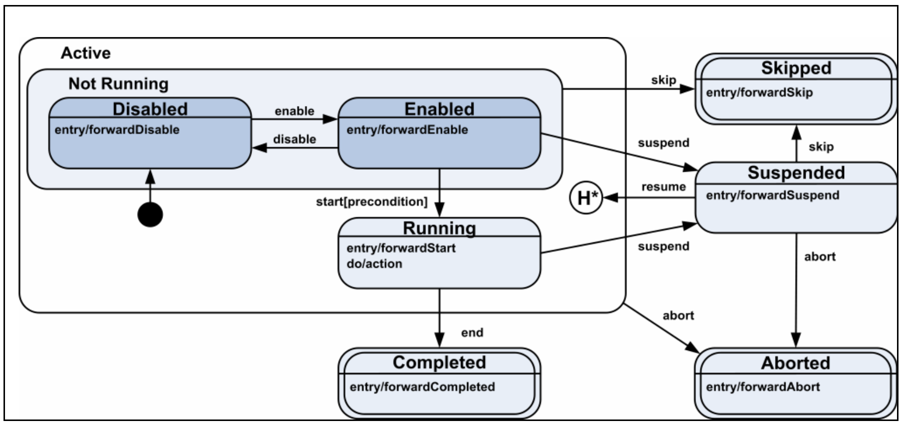


Figure 3: Generic Task State Chart

As long as a task is not started, it can be skipped, which is either due to an optional (opt), iterative (*) or choice ([]) operator. Additionally a task may be skipped when a super-ordinate task becomes skipped or disabled. Note that each state of the task state chart is equipped with so called entry actions whose purpose is to notify the state charts of sub- and super-ordinate tasks of state changes. This implements an update mechanism to assure synchronization between all state charts.

N-Ary Operators ($t_1, t_2, \ldots, t_n$)	
Choice ([])	Only one operand task is executed
Order Independence (|=|)	Operand tasks are executed in any order with no interleaving of subtasks
Concurrent (|||)	Interleaved execution of operand tasks and their subtasks
Enabling (>>)	Operand tasks are executed sequentially
Binary Operators (t_l, t_r)	
Disabling ([>))	Execution of t_l is aborted as soon as t_r is started
Suspend/Resume (|>)	At any time the execution of t_l may be interrupted by t_r. After t_r has finished its execution t_l resumes.
Unary Operators (t)	
Iteration (*)	Repetitive execution of t
Instance Iteration (#)	Interleaved repetitive execution of *t*. Please consult [Sinnig, 07] for more details
Optional (opt)	Execution of t is optional

Table 1: Semantics of CTML Operators

In CTML, not only each task but also each temporal operator is represented by a state chart which formally implements the semantics given in Table 1.

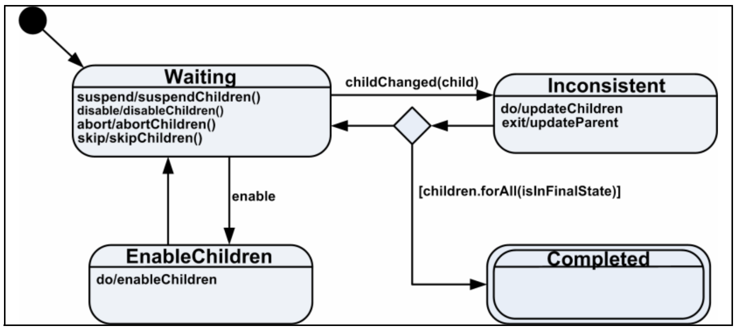


Figure 4: Generic Temporal Operator State Chart

In Figure 4 the generic state chart for a temporal operator is given. It starts in state *waiting* in which messages from superordinate state charts are dispatched to its children. An exception to this rule is the “enable” message which triggers a state transition to state *enableChildren* in which the set of subordinate task state charts are enabled according to the semantics of the operator (e.g. in case of the choice operator ([]) each child becomes *enabled* whereas in case of the enabling operator (>>) only the first child becomes *enabled*). Figure 4 also portrays how the operator state chart handles a temporally *inconsistent* state which is due to a state change of a child. Depending on the current state of the changing child the state chart implements the semantics of each temporal operator (e.g. if a child task of the choice operator is executed all other children become *skipped*, whereas if a child task of an enabling operator is executed the next sibling task becomes *enabled*). The operator state chart changes its state to *completed*, if, and only if, all children state charts are in a final state. Otherwise it returns to the state *waiting*. By mapping each task and operator to a state chart a network of communicating state charts is created, where operator state charts mediate messages between task state charts of adjacent levels of abstraction.

This far, we have defined the execution semantics of individual collaborative task expressions. We now continue with the definition of semantics of a collaborative task model. Thereby, the main principle is as follows: (1) For each role, based on the associated task expression, we create a network of communication state machines (as shown previously). (2) For each actor, an individual copy (instance) of the corresponding role state machine network is created. (3) The resulting state machine networks are composed and run concurrently at simulation time. In essence, a collaborative task model is transformed into a set of concurrently running networks consisting of task state machines and operator state machines.

4.2.1 Preconditions

Synchronization across networks is achieved by means of preconditions and effects. Formally, preconditions are logical statements which may contain *allInstances* and *oneInstance* quantifiers. Using the quantifiers, it is possible to either "refer" to all task models (state machine networks, respectively) that belong to a given role *or* one particular task model of a given role. The preconditions given in Table 2 represent a subset of the preconditions used to model our scenario and express the synchronization points between the task executions of different actors. The meaning of the presented statements is as follows:

1. A presenter is allowed to start her presentation after the chairman has announced the talk.
2. The listeners are allowed to ask questions after the chairman has opened the discussion session.
3. The chairman can wrap-up the session after all presenters have finished their talk.

#	Role	Task	Precondition
1.	*Presenter*	*StartsPresentation*	*Chairman.oneInstance.AnnoucesTalk.completed*
2.	*Listener*	*AsksQuestion*	*Chairman.oneInstance.OpensDiscussion.completed*
3.	*Chairman*	*Wraps-up Session*	*Presenter.allInstances.EndsPresentation.completed*

Table 2: Subset of Preconditions Used in "Session" Scenario

Additionally, OCL constraints [UML, 07] can be used to further constrain the potential task execution based on the domain model.

4.2.2 Effects

Effects are operational constructs denoting a state change as a consequence of a task execution. They can either affect the state of tasks (e.g. enabling/disabling/trigger tasks) or the state of the environment (by changing the state of objects or actors). For our example (Figure 2) we formalized the following effects:

1. After ending her presentation the Presenters' state is set to *presented*.
2. When the Presenter starts her talk the projector connects with the Presenters' notebook.

The formalizations are given in Table 3. Note that "this" denotes the actor currently performing the task in the style of common object oriented languages. Similar to OCL, point notation is used to refer to attributes or associated objects.

#	Role	Task	Effect
1.	*Presenter*	*EndsPresentation*	*this.presented=true*
2.	*Presenter*	*StartsPresentation*	*Projector.inputSource= this.notebook*

Table 3: Subset of Effects Used in "Session" Scenario

5 Tool Support: CTML Editor and Simulator

Having defined the syntax and semantics of CTML we now introduce the second component of our formal framework; the CTML Editor and Simulator. As hinted by the name, the tool was designed to assist software engineers with (1) the creation of CTML models and (2) the animation and interactive simulation of CTML models. In what follows, both key purposes are described in detail. As an illustrative example, we will use the "Session" scenario of Section 1.

The CTML Editor provides software engineers with an Eclipse-based IDE to specify roles, actors, domain entities (and their instances) (Figure 1) as well as collaborative task expressions (Figure 2). In case of the latter, the user conveniently defines the task-subtask hierarchy by "dragging" tasks into a task tree. Only tasks at the same hierarchy level can be related with certain temporal operators.

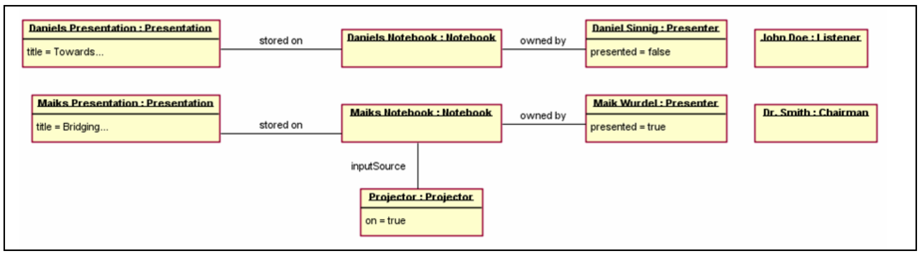


Figure 5: Object Diagram Specifying the Domain State

Since the tool is Eclipse-based, the software engineer can define the domain model using the provided EMF Ecore model editor [EMF, 07]. Instances, as shown in Figure 5, are created using EMF Ecore *instance* editor.

Please note that the domain model, as depicted in Figure 1, not only captures the properties of the entities involved (roles and domain objects), but also their interconnections using class associations. These can be used by preconditions and effects to navigate through the model, similar to the navigation in OCL [UML, 07].

Once a CTML model has been defined it is compiled into a network of communicating state machines. At this point the CTML Simulator can be used to interactively walk through the CTML model (Figure 6). At startup the simulator shows the various involved tasks in form of task trees. The icons defined for each task symbolize the current state of the respective task state machine. Hereby, each state of the state machine is mapped to a different symbol. Upon mouse-click the tool attempts to execute the respective task. This is however only possible if the corresponding task state machine is in state *enabled* and the assigned preconditions (in brackets after the task name) are fulfilled. Every task execution may trigger the execution of associated effects.

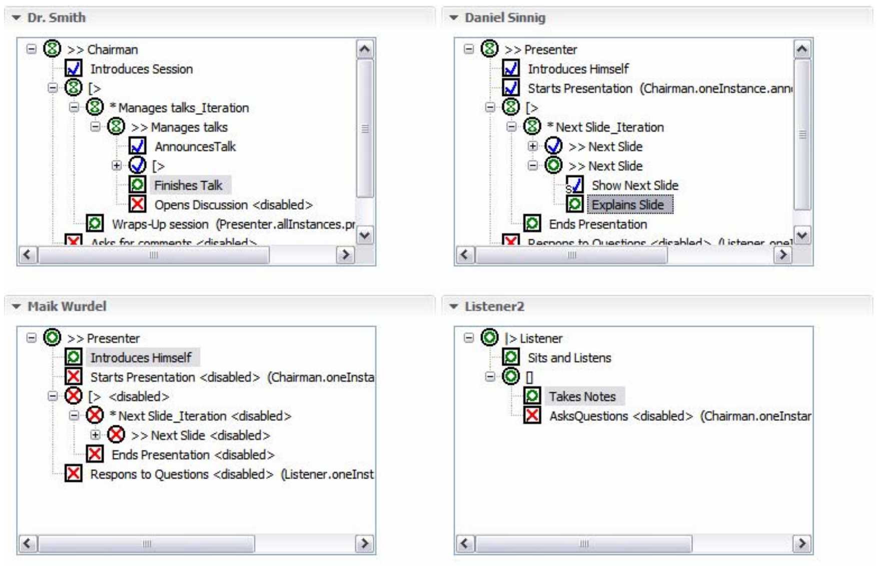

Dr. Smith
>> Chairman
Introduces Session
[>
* Manages talks_Iteration
>> Manages talks
AnnouncesTalk
[>
Finishes Talk
Opens Discussion <disabled>
Wraps-Up session (Presenter.allInstances.pr
Asks for comments <disabled>
Daniel Sinnig
>> Presenter
Introduces Himself
Starts Presentation (Chairman.oneInstance.ann
[>
* Next Slide_Iteration
>> Next Slide
>> Next Slide
Show Next Slide
Explains Slide
Ends Presentation
Respons to Questions <disabled> (Listener.one
Maik Wurdel
>> Presenter
Introduces Himself
Starts Presentation <disabled> (Chairman.oneInsta
[> <disabled>
* Next Slide_Iteration <disabled>
>> Next Slide <disabled>
Ends Presentation <disabled>
Respons to Questions <disabled> (Listener.oneInst
Listener2
|> Listener
Sits and Listens
[]
Takes Notes
AsksQuestions <disabled> (Chairman.oneInstar

Figure 6: Interactive Simulation of the CTML Model for the "Session" Scenario

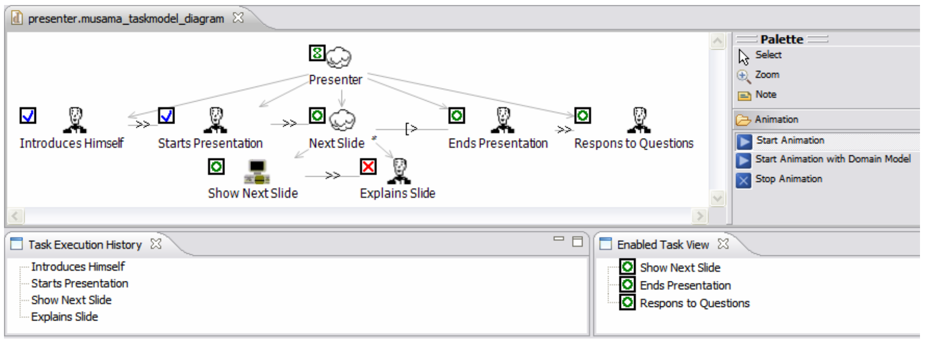
6 Development Life Cycle of CTML Models

In this section we propose a lifecycle for the iterative and incremental development of CTML models. We believe that a CTML model is best developed in five steps:

1. Definition of roles and corresponding collaborative task expressions
2. Animation and validation of these sub-specifications

3. Specification of the environment including actors, associated roles and the domain
4. Annotation of tasks with precondition and effects
5. Animation and validation of the entire specification

Each phase is fully supported by our tool environment. Note that instead of creating the entire model at once, which can be quite overwhelming, we suggest to first define (1.) and test (2.) the involved roles and their individual collaborative task expressions. Both steps can be performed iteratively. In case of an unsatisfying simulation the developer typically adapts the underlying specification and restarts the simulation. Additionally, refinement checks can be undertaken. This sequence is to be repeated until the simulation exhibits the expected behavior. The simulation of the task expression for the role "Presenter" is depicted in Figure 7. In the lower left side of the figure an execution history is offered as well as a summary of all currently enabled tasks.



Task Execution History:
- Introduces Himself
- Starts Presentation
- Show Next Slide
- Explains Slide

Enabled Task View:
- Show Next Slide
- Ends Presentation
- Respons to Questions

Figure 7: CTML Simulation of the "Session" Scenario

Next (3.) the designer defines the environment and involved actors (see Figure 1, in Section 4). Additionally (4.) task specifications are completed by adding preconditions and effects based on the analysis of the dependencies between actors and roles within the scenario. Finally (5.) the entire specification, consisting of several "concurrently" executing task expressions, can be tested and simulated (see Figure 6).

As common in modern software development processes, CTML specifications are developed iteratively and incrementally. Collaborative task models are gradual *refined* or adapted during the development life cycle based on the feedback obtained during animation. The question arises whether the refined model correctly implements the requirements of its base model. For this purpose we have developed a formal refinement relation for task models [Wurdel, 08b]. The proposed definition of refinement relies on the principle of *mandatory scenario equivalence* according to which the refining model must implement all mandatory task of the base model. Moreover, the relative temporal ordering of all mandatory tasks must be preserved. Which scenarios are mandatory is determined by the requirements engineer and/or domain expert. A formal definition, examples as well details about the implemented model checking algorithm can be found in [Wurdel, 08b].

7 Conclusion and Future Research

In this work we presented a specification framework for collaborative environments. First, we distilled a set of key requirements according to which the framework should include a task-based specification language which is capable of modeling cooperation among actors and offers capabilities of integrating the domain model. We argued that a formal syntax and semantics, an iterative development lifecycle as well as tool support are needed to make effective use of such a framework. Based on these requirements we presented our specification framework consisting of two components: the collaborative task modeling language (CTML) and the tool CTML Editor and Simulator. The design of CTML is based on the assumption that the behavior of an actor can be approximated through her role. The behavior of a role is defined by collaborative task expressions. Collaboration of actors is modeled by means of preconditions, effects and temporal operators. Preconditions and effects are either defined over the state of domain objects or the state of related tasks. In order to make CTML suitable for tool support and formal analysis, a rigorous syntax and semantics has been defined as well.

The tool CTML Editor and Simulator provides software engineers with a convenient way to handle CTML specifications. In particular, the tool can be used to create, validate and simulate collaborative task models. Based on our experiences, we determined that simulation is a very helpful validation device especially during early stages of development as it helps to gain understanding about the behavioral dynamics involved in the collaboration.

For future research we plan to further enhance our tool by integrating a model-checker component which will verify that certain properties of the CTML model (e.g. liveness, deadlock freedom) are fulfilled. The ultimate goal of our research is to use CTML specifications at runtime to track the task performance of users within a collaborative environment. This would enable us to deliver the appropriate support based on the current task performance of users. This requires an enhancement of our framework in terms of flexibility and vagueness. In real world scenarios users can leave the environment while new users may enter. The same applies for devices (e.g. a laptop runs out of battery). We are currently investigating strategies for incorporating such unforeseen events.

Acknowledgements

Maik Wurdel is supported by a grant of the German National Research Foundation (DFG), Graduate School 1424, Multimodal Smart Appliance Ensembles for Mobile Applications (MuSAMA)

References

[Annett and Duncan, 67] Annett, J. and K. D. Duncan (1967). "Task Analysis and Training Design." *Journal of Occupational Psychology* **41**: 211-221.

[Bomsdorf, 07] Bomsdorf, B. (2007). "The WebTaskModel Approach to Web Process Modelling." TaMoDia 2007 **4849**: 240-253.

[Dittmar and Forbrig, 03] Dittmar, A. and P. Forbrig (2003). Higher-order Task Models. DSV-IS 2003. Funchat , Portugal.

[EMF, 07] EMF. (2007). "Eclipse Modeling Framework." Retrieved September 23, 2008, from http://www.eclipse.org/emf.

[Garrido and Gea, 02] Garrido, J. L. and M. Gea (2002). A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling. DSV-IS 2002, Springer-Verlag.

[John and Kieras, 96] John, B. E. and D. E. Kieras (1996). "The GOMS family of user interface analysis techniques: comparison and contrast." *ACM Transactions on Computer-Human Interaction* **3**(4): 320-351.

[Klug and Kangasharju, 05] Klug, T. and J. Kangasharju (2005). Executable Task Models. TaMoDia 2005. Gdansk, Poland.

[Larman, 04] Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition), Prentice Hall PTR.

[Molina, 08] Molina, A. I., M. A. Redondo, M. Ortega and U. Hoppe (2008). "CIAM: A Methodology for the Development of Groupware User Interfaces." *Journal of Universal Computer Science* ***14****: 1435-1446.*

[Mori, 02] Mori, G., F. Paternò; and C. Santoro (2002). "CTTE: Support for Developing and Analyzing Task Models for Interactive System Design." *IEEE Trans. Softw. Eng.* **28**(8): 797-813.

[Paterno, 99] Paterno, F. (1999). Model-Based Design and Evaluation of Interactive Applications. London, UK, Springer-Verlag.

[Penichet, 08] Penichet, V. M. R., M. D. Lozano, J. A. Gallud and R. Tesoriero (2008). Analysis models for user interface development in collaborative systems. CADUI 2008. Alabcete, Spain.

[Petri, 62] Petri, C. A. (1962). Fundamentals of a theory of asynchronous information flow. IFIP Congress'62.

[Schmidt, 98] Schmidt, A., M. Beigl and H. W. Gellersen (1998). There is more to Context than Location. Karlsruhe, University of Karlsruhe.

[Sinnig, 07] Sinnig, D., M. Wurdel, P. Forbrig, P. Chalin and F. Khendek (2007). Practical Extensions for Task Models. TaMoDia 2007, Springer. **4849:** 42-55.

[Sommerville, 06] Sommerville, I. (2006). Software Engineering: (Update) (8th Edition) (International Computer Science). Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc.

[UML, 07] UML. (2007). "Unified Modeling Language." Retrieved July 10, 2007, from http://www.uml.org/.

[Wurdel, 08a] Wurdel, M., D. Sinnig and P. Forbrig (2008a). Task-Based Development Methodology for Collaborative Environments. Engineering Interactive Systems 2008. Pisa, Italy, Springer. **5247:** 118-125.

[Wurdel, 08b] Wurdel, M., D. Sinnig and P. Forbrig (2008b). Task Model Refinement with Meta Operators. DSV-IS 2007. Kingston, Canada.

[Wurdel, 08c] Wurdel, M., D. Sinnig and P. Forbrig (2008c). Towards a Formal Task-based Specification Framework for Collaborative Environments. CADUI 2008. Albacete, Spain.