Agile Software Development Process with Task Models having Triggers, Preconditions and Context Constraints

Peter Forbrig
University of Rostock
Albert-Einstein-Str. 22,
18051 Rostock
peter.forbrig@uni-rostock.de

Gregor Buchholz
University of Rostock
Albert-Einstein-Str. 22,
18051 Rostock
gregor.buchholz@uni-rostock.de

ABSTRACT

Agile software development asks for seamless integration of software engineering methods and approaches from HCI. A process model will be discussed that includes HCD as one sprint ahead in the corresponding process model. Additionally, model simulations are suggested for improved evaluation of design decisions.

The language CoTaSL (Cooperative Task Specification Language) is introduced for specifying the activities of users and certain context aspects. Its editor was developed based on the Xtext framework. The language is a human-centered form of CoTaL (Cooperative Task Language) that can be simulated by CoTaSE (Cooperative Task Simulation Environment). CoTaL and CoTaSE will be discussed as well.

Different approaches for specifying collaborative work are presented. Advantages and disadvantages are discussed. An outlook is provided that suggests some further support for CoTaSL by other tools.

Author Keywords

Agile Development Process, Task model, Constraints, Context, Coordination model, Team model, Role model.

ACM Classification Keywords

H.5 [Information Interfaces and Presentation]: H.5.2 User Interfaces – User-centred design; H.5.3 Group and Organization Interfaces – Theory and models.

General Terms

Human Factors; Requirements Specification, Simulation

INTRODUCTION

Currently, it is common ground that software should be developed iteratively in an agile way. This is necessary to improve the communication with the stakeholders. Otherwise, it is difficult or even impossible to capture the correct requirements. However, the classical waterfall model has some advantages for managing milestones. Therefore, a combination of both approaches makes sense that supports both the management aspect and the software development aspect.

In this paper, we will discuss the usage of extended task models that together with corresponding tool support can support a human-centered agile development process. The language CoTaSL was developed to allow specifications of collaborative activities. This is especially important for developing smart collaborative environments and workflow applications.

Each CoTaSL specification consists of one team model and several role models. The simulated team model instance reflects the state of the collaboration of the different role model instances.

First, we will focus on the integration of HCI aspects into a process model of agile software development. After that, the language CoTaSL will be introduced based on CoTaL and different simulation situations in CoTaSE.

SOFTWARE DEVELOPMENT PROCESS MODELS

Agile software development has become popular since the 1990s because many projects failed. The reason for that was that it took too much time from finalized requirements specification to first tests of the developed system. At the beginning of the 2000s a manifesto [1] was published that characterizes the agile idea by twelve main principles.

An overview of influences on agile software development is presented by Fig. 1. It shows approaches from planning, analysis, design, build, test, deploy, and review that found their way into the agile approach.

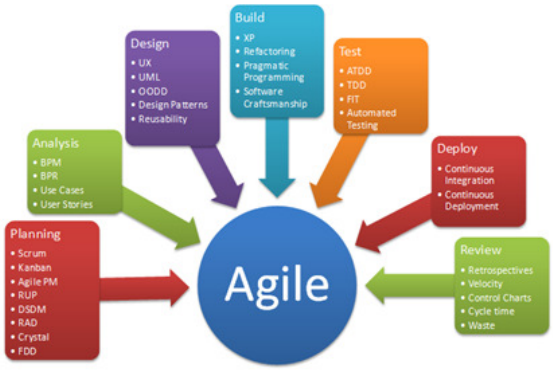

Figure 1. Influences to agile software development[1]

[1]http://blog.kiandra.com.au/wp-content/uploads/2013/08/Agile-21.png

Currently, one of the most popular agile software development approaches is SCRUM [19]. It is quite good characterized by the following Fig. 2.

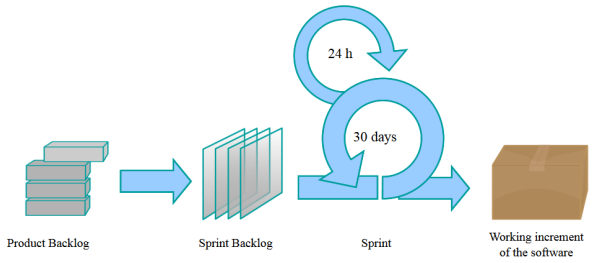

Figure 2. Model for the SCRUM development process

Originally, the process model of SCRUM (like software engineering in general) focused on software development activities only. However, usability aspects are important as well. Therefore, the integration of HCI aspects into the development process is discussed by several authors.

Two interleaving processes for developers and HCI specialists are suggested by Sy [21]. She suggest the gathering of some user data and the establishment of a common plan at the very beginning of a project. This has to be done by all participants. Developers start in the first development cycle with implementations that are independent from the user interface. This could e.g. be certain internal services of the application. At the same time, HCI specialists provide design solutions for cycle two and gather customer data for cycle three.

Developers implement the design solutions from cycle one in cycle two and in parallel their code from cycle one is tested by HCI experts. Additionally, they design for the next cycle and analyze for the cycle after the next cycle. This is the general development pattern. In this way, HCI specialists work two cycles ahead to developers in analyzing customer data and one cycle ahead in developing design solutions. Paelke et al. [16] published another process model for combining agile software development and agile user-centered activities.

HCD (Human-Centered Design) [12] is popular among usability and user experience experts in the same way as SCRUM with software developers. It provides an iterative approach for requirements gathering and considers the context of use, the requirements of the users and the evaluation of design solutions. The HCD process has been standardized by ISO 9241-210. Fig. 3 presents a part of an agile process model from [9] that includes an adapted HCD process model. It combines the ideas of HCD, Sy [21] and Paelke et al. [16]. A first version of the process model was presented in [8].

Currently user stories, scenarios, or use-case slices are used to capture requirements of applications. However, sometimes more elaborate activities for specifying activities are helpful to explore an application domain. Task models have been proven a useful tool for HCD. However, sometimes detailed features of the models are missing.

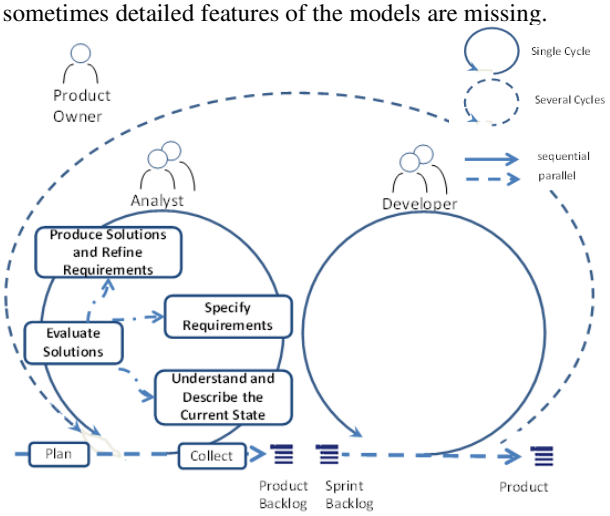

Figure 3. HCD details for SCRUM model sprints

The next paragraph will discuss a special version of task models that allows the creation of instances and the specification of context dependencies. This is especially important for specifications in smart environments or applications following a complex workflow.

SPECIFICATION OF SEQUNCES OF ACTIONS

Task Model

Sequences of actions can be specified by use cases [22] or business process models in the notation of BPMN [4] or S-BPM [6]. However, in the field of software ergonomics task models have proved to be helpful specifications [17]. TKS [13] and HTA [2] were one of the first approaches to specify possible task executions by models. Currently, systems like CTTE [5] and HAMSTERS [11] exist that allow the generation of other models and the cooperation with different other systems. Additionally, they allow the simulation of task models. Collaboration specification was discussed in [14], [18] and [20]. This allows very good stakeholder involvement at early stages of software development, which can be considered to be a substantial contribution to user centered development methodologies.

However, our application domain of “Smart Environments” asks for features that neither the cooperation model of CTTE nor the task groups in HAMSTERSs fulfill. This includes, first and foremost, the opportunity to have a variable number of task model instances. It is not known before the actual execution how many actors will collaborate. The current systems do not have this flexibility. Additionally, we require the temporal relation “instance iteration” that means the second iteration of a task can be started before the first ends. Task models need this feature for practical applications. The importance of models was already discussed in [7].

The specification language CoTaL (Cooperative Task Language) and a corresponding interpretation environment CoTaSE (Cooperative Task Simulation Environment) were developed for this reason. Some ideas were taken from the predecessor language CTML [23]. The separation of task models into several role models and one team model is one of the ideas of CTML that we included into CoTaL. The activity of a certain kind of actor (in terms of use cases), subject (in terms of S-BPMN) or pool (in terms of BPMN) is modelled as a role. During runtime several instances of a role model can exist. This is of course also true for a simulation. The instances specify the state of the task performance of a specific person (actor instance). The instantiation of a task model immediately delivers a stateful "running" model instance. This is very similar to the relation of classes and objects in object-oriented programs.

The cooperation of related role instances is reflected by the instance of a team model. This model may vary in its level of detail. It is thus possible to reflect each and every task of all role models. However, it is also viable to have one single team task only. The state of this task model shows when the collaborative activities are running and when they are finished. The optimal solution seems to be something in between those two extremes.

The two kinds of task models will be discussed within the following paragraph by an example, which assumes that a conference session has to be organized in a smart meeting room. It is further assumed that there are only the two roles of "Chair" and "Presenter". The chair has to introduce a session first, followed by the presenters. After all presentations of the conference session are finished a joined discussion will be performed. We will first have a look at the structure of the team model and will later discuss the structure of role models.

Team Model

The cooperation of actors is specified by a team model. The model specifies all roles involved and specifies events coming from individual task model instances. Activities of actors can be represented by different task model instances simultaneously. For example, a person can have the role of a chair and the role of a presenter at the same time.

Let us have a look at a simple example of performing a conference session. A chair has to manage the presentations first and afterwards the discussion. The task "Manage presentation" consists of two subtasks "Introduce talk" by the chair followed by "Present" assigned to the presenter.

In the following two visual representations of the corresponding team will be shown. They do not show who executed or is intended to perform a tasks. The visualization of task tree instances uses different colors and symbols to illustrate the state of the respective task. The set of possible task states include *disabled* (✕, red), *enabled* (○, yellow), *running* (🏃, green), *skipped* (↴, gray) and *finished* (✓, white). Please keep in mind that only leaf tasks can be started directly. Therefore, inner nodes are depicted in a slightly paler colour shade. Their state is derived bottom-up from their respective child tasks. Fig. 4 provides a visualization of a model instance that consists of the team model.

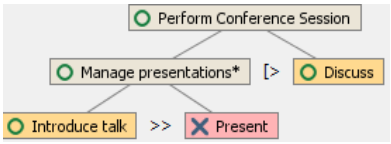

Figure 4. Session example of a task tree instance.

According to the model instance's current state, a talk can be introduced or the discussion can be started. A talk has to be introduced first and can only be presented afterwards (>>, enable). To perform a conference session several presentations have to be managed (*, iteration). The sequence of presentations is stopped ([>, deactivate) by starting the discussion. Fig. 4 presents the task visualization like CTTE. We will call it task tree. An alternative presentation like in HAMSTERS, where temporal operators are represented as nodes is shown in Fig. 5.

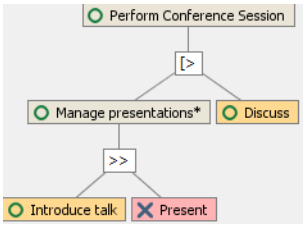

Figure 5. Session example of a syntactic tree instance.

The visualization of the task model in Fig. 5 is called syntactic tree. Within the CoTaSE tool both representations of task model instances can be shown alternatively or even together. The XML specification of the team model is the following one.

```
<?xml version="1.0" encoding="UTF-8"?>
<taskmodel name="Performing a conference" role="Team">
  <roles>
    <role name="Chair" file="Chair.xml"/>
    <role name="Presenter" file="Presenter.xml"/>
  </roles>
  <task name="Perform Conference Session">
    <task name="Manage presentations"
       operator="disabling" iterative="true">
      <task name="Introduce talk" operator="enabling"
        startTrigger=
          "Chair.oneInstance.IntroducePresenter.start"
        endTrigger=
          "Chair.oneInstance.IntroducePresenter.end">
      </task>
      <task name="Present"
        startTrigger=
          "Presenter.oneInstance.IntroduceTalk.start"
        endTrigger=
          "Presenter.oneInstance.EndTalk.end">
      </task>
    </task>
    <task name="Discuss"
      startTrigger="Presenter.allInstances.EndTalk.end"
      endTrigger="Chair.allInstances.CloseSession.end">
    </task>
  </task>
</taskmodel>
```

For each leaf task in the team model there exists a start trigger and an end trigger. Triggers are specified in an OCL-like notation [15]. The first part of the expression is a role name that is followed by "oneInstance" or "allInstances". This is followed by a task that is started or ended. The task "Introduce talk" is started when an instance of the role "Chair" has started the task "Introduce presenter" (*Introduce presenter* is equal to *IntroducePresenter*). It ends when the introduction is ended. A presentation is started, when one presenter started to present his or her talk. The discussion starts when all talks were presented. The conference session ends when all "Chairs" closed the session.

To avoid the XML details of CoTaL the domain specific language CoTaSL (Cooperative Task Specification Language) was designed based on Xtext [25]. Xtext generates a syntax driven text editor that can be combined with Xtend [24] to generate CoTaL specifications.

The above example of CoTaL looks like the following specification.

```
team for t {
  task Perform_Conference_Session = Manage_presentation{*} [> Discuss;
    task Manage_presentation = Introduce_talk >> Present;
      task Introduce_talk;
        trig Introduce_talk = ( start.Chair.oneInstance.Introduce_presenter.start );
        trig Introduce_talk = ( end.Chair.oneInstance.Introduce_presenter.end );
      task Present;
        trig Present = ( start.Presenter.oneInstance.Introduce_talk.start );
        trig Present = ( end.Presenter.oneInstance.Give_talk.end );
      task Discuss;
        trig Discuss = ( start.Presenter.allInstances.Give_talk.end );
        trig Discuss = ( end.Chair.oneInstance.Close_session.end );
}
```

It is assumed that it is much easier to use the domain specific language CoTaSL than CoTaL, which is optimized for machine interpretation while CoTaSL is intended for human usage.

Role Model

A role model specifies the activities of one role. In our example, we need one for the role "Chair" and one for the role "Presenter". The XML specification of the task model of a "Chair" will be discussed first.

```
<?xml version="1.0" encoding="UTF-8"?>
<taskmodel name="Chairing a session" role="Chair">
  <task name="Chair session">
    <task name="Introduce session"
      operator="enabling">
    </task>
    <task name="Introduce presenter"
      operator="disabling"
      iterative="true">
    </task>
    <task name="Open discussion"
      operator="enabling"
      precondition="Presenter.allInstances.EndTalk">
    </task>
    <task name="Close session"/>
  </task>
</taskmodel>
```

The above specification uses a precondition that has a similar notation to that of triggers. In preconditions, the completed execution of the mentioned task is assumed. Therefore, the information ".start" or ".end" at the end of the expression is missing.

According to the model, a chair introduces a session and several presenters afterwards. After all presentations, the discussion is opened and finally the chair closes a session. A precondition for opening the discussion is the fact that all presenters have finished their talks. Fig. 6 visualizes the corresponding task hierarchy and presents the start state of the task instance.

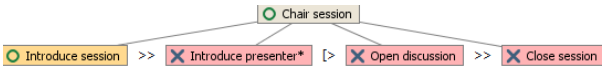

Figure 6. Task model instance of a chair.

In CoTaSL these activities can be specified as follows:

```
role Chair for Chair {
  task Chair = Introduce_session >> Introduce_presenter{*} >> Open_discussion >>
               Close_session;
    pre Close_session -> (Presenter.allInstances.End_talk );
}
```

Let us assume that a presenter introduces his or her talk first and afterwards, a series of slides is explained before the talk is ended. A slide is explained by first activating it, taking an optional nip of water afterwards and finally explain the slide. Fig. 7 provides a corresponding animated task model instance.

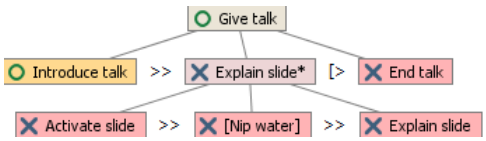

Figure 7. Task model instance of a presenter after instantiation.

Below you will find the CoTaL specification of the task model of the role "Presenter". A talk can be introduced after a chair introduced a new presenter.

```
<?xml version="1.0" encoding="UTF-8"?>
<taskmodel name="Presenting a talk" role="Presenter">
  <task name="Give talk">
    <task name="Introduce talk"
      operator="enabling"
      precondition=
        "Chair.oneInstance.IntroducePresenter">
    </task>
    <task name="Explain slide"
      operator="disabling" iterative="true">
      <task name="Activate slide" operator="enabling"/>
      <task name="Nip water" operator="enabling"
         optional="true">
      </task>
      <task name="Explain slide"/>>
    </task>
    <task name="End talk"/>
  </task>
</taskmodel>
```

There seems to be no further discussions of the role model of a presenter necessary. No new language features were used.

In the following example we will have three presenters Paul, Peter, and Penny and two chairs Charles and Chris. The following Fig. 8 presents the visualization of the state of the simulated team and role model instances of chair Charles.

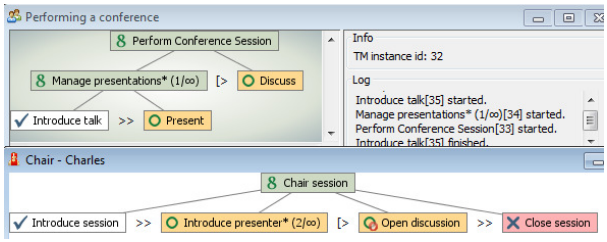

Figure 8. Model instances after scenario "Introduce session" and "Introduce presenter".

One can see the thrown events in the log information of each model instance. In Fig. 8 only the team model logs are presented because of lack of space. For Charles the task "introduce presenter" is already executed once and now the second activation waits. From the team model instance one can see that yet no presentation was given. However, a presentation could start right now. After Paul performed his talk with four slides (the fifth could have been presented), the state of the model instances presented in Fig. 9 is reached. The team model instance and the role model instance of Charles reflect the fact that the second talk can be introduced.

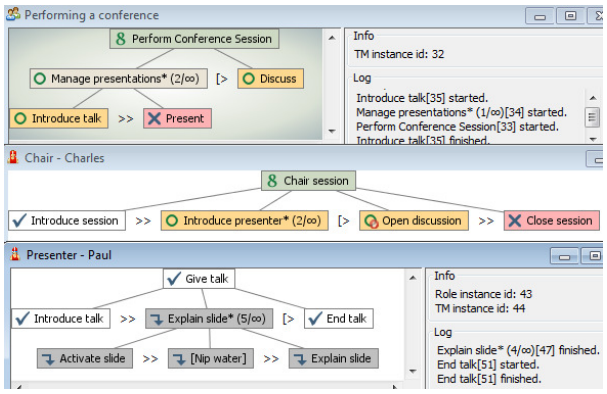

Figure 9. Model instances after the first talk

According to the temporal relations of the task model chair Charles can open the discussion. However, there is a constraint that is still not fulfilled. The constraint is the precondition "`Presenter.allInstances.endTalk`", which means that the discussion cannot be started until all presenters finished their talks. The red overlay icon in Charles's task "Open discussion" symbolizes this. We now assume that all presenters finished their talks and chair Charles can open the discussion. This situation is reflected by Fig. 10. The task models of the presenters are omitted because they all look like that of presenter Paul in Fig. 9.

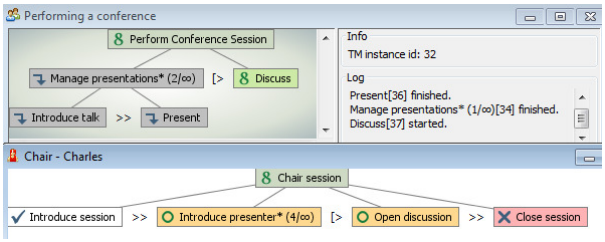

Figure 10. Model instances after all talks.

Charles can open the discussion at this stage. No new presenters can be introduced because all presenters were active already. "Introduce presenter" is in the fourth iteration. Fig. 11 presents the final states for the model instance of the conference and the chair.

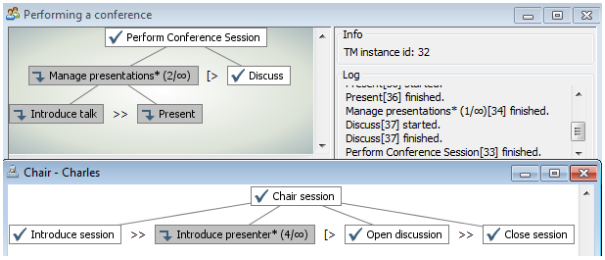

Figure 11. Final model instances

The scenario of the conference session went well. However, the specified constraints were very general. One instance of a presenter had to start a talk and one instance had to end it. Indeed, it is not checked whether this is the same instance. The following situation would be possible.

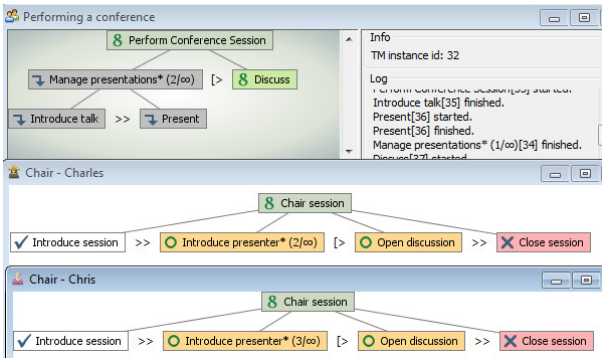

Figure 12. Model instances after three talks

In the example of Fig. 12 Charles and Chris introduced a session. Additionally, Charles introduced one presenter and Chris introduced two. Both can open the discussion now. However, this is not the way a session should be performed. The chair who opened a session should also close it. Additionally, there should not be two openings. To allow a more precise specification the concept of context and related binding of variables is used.

Context and Binding Variables

Sometimes it is necessary to specify certain constraints. A session should e.g. be closed by those chair that opened it. The constraint could be specified by OCL [15] expressions. However, their readability is limited. Based on the experience with CTML [23], a very reduced version of OCL like expressions is used. Let us have a look at the extended example of the team model.

```
team for t {
  contextVar Session { roleVar (Chair) C;};
  contextVar Talk { roleVar (Presenter) P;};
  task Perform_Conference_Session = Introduce_session >>
            Manage_presentation{*} [> Discuss;
  contextBind Perform_Conference_Session -> Session;
    task Introduce_session;
    roleVarBind introduce_session ->  C.introduce_session;
    task Manage_presentation = Introduce_talk >> Present;
    contextBind manage_presentations -> Talk;
      task Introduce_talk;
      roleVarBind Introduce_talk -> C.Introduce_presenter;
      task Present;
      roleVarBind present_talk -> P.give_talk;
    task Discuss;
   roleVarBind open_discussion -> C.open_discussion;
}
```

Contexts can be defined for certain tasks in the team model:

```
contextBind manage_conference -> Session;
contextBind manage_presentations -> Talk;
```

Linked to such contexts it is possible to introduce variables for roles:

```
contextVar Session { roleVar (Chair) C; }
contextVar Talk { roleVar (Presenter) P; };
```

The value of such a variable can be bound. The binding of values within such a context takes place by a certain task execution of an instance of the specified role:

```
roleVarBind introduce_session->C.introduce_session;
```

This would mean that the instance of any role model that executes first the task "`Introduce Session`" is stored into the variable "`C`". Later use of this variable will refer to the stored instance.

By specifying contexts, activities can be bound to a certain subject (instance of a role). In the above example, the chair that introduces a session, introduces the presenter and closes the session has always to be the same person (role instance). Additionally, within the example specification presenters are bound to a talk. However, there is a need for more dynamic bindings. Otherwise, the first presenter had to give all talks. Within the iteration every time the presenter that first started his talk is bound to the corresponding variable. This is possible by a specific attribute (`bindAllIterations->"false"`). The default value of this attribute is "`true`" and means that the first bound values are the same for all iterations.

With this improved specification with an additional team task ("`Open session`") and the specified constraints, the situation presented in Fig. 10 looks now like shown in Fig. 13. The chair Charles cannot introduce a new presenter after one presenter was introduced. In the old version of the specification, the chair was able to introduce the next talks while a presenter was still showing slides. This is no longer possible with the new extended specification. For both versions of the specification, it is possible within the CoTaSE environment to create new instances of role models. In this way, a new presenter can be created. This reflects the fact that a new subject enters the room that has the corresponding role. This instance creation, of course, influences the states of the model of chair Charles and the model of the team.

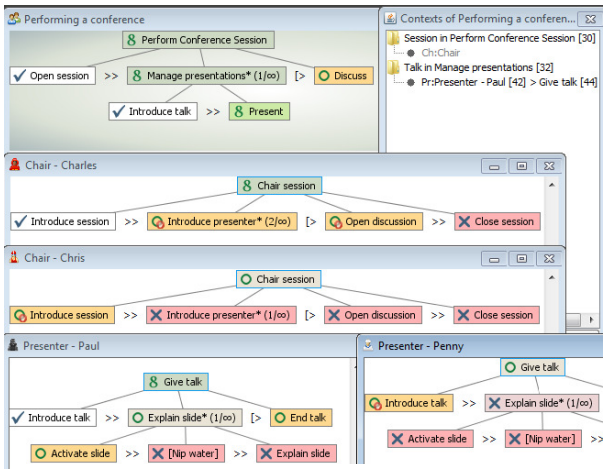

Figure 13. State after start of the first presentation with context binding

In the upper right corner, one can see that Paul is the current presenter that is stored in `Pr`. Charles is the chair that started the session. Neither Charles nor Chris are at this stage allowed to introduce a new talk. Chris has to wait until the whole session is over and Charles has to wait until the current presentation is finished.
However, in certain circumstances it makes sense to distinguish between iteration and instance iteration. Instance iteration allows the beginning of the next iteration before the preceding iteration ended.

The next paragraph will present some details of instance creation and instance iteration.

Dynamic Instances and Instance Iteration

The environment CoTaSE allows the creation of task model instances during runtime, in our case simulation time. Fig. 14 shows the simulation state after the presentations of Paul and Penny (all presenters). Fig. 15 presents the states of the model instances after creating the presenter Peter. Chair Chris cannot open the discussion anymore because there is the constraint that all presenters ended their talks. However, he can introduce a presenter. This was not possible without the new presenter. The team model changes as well. The management of presentations becomes active again and "Introduce talks" is not finished anymore.

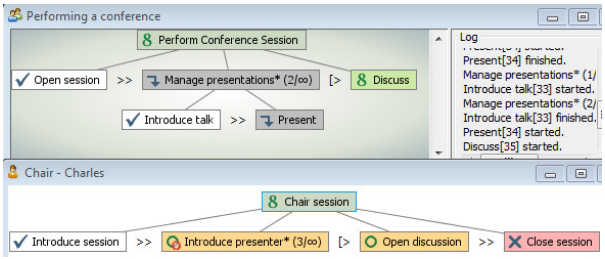

Figure 14. Situation after all two presentations

Experimenting with task models resulted in some further needs related to instances. From our point of view it is necessary to provide support for instance iterations. Sometimes it makes sense to start with the next iteration before the first one is finished. For the domain of conference session it might be possible that a chair manages several online presentations in parallel. In this case the second presentation can start before the first one finishes. It is assumed that at most four presentations can run in parallel. Fig. 16 presents the situation after the first presenter was introduced and started and the second one was introduced. Instance iterations (marked with a "#" followed by up to two integers specifying optional minimum and maximum instance numbers) are used in the team model and in the model of the chair. Instances of the tasks within the iterations are executed in parallel. The interleaving operator (|||) is used to visualize this fact.

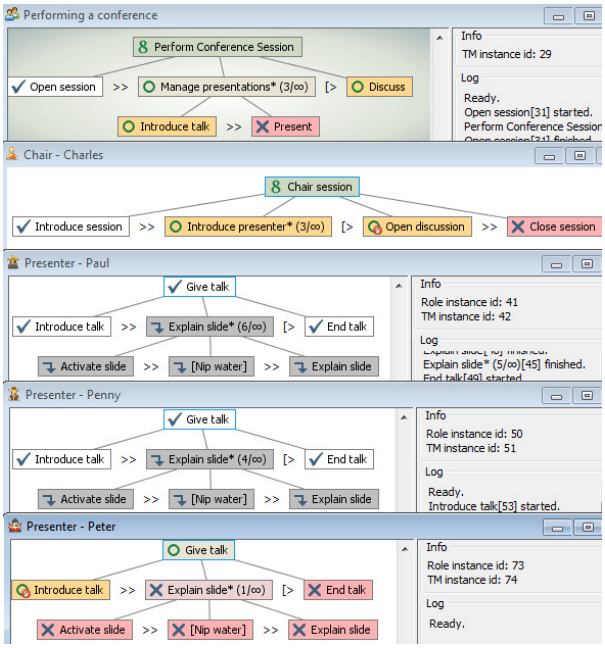

Figure 15. Situation after creating presenter Peter in the state provided in Fig. 14.

One can see in the team model of Fig. 16 that two presentations were introduced. The same fact can be concluded for the model of the chair Phil. He is able to introduce the third presentation. While Paula already introduced her talk and explains slides, Jan is able to introduce his talk. The third presenter Peter is not included in Fig. 16 because his state is identical to that of Fig. 15. He cannot introduce his talk because the chair did not introduce the third talk of the conference session.

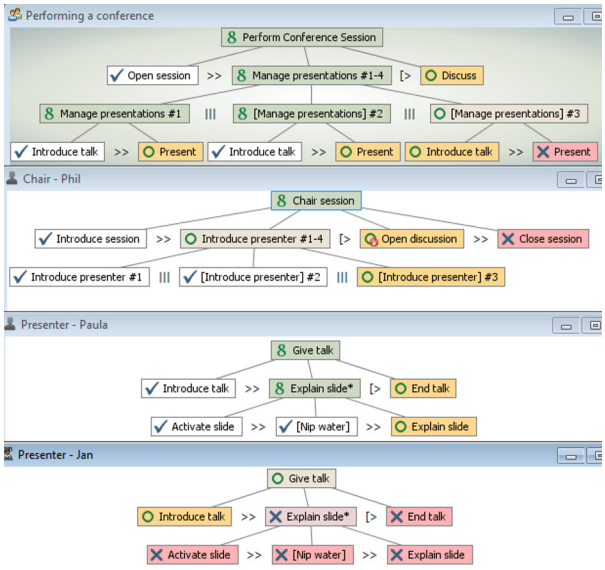

Figure 16. Task model instances with instance iteration.

SUMMARY

It was suggested in this paper that models for describing activities of users help to integrate HCD into agile software development. A process model was presented that suggests HCD activities in one sprint ahead to the development.

The language CoTaSL was introduced as a domain specific language for task models. The language allows very precise specifications, allows context bindings and instance iterations. Additionally, CoTaSL allows abstract and human-centered specifications.

In [10] one can see how the principles of such a language (it was CoTaL there) allow subject-oriented business process specifications. A kind of workflow system based on this concept is presented in [3]. It allows distributed collaborative simulation of activities over the Internet.

Currently, tool support is provided for CoTaSL by a syntax-driven editor and a generator to CoTaL. The generated code can be simulated within the environment CoTaSE. However, the language can be transformed into a CTT specification or the notation of HAMSTERS.

For the future, the inclusion of more features into the language might be useful. In this way, it can become a general programming language for task models. It might even be possible to find another even more abstract domain specific language that can be translated to CoTaSL.

ACKNOWLEDGMENTS

The work was supported by DFG graduate school 1424 Multimodal Smart Appliance Ensembles for Mobile Applications (MuSAMA) at the University of Rostock, Germany.

REFERENCES

1. Agile manifest: http://agilemanifesto.org/iso/en/principles.html, last visited, July 5th 2017.
2. Anett, J.: Hierarchical Task Analysis. In Diaper D., Stanton N. (Eds), The Handbook of Task Analysis for Human-Computer Interaction (pp. 67-82). Lawrence Erlbaum Associates, 2004.
3. Buchholz, G., and Forbrig, P: Extended Features of Task Models for Specifying Cooperative Activities, accepted for EICS 2017, Lisbon.
4. BPMN: http://www.bpmn.org/ last visited, July 5th 2017.
5. CTTE: http://giove.cnuce.cnr.it/ctte.html, last visited January 6th 2017.
6. Fleischmann, A., Schmidt, W., Stary, C., Obermeier, S., and Börger, E., 2012: Subject-Oriented Business Process Management. Springer, ISBN 978-3-642-32391-1, pp. I-XV, 1-375
7. Forbrig, P.: Interactions in Smart Environments and the Importance of Modelling, Proceedings of the National Conference on Human-Computer-Interaction - Romanian Journal of HCI, 2012, http://rochi.utcluj.ro/rrioc/articole/RoCHI-2012/RoCHI-2012-Forbrig.pdf
8. Forbrig, P., and Herczeg, M.: Managing the Agile Process of Human-Centred Design and Software Development, IFIP WG 13.2 Workshop proceedings: User Experience and User-Centered Development Processes. INTERACT 2015. Bamberg.
9. Forbrig, P: When Do Projects End? - The Role of Continuous Software Engineering. Prc. BIR 2016: 107-121
10. Forbrig, P., and Buchholz, G.: Subject-Oriented Specification of Smart Environments, Proc. 9th Conference on Subject-oriented Business Process Management, S-BPM ONE 2017, Darmstadt, Germany, March 30-31, 2017. ACM 2017, ISBN 978-1-4503-4862-1, http://dl.acm.org/citation.cfm?id=3040570
11. HAMSTERS: https://www.irit.fr/recherches/ICS/softwares/hamsters/, last visited January 6th 2017.
12. HCD: https://www.iso.org/standard/52075.html, last visited May 20th, 2017
13. Johnson, P., Johnson, H., Waddington, R., and Shouls, A.: Task-Related Knowledge Structures: Analysis, Modelling and Application. BCS HCI 1988: 35-62.
14. Martinie, C., Barboni, E., Navarre, D., and Palanque, P. A., Fahssi, R., Poupart, E., and Cubero-Castan, E.: Multi-models-based engineering of collaborative systems: application to collision avoidance operations for spacecraft. EICS 2014: 85-94
15. OCL: http://www.omg.org/spec/OCL/, last visited January 6th 2017.
16. Paelke, V. and Nebe, K. Integrating Agile Methods for Mixed Reality Design Space Exploration. In Proceedings of the 7th ACM conference on Designing interactive systems (DIS '08). ACM, New York, NY, USA, 240-249. http://doi.acm.org/10.1145/1394445.1394471
17. Paternò, F.: Model-Based Design and Evaluation of Interactive Application, Springer Verlag, ISBN 1-85233-155-0.
18. Penichet, V. M. R., Lozano, M. D., Gallud, J. A., and Tesoriero, R.: Task Modelling for Collaborative Systems. TAMODIA 2007: 287-292
19. SCRUM: https://en.wikipedia.org/wiki/Scrum_(software_development), last visited May 20th 2017.
20. Solano, A., Toni Granollers, T., Collazos, C. A., and Rusu, C.: Proposing Formal Notation for Modeling Collaborative Processes Extending HAMSTERS Notation. WorldCIST (1) 2014: 257-266
21. Sy, D.: Adapting usability investigations for agile user-centered design. J. Usability Stud. 2 (3), pp. 112-132, 2007.
22. UML: http://www.uml.org/, last visited January 6th 2017.
23. Wurdel, M., Sinnig, D. and Forbrig, P. 2008. CTML: Domain and Task Modeling for Collaborative Environments. In: *Journal of Universal Computer Science 14(19) (Special Issue on Human-Computer Interaction)*, 3188-3201.
24. Xtend: http://www.eclipse.org/xtend/, last visited May 20th, 2017.
25. Xtext: https://eclipse.org/Xtext/documentation/, last visited May 20th, 2017.