Focusing on Two Different Aspects of Collaboration in Software Engineering: Supporting Collaborative Modeling and Specifications of Collaborative Activities

Peter Forbrig, Anke Dittmar, Gregor Buchholz, and Mathias Kühn

University of Rostock, Rostock, Germany
{peter.forbrig,anke.dittmar,gregor.buchholz,mathias.kuehn}@uni-rostock.de

Abstract. The paper discusses two aspect of collaboration in software engineering. The first one focusses on methodological support for collaborative modelling and the second one on the specification of collaborative activities and their simulation. For both aspects links to the literature are provided and existing tool support is discussed. The language CoTaL and their simulation environment CoTaSE are shortly introduced.

Keywords: collaborative modelling · collaborative activities specification

1 Introduction

Professional software development is most of the time a collaborative activity. Only in very rare situations software is developed by one person only. There are several studies on collaborative programming like [8] and [9]. Rittgen [11] focusses on collaborative modelling that is less studied. Renger et.al. [10] provide a literature review about collaborative modeling. With new technologies like table tops the question arises how collaborative modeling can be supported.

2 Collaborative Modelling with Class Diagrams

This section describes a study [5] that was conducted on supporting collaborative software design sessions. Several groups of software designers were observed while performing certain design tasks. The study included an initial analysis of manual collaborative modelling sessions each with 3 participants (Fig. 1). Paper was used to represent classes and associations. Based on this analysis, a software prototype for interactive table tops was developed that support teams up to 3 designers in modeling with class diagrams (Fig. 2). The software considers different spaces on its graphical interface. Based on the analysis, these spaces can be grouped into personal spaces for each designer and one group space for all designers.

Different class diagram designs can be reflected at runtime to compare alternatives for certain design solutions. Ongoing investigation is made on improving design processes by supporting teams with different strategies. One strategy can be to guide sessions for structuring processes at all. However, tool support for multiple designers of collaborative teams differs from tool support for single designers.

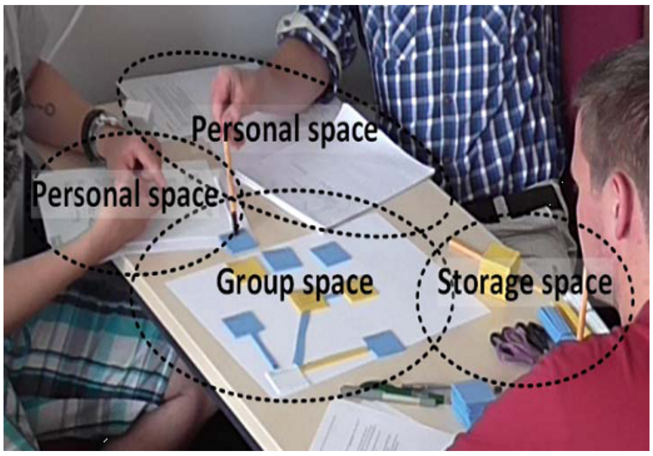


Fig. 1. Analysis of collaborative work with manual support.

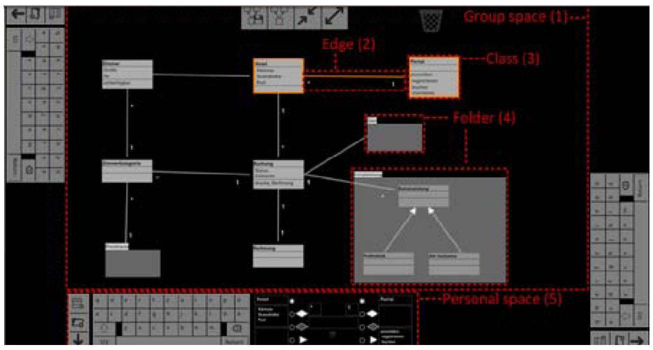


Fig. 2. Class diagram editor for three developers.

Software that can be used for collaborative design sessions must satisfy needs of persons and groups as well. The analysis (Fig. 1) has shown that software requirements exist not only for collaborative but also for individual support.

This results in a user interface that is shown in Fig. 2. Individual designers have their own personal spaces for editing classes, relations, and notes of models.

All personal spaces are equipped with toolbars and software keyboards that support making edits. The group space shows designed class diagrams and allows designers to adapt layouts. Classes, relations, and notes of diagrams can be blocked by designers that select them in the group space. This strategy helps to avoid conflicts when editing elements.

Individual support for designers is integrated to the user interface of this software prototype. Personal spaces could also be provided on different devices what is part of other studies. However, the problem of merging models that come from more than one participating designer arises when targeting the functionality of how to deal with conflicting model elements. The same problem arises when designers try to merge different forks of alternative models designs. Solutions for this problem are parts of ongoing studies. However, the process of designing models with class diagrams also is affected by the expertise of participating designers.

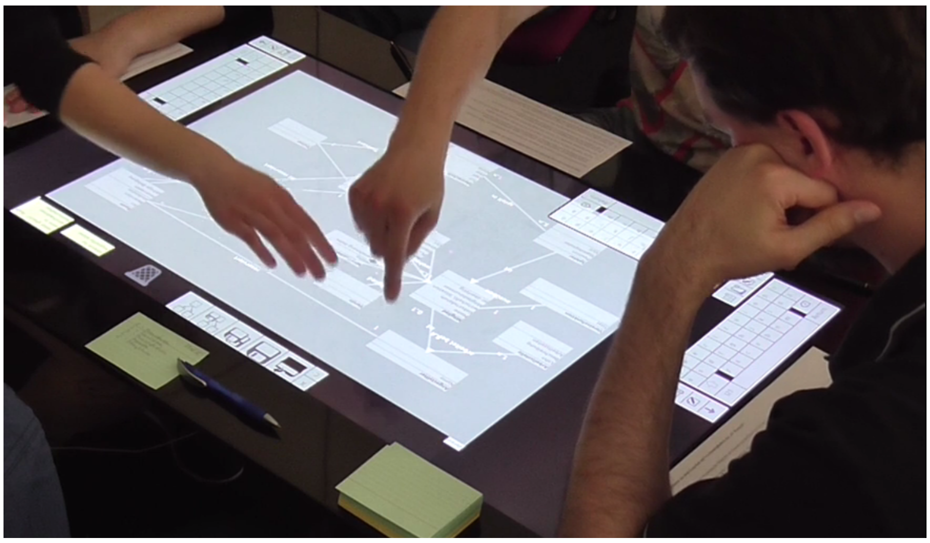


Fig. 3. Facilitated design session with the software prototype.

Design sessions can be structured in different ways. Most of the time designers are on their own and the problem can arise that the design process is getting unstructured. Moderators can facilitate design sessions in a way that they guide the process to open up unknown domains in a more structured way. However, also moderators must have a certain expertise to help participants in designing domain-specific models. Another solution can be to make rules for designers on how to behave in collaborative design sessions. These rules need to be identified by comparing the results coming from structured and unstructured sessions, what is part of ongoing studies.

3 Specifying Collaborative Activities

The software under development often has to support collaborative activities as well. Therefore, it is necessary to have precise specifications for these activities. Some of our studies recognized that current task models and business process specifications [2] are not precise enough [1].

3.1 A Language for Specifying Cooperative Activities

Based on the concept of task models [4], a language CoTaL [1] was developed to allow dynamic variable binding for variables. In this way context dependencies can be specified in an easy way. It is e.g. possible to specify that the doctor who diagnosed a patient also has to perform the treatment.

Besides the language CoTaL an environment CoTaSE [3] was developed that allows the animation of the CoTaL specifications. It can be characterized as follows:

- Hierarchical description of a team model and several role models.
- Temporal relations are support. It includes instance iteration.
- Models can be simulated.
- Each role model can be instantiated several times.
- Instantiation can be performed during runtime. (simulation time)
- Triggers, preconditions and contexts can be specified.

Specifications can be in depth validated before implementing the software. An overview of the language and its application to smart environments is provided in [1]. To provide an example we picked one from the literature. It is taken from X and based on the following natural language description.

"An employee fills in a holiday application form. He/She puts in a start and end date of his/her vacation. The responsible manager checks the application and informs the employee about his/her decision; the holiday request might be rejected or get approved. In case of approval the holiday data are sent to the human resource department which updates the days-off in the holiday file." ([6], p. 220)

From the text one can identify three roles: Employee, Manager, and Human Resources (HR). A task Process request is identified for the manager. It has to be of type instance iterative. Otherwise, a manger would only be able to accept one request at any time. According to the provided specification below, there has to be at least one instance iteration. Additionally it is specified that there is no maximal number of requests (-1 represents the infinite symbol).

Three variables E1, M1 and H1 are declared and bound to the context C1. In this way, the instance of the employee that requests a vacation (bindVar-Task = "E1.Ask") is stored in variable E1. This has the consequence, that the task “GoOnVacation” is restricted to the same Person by “bindVarTask = E1.GoOnVacation”. These details are often neglected in business process specifications. They often do not specify such details. Some comparisons of business process specification examples are provided in [7].

It seems to be obvious, that such details of specifications are very important for the implementation of corresponding software systems. They should be analyzed and clarified during the requirements analysis activities.

```
<?xml version="1.0" encoding="UTF-8"?>
<taskmodel name="Handling vacation request" role="Team">
  <roles>
    <role name="Employee" file="Employee.xml" />
    <role name="Manager" file="Manager.xml" />
    <role name="HR" file="HR.xml" />
  </roles>
  <task name="Process request" bindingContext="C1"
     instIterative="true" instmin="1" instmax="4">
    <roleVar role="Employee" var="E1"/>
    <roleVar role="Manager" var="M1"/>
    <roleVar role="HR" var="H1"/>
    <task name="Create request" operator="enabling"
      bindVarTask="E1.Ask">
    </task>
    <task name="Request accepted" operator="enabling"
      bindVarTask="M1.ManageRequest" />
    <task name="Handle decision">
      <task name="Option Yes" operator="choice">
        <task name="Decide Yes" operator="enabling"
          bindVarTask="M1.AcceptRequest" />
        <task name="Inform" operator="enabling"
          bindVarTask="M1.CongratulateEmployee" />
        <task name="Handle Yes" >
          <task name="Do something"
            operator="interleaving"
            bindVarTask="E1.GoOnVacation" />
          <task name="Document Yes"
            bindVarTask="H1.Archive" />
        </task>
      </task>
      <task name="Option No">
        <task name="Decide No" operator="enabling"
          bindVarTask="M1.TurnDownRequest" />
        <task name="Inform" operator="enabling"
          bindVarTask="M1.RejectEmployee" />
        <task name="Handle No">
          <task name="Work as usual"
            bindVarTask="E1.GoToWork" />
        </task>
      </task>
    </task>
  </task>
</taskmodel>
```

The initially animated models of the vacation request example are provided by Fig. 4 and Fig. 5. Green circles signal that the corresponding task can be executed. If a read circle is attached to the green one this means that in principle (according to the own temporal relations) the task of the model instance could be executed. However, certain constraints are not fulfilled. For Fig. 4 human resources has to announce first that holidays are approaching. Employees can ask for vacation afterwards and managers can process requests after they were formulated.

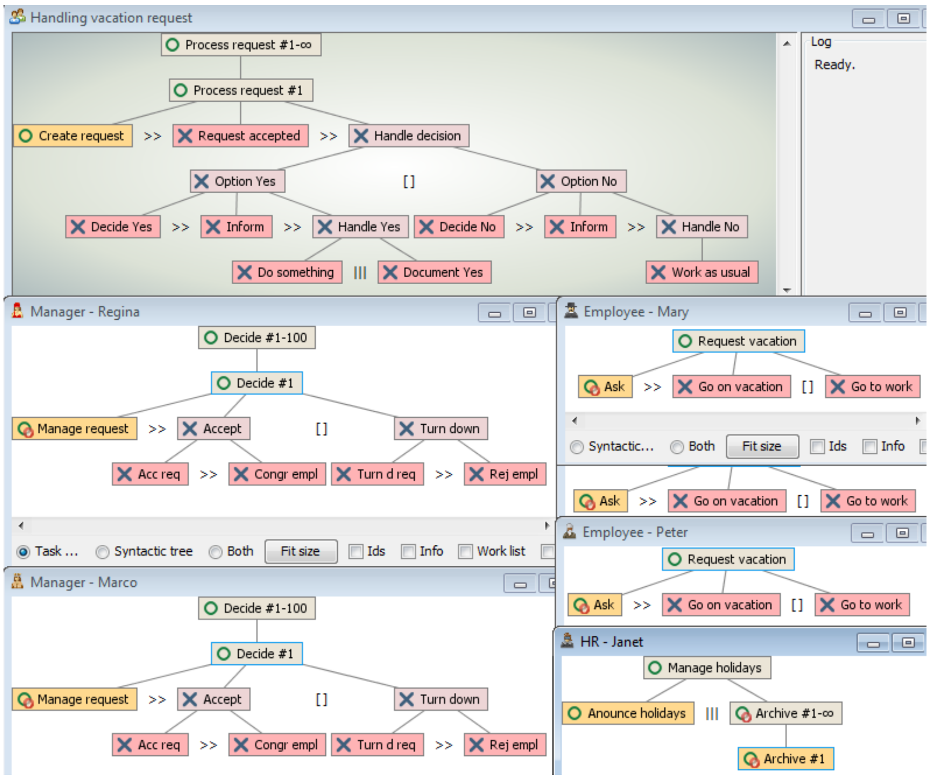
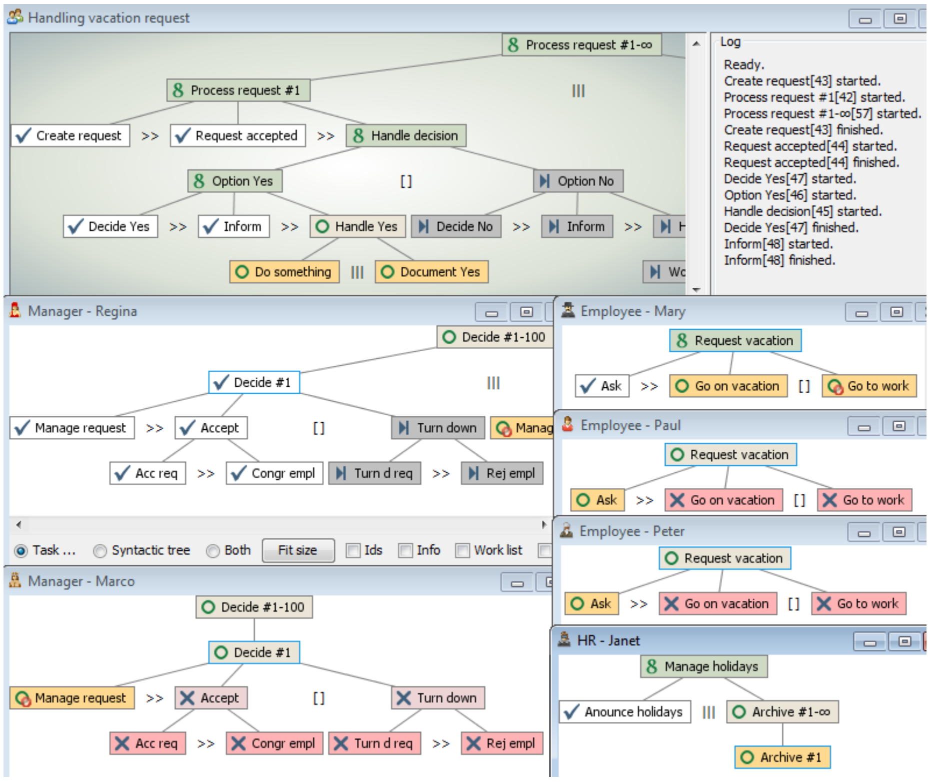


Fig. 4. Initial task model instances of the holiday request example.

In the upper part of Fig. 4 on can the see the model that describes the collaboration. We call it team model. It exists only once. All other models are role models and different instance can exist. In the example above there are two managers Regina and Marco. Additionally, there are two employees Mary and Peter. For human resources there is only on instance that is called Janet.

The simulation environment CoTaSE allows the dynamic creation of further role model instances at any time. In this way constraints can change.

Let us create a further role instance for employee that is called Paul. We will have a look at the model instances after the first vacation request was created.



```
Log
Ready.
Create request[43] started.
Process request #1[42] started.
Process request #1-∞[57] started.
Create request[43] finished.
Request accepted[44] started.
Request accepted[44] finished.
Decide Yes[47] started.
Option Yes[46] started.
Handle decision[45] started.
Decide Yes[47] finished.
Inform[48] started.
Inform[48] finished.
```

Fig. 5. A task model instance after first request was accepted.

Already from the team model one can imagine that in parallel to request one there might be the next one processed in parallel. One can also see that manager Regina made her first decision. She accepted a request. The request came from Mary who can go on vacation now. Regine could manage the second request. However, there is no further request available. A request from Paul or Peter could be processed by Regina or Marco.

3.2 Summary of Specifying Collaborative Activities

While analyzing existing language and tools for task models a lack of support for specifying cooperative work was identified in the domain of smart meeting rooms. A new specification language CoTaL was introduced that extends features of task models in such a way that more precise specifications are possible.

Each role is specified by a task model. The communication between roles is characterized by a specific task model that is called team model. It reflects the activities of all actors and exists only once. Each actor is associated with one or more role model instances.

The specification of task models specifying cooperative activities was extended by context definitions. Within such task context, variables can be defined. They allow the storing of references to instances of roles. A language CoTaL is provided for this purpose. This language and its interpreting environment CoTaSE were discussed.

The environment CoTaSE allows the dynamic instantiation of task models during runtime which is an important feature for the application of task model specifications for smart environments and cooperative work in general. It implements the temporal operator of instance iteration. Additionally, it provides the feature of ending and starting tasks (in contrast to executing tasks as an atomic action, i.e. a task execution that takes place at a specific point of time instead of a time span $\Delta t = t_{end} - t_{start}$) which is important for simulating parallel activities. Tasks can have preconditions, start triggers, and end triggers. Scenarios can be created that are useful for testing the final application or for other duties. It is for instance possible to generate a series of scenarios that are used to construct new versions of task models.

4 Summary

Two different aspects of collaboration in software engineering were discussed. The first one focusses on collaborative modelling, which is not very well understood yet. More precise methodological support seems to be necessary in the future. This has to go together with corresponding tool support. Additionally, specifications were discussed that allow precise descriptions of collaborative work. Also for this aspect more conceptual work and tool support are necessary. It would be a challenge for the future to describe the activities in collaborative modelling by corresponding task models.

References

1. Buchholz, G., and Forbrig, P.: Extended Features of Task Models for Specifying Cooperative Activities. Proc. ACM Hum.-Comput. Interact. 1, 1, Article 7 (June 2017), pp. 1-21. doi: 10.1145/3095809
2. BPMN: http://www.bpmn.org/, last visited July 31 2017.
3. CoTaSE: http://www.cotase.info/, last visited July 31 2017.
4. CTTE: http://giove.cnuce.cnr.it/ctte.html, last visited July 31 2017.
5. Dittmar, A., Buchholz, G., and Kühn, M.: Effects of Facilitation on Collaborative Modeling Sessions with a Multi-Touch UML Editor. In: Proceedings of the ICSE-SEET 2017, pp. 97-106. doi: 10.1109/ICSE-SEET.2017.14

6. Fleischmann, A. and Stary, C.: Whom to talk to? A stakeholder perspective on business process development. Universal Access in the Information Society, 11(2), pp. 125-150, http://link.springer.com/article/10.1007/s10209-011-0236-x
7. 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
8. McDowell, C., Werner, L., Bullock, H., and Fernald, J.: The effects of pair-programming on performance in an introductory programming course. SIGCSE Bull. 34, 1 (February 2002), pp. 38-42. doi: 10.1145/563517.563353
9. Nosek, J. T.: The case for collaborative programming. Commun. ACM 41, 3 (March 1998), pp. 105-108. doi: 10.1145/272287.272333
10. Renger, M., Kolfschoten, G. L., and de Vreede, G.-J.: Challenges in collaborative modelling: a literature review and research agenda, M. Renger, G.L. Kolfschoten, G.-J. de Vreede, Challenges in Collaborative Modeling: A Literature Review, CIAO! 2008 and EOMAS 2008, LNBIP 10, 2008, pp. 6177.
11. Rittgen, P.: Collaborative Modeling: Roles, Activities and Team Organization. Int. J. Inf. Syst. Model. Des. 1, 3 (July 2010), pp. 1-19. doi: 10.4018/jismd.2010070101