BUILDING UML MODELS COLLABORATIVELY

Weiqin Chen, Roger Pedersen and Øystein Pettersen
Department of Information Science and Media Studies, University of Bergen, Fosswinckelsgt. 6, Bergen, Norway

Keywords: Computer Supported Collaborative Work (CSCW), Software Agents, UML modelling.

Abstract: This paper presents the design, implementation and evaluation of a distributed collaborative UML modelling tool. The tool is designed as a platform for building UML models in collaborative design of software. We conducted formative evaluations and summative evaluation to improve the environment and discover how users collaborate through the environment. The findings indicate that this environment is both user-friendly and informative. We also found that less experienced users can learn from their more experienced collaborators.

1 INTRODUCTION

Modelling is an important activity in helping people understand better a complex domain of the world. Models use a set of rules and concepts to visualize and explain complex relationships within a given domain. Over the last few decades, several modelling languages have been created, and many of these have been computerized. Examples of computerized modelling languages are the Entity-relationship (ER) modelling language and the Unified Modelling Language (UML) (Jacobson et al., 1999). UML is a language for specifying, visualizing, and constructing the artefacts of software systems. The language has gained enough support that it is now the industry-standard language for visualizing and constructing software. UML supports several different modelling diagrams such as use case diagrams, class diagrams, object diagrams, sequence diagrams, collaboration diagrams, statechart diagrams, activity diagrams, component diagrams, and deployment diagrams. With the development of open source community, the requirement for supporting distributed modelling and programming is increasing rapidly.

Normally a diagram is created on a computer, on a piece of paper, or on a whiteboard. When it has been created on a computer there is usually only one user who can manipulate the diagram at a time. A problem occurs if a group wants to create a class diagram collaboratively. Participants have to create a draft, email these drafts to one another and comment on each other's work. This is a time-consuming process. ArgoUML (http://argouml.tigris.org/), Unimodeler (http://www.unimodeler.com/), and other commercial UML modeling tools support most of the activities when creating UML models on a piece of paper. However, it does not allow for distributed collaboration among multiple users. In order to support distributed collaborative modelling, some environments have been developed. For example, COLER (Constantino-Gonzalez and Suthers, 2000) is a web-based collaborative ER modelling environment. Cool Modes (Pinkwart, 2003) is a framework supporting collaborative modelling such as Petri Nets and System Dynamic models. However, to our knowledge, an environment supporting distributed collaborative UML modelling does not exist.

In the project presented in this paper our main concern is to study how collaboration technology facilitates distributed collaborative UML modelling and how this technology can best be developed. More specifically, we have tried to shed some light on the following research questions:

- How should a distributed environment be developed to facilitate collaborative UML modelling?
- How does such a tool support users' interactions?

In order to address these research questions, we first developed a prototype based on our understanding of collaborative UML modelling activities when using whiteboard or paper. Then an experiment was conducted in order to study how users use this environment to build UML models collaboratively and to receive feedback on the design of the environment. Based on the findings from the experiment, we improved the prototype and conducted another experiment. After three iterations of improvement on the prototype, we conducted a final summative evaluation to review the quality of the environment.

This paper is organized as follows. We begin by introducing some specific research areas into which the work presented in this paper falls. This is followed by a detailed description of our approach, including the design, development and evaluation iterations. Finally, we discuss the answers to the research questions based on the evaluations and conclude the paper by identifying some future improvements.

2 RELATED RESEARCH AREAS

The general research areas of this paper are Computer Supported Collaborative Work (CSCW) and software agents.

2.1 CSCW and Groupware

A shared concern within the CSCW community is how technology influences working environments in both small groups and large organizations and how this technology can be best developed (Grudin, 1994). The technology in CSCW is often referred to as groupware. Groupware design involves understanding groups and how people behave in groups. It also involves having a good understanding of networking technology and how aspects of that technology (for instance, delays in synchronizing views) affect a user's experience.

Awareness is an important feature and technique to help users coordinate and manage collaboration. According to Dourish and Bellotti (Dourish and Bellotti, 1992), awareness is an understanding of the activities of others which provides a context for one’s own activity. This context is used to ensure that individual contributions are relevant to the group’s activity as a whole, and to evaluate individual actions with respect to group goals and progress. The information, then, allows groups to coordinate activities and manage the process of collaborative working (Gutwin et al., 1996).

The distributed collaborative UML modelling tool presented in this paper can be considered to be a groupware. In addition, awareness information is provided to facilitate the collaboration.

2.2 Software Agents

Nwana (Nwana, 1996) classified software agents according to three ideal and primary attributes which agents should exhibit: autonomy, cooperation and learning. Autonomy refers to the principle that agents can operate on their own without the need for human guidance. They “take initiative” instead of acting simply in response to their environment (Wooldridge and Jennings, 1998). Cooperation refers to the ability to interact with other agents and possibly humans via some communication language which means they should possess a social ability. Agent learning refers to agents’ capability of improving their performance over time.

Malone, Grant and Lai (Malone et al., 1997) review their experience in designing agents to support human working together (sharing information and coordination). From the experience, they found two design principles:

Semiformal systems: don’t build computational agents that try to solve complex problems all by themselves. Instead, build systems where the boundary between what the agents do and what the humans do is a flexible one.

Radical tailorability: don’t build agents that try to figure out for themselves things that humans could easily tell them. Instead, try to build systems that make it as easy as possible for humans to see and modify the same information and reasoning processes their agents are using.

There are two more concerns when software agents are built: competence and trust (Maes, 1997). Competence refers to how an agent acquires the knowledge it needs to decide when, what and how to perform the task. Trust refers to how we can guarantee that the user feels comfortable in following the advice of the agent, or delegating tasks to the agent. It is probably not a good idea to give a user an interface agent that is very sophisticated, qualified and autonomous from the start (Maes, 1997)(Maes 1997). That would leave the user with a feeling of loss of control and understanding.

The design of the agents in our project follows the principles identified by Malone, Grant and Lai. The agents provide advice and awareness information to the users and it is up to the users to take the advice or ignore them. The advice includes explanations in the text so that the users can understand why the advice is given.

3 A COLLABORATIVE UML MODELLING TOOL

In order to provide users with the operations required when creating UML class diagrams on a whiteboard or a piece of paper, we have identified the following functionality that should be provided by the environment. These requirements are based on functionality found in ArgoUML and COLER (a combination of ArgoUML's UML functionality and COLER's distributed features).

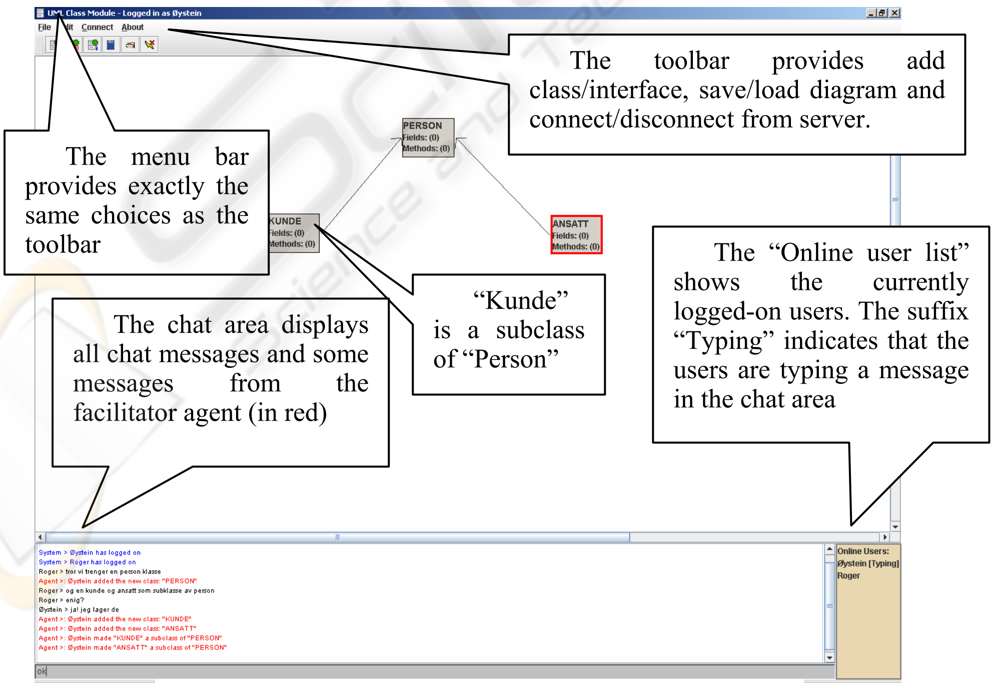
- Create UML objects: classes, abstract classes and interfaces.
- Create fields and methods in the above objects.
- Offer the possibility to extend/implement classes/interfaces.
- Offer the possibility to create an association between two classes.
- Delete classes, interfaces, methods, fields, associations and extensions.
- Rename classes, fields and methods.
- Move classes in the workspace.

Furthermore, because this environment is supposed to support distributed collaboration, users should be provided with a shared workspace where they can perform the operations and a chat client where they can discuss the corresponding activities in the shared workspace (Figure 1).

In addition, UML modelling language includes many rules. Considering the complexity of these rules, the users may not be aware of them when creating UML diagrams. Somehow the environment should provide help to the users concerning these rules. One approach could be to design the environment to block users from violating the rules. The disadvantage of this approach is that users will not be able to learn. Another approach could be to allow them to try and then provide related knowledge based on their actions. We believe that the users can learn from their mistakes and the explanations and advice provided by the environment.

In order to help users coordinate and achieve effective collaboration, the environment should also provide awareness information and advices to facilitate the collaboration. These should include:

- Monitoring the workspace including all activities by all participants.
- Encouraging idle users to participate.
- Encouraging active users to involve less active users in the discussion and the modelling.
- Encouraging inactive users to be more involved.
- Presenting awareness indicators.


UML Class Module - Logged in as Øystein
File
Edit
Connect
About
The toolbar provides add class/interface, save/load diagram and connect/disconnect from server.
The menu bar provides exactly the same choices as the toolbar
PERSON Fields: (0) Methods: (0)
KUNDE Fields: (0) Methods: (0)
ANSATT Fields: (0) Methods: (0)
"Kunde" is a subclass of "Person"
The "Online user list" shows the currently logged-on users. The suffix "Typing" indicates that the users are typing a message in the chat area
The chat area displays all chat messages and some messages from the facilitator agent (in red)
Online Users:
Øystein [Typing]
Roger

```
System > Øystein has logged on
System > Roger has logged on
Roger > tror vi trenger en person klasse
Agent >: Øystein added the new class: "PERSON"
Roger > og en kunde og ansatt som subklasse av person
Roger > enig?
Øystein > ja! jeg lager de
Agent >: Øystein added the new class: "KUNDE"
Agent >: Øystein added the new class: "ANSATT"
Agent >: Øystein made "KUNDE" a subclass of "PERSON"
Agent >: Øystein made "ANSATT" a subclass of "PERSON"
```

```
ok
```

Figure 1: Screenshot of shared workspace and chat area.

3.1 System Design and Development

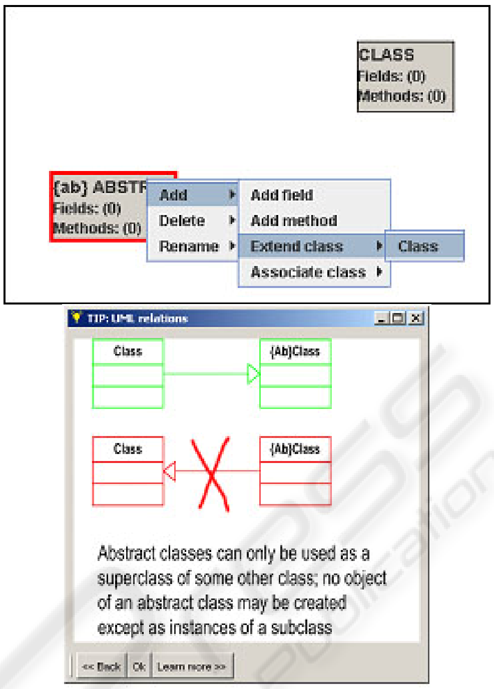
Whenever a user manipulates the workspace, the client sends the manipulation to the server. The server then sends the manipulation to all clients, who update the workspace accordingly. When the server receives an update, it also updates the activity log or the chat log. These logs are monitored by the agents. Both domain agent and facilitator agent have different rules which can be triggered by activities in the workspace and chat area.

The workspace support concurrent access and real-time interaction. All users can manipulate the workspace simultaneously. It is the server that controls the sequence of updates according to the timestamp that it receives each update. When one user manipulates an object (class, interface, associations, method etc) in the shared workspace, the other users will see a symbol showing who is manipulating this object when they move the mouse over this object. In this way they can coordinate their activities. When one user tries to delete one object, the agent will prompt them to discuss in the chat area and then vote to decide whether to delete this object or not.

Because all users are geographically distributed, the coordination of the collaboration depends on their self-regulation facilitated by the agents and discussions in the chat area.

3.2 Domain Agent

This agent is triggered when a user tries to conduct illegal operations. For example, when a user tries to make an abstract class a subclass of a regular class, the domain agent will pop up a window containing some explanations about why this is illegal and the correct way to do the operation. The pop-up window contains a few different pages. The sequence is decided by the operation the user tries to perform. Users can walk through all of these slides one by one, or close the window whenever they want. Figure 2 shows that a user has tried to make an abstract class a subclass of a regular class, and the domain agent has thus presented an explanation for why this is illegal.



Abstract classes can only be used as a superclass of some other class; no object of an abstract class may be created except as instances of a subclass

Figure 2: Domain agent showing an explanation when one user trying an illegal operation.

3.3 Facilitator Agent

In order to be able to monitor the users' collaboration, every action the users perform is logged and weighted. To ensure, for instance, that adding a class was more valuable than moving a class, we decided to use the following weighting:

Table 1: Weighting of actions.

Points	Action
6	Add new class/abstract class/interface
5	Add new link or association
4	Delete class/abstract class/interface
3	Delete link or association, add a new method or field
2	Delete method or field
1	Rename class/abstract class/interface and chat

The facilitator agent performs based on the weights of the different actions. It provides the following functionality according to user actions and mathematical formulas:

1. Notify all users when a new user logs on;
2. Notify all users when a new class/interface, association or extension is added, and the name of the user who added it;

3. Inform all users of the results regarding deletion;
4. Notify all users of who is currently typing a message in the chat area;
5. Advise more active participants to involve less active participants;
6. Encourage less active participants to discuss the task with other participants.

Functions 1-4 provide awareness indicators, and 5 and 6 have several variations of messages to the participants. For example, the facilitator agent checks the user's activity compared to the rest of the group and presents the following advice:

4 EVALUATION

In order to evaluate the environment and try to answer the research questions, we conducted three formative evaluations and a summative evaluation. In these evaluations, the participants worked in groups to solve different modeling tasks. Each task was presented to the participants as a scenario. For example one task was that the participants were to model a hotel information system for a group of hotels including booking, employee information and guest information. Each hotel would also have an overview of available rooms at other hotels. All participants were geographically distributed and they only used the environment to communicate with each other.

We use a combination of qualitative and quantitative data collected with the following methods:

Observations: During each evaluation we visited the participants to observe them using the system.

Logs: The server logged all actions performed by each participant and timestamped when the action was performed.

Questionnaire: After each task we handed out questionnaires to the participants asking for their opinions on some statements. The questionnaire included statements from five aspects: participant background with 3 questions (e.g. experience with UML), workspace with 12 questions (e.g. user-friendliness of workspace design), chat area with 5 questions (e.g. user-friendliness of chat area), agent with 15 questions (e.g. helpfulness/informativeness of agent advice, awareness indicators, activeness of agents, message presentation forms) and summary with 3 questions. For each statement the participants could comment "Totally agree", "Agree", "No opinion", "Disagree" or "Totally disagree". We also used open-ended questions to allow the participants to comment more thoroughly.

Interviews: Two participants in each group were interviewed in each evaluation to verify our observation and the answers from questionnaire.

4.1 Formative Evaluation

The goal of the formative evaluation was to detect possible problems and potential improvements. It was conducted three times with the same group of students who we considered to be expert users; they are master's students in information science who have taken course related to UML modeling. Each formative evaluation was carried out after some major improvements of the system. In each formative evaluation, the group spent approximately 40 minutes on the task and the participants were asked to answer the questionnaire afterward. The formative evaluations focused the software agents and the usability of the system. Improvements were made after each evaluation. The facilitator agents were "tuned" manually between each of the evaluations to fit the working pattern of the users better. The weighting of the actions and the time intervals for the agents' interventions were adjusted. The usability was also improved, including the graphical interface and new features based on feedback from the participants.

4.2 Summative Evaluation

The goal of the summative evaluation was to validate the results of the formative evaluations and review the quality of the environment. In this evaluation the participants were end users, a mix of experienced and novice users when it came to UML modeling. The participants were divided into three groups (two persons with "advanced" knowledge and two with basic knowledge in each group).

The summative evaluation confirmed the results from the formative evaluations. The most interesting difference found between the formative evaluations and the summative evaluation is in the action pattern. While the chat took 55% of the total number of actions in the formative evaluations, it took 38% in the summative evaluation. Half of the participants in the summative evaluation were novice users. From the observation and logs, we discovered that some novice users did not discuss the task with others before starting the modelling actions in the shared workspace. Their class diagram could be considered to be poorly designed in terms of the object-oriented principles. Only after the more experienced collaborators made some suggestions was the class diagram improved.

4.3 Summary

The results from the evaluations can be summarized as follows:

Usability: The participants were satisfied with the usability of the environment. 7 out of 8 participants thought that the workspace was both easy and intuitive to use (7 out of 8). The majority of the participants (7 out of 8) felt that the graphical user interface was well-designed. All 8 participants thought that the chat area was useful for the collaboration and it was an important supplement to the workspace.

The software agents: We received positive feedback on the agents. The participants found that the awareness indicators were especially important for their collaboration. The software agents were found to play a rather passive role in the collaboration (7 out of 8). The messages presented in the chat area were often ignored by participants because they were "busy with chatting or working on the workspace". Only the message presented in a popup dialog box drew attention because one had to click a button to get rid of it. This only happened when one tried to break the rule of UML or one tried to delete a component created by others. In these cases, the popup dialog box created a breakdown in the collaboration process and the participants had to stop what they were doing and pay attention to the message from the agents. The majority of the participants (6 out of 8) thought the breakdown was necessary.

Users with different knowledge levels regarding UML modelling in the same groups also expressed that working with more experienced users gave them a better understanding of UML modelling. For example, when a more experienced user wanted to delete a class, the environment demanded that the majority of the group members had to agree. This encouraged less experienced users to ask for explanations. Some less-experience users also reported that working distributively could prevent them from being overrun by the more experienced users and allow them to try and fail. Their mistakes were corrected by the more experienced users in the group and sometimes by the domain agent.

5 CONCLUSION

In this paper we have described the design, implementation and evaluation of a distributed collaborative UML modelling tool. The goal is to support distributed collaborative building of UML diagrams. In order to coordinate the collaboration, we have designed two types of agent: a facilitator agent and a domain agent.

Based on the evaluations, we have planned some future activities, for example, to study the improvement in models created by a group. This will give us some insights on how the use of this environment actually affects the quality of the models.

REFERENCES

Constantino-Gonzalez, M., and D. Suthers, 2000, A coached collaborative learning environment for Entity-Relationship modeling. Proceedings of the 5th International Conference on Intelligent Tutoring System: Montreal, Canada, p. 324-333.

Dourish, P., and V. Bellotti, 1992, Awareness and coordination in shared workspaces: ACM Conf. CSCW, p. 107-114.

Grudin, J., 1994, CSCW: History and Focus: IEEE Computer, v. 27, p. 19-26.

Gutwin, C., S. Greenberg, and M. Roseman, 1996, A Usability Study of Awareness Widgets in a Shared Workspace Groupware, Proc. of CSCW'96, ACM Press, p. 258-267.

Jacobson, I., G. Booch, and J. Rumbaugh, 1999, The Unified Software Development Process, Addison Wesley Longman, Inc.

Maes, P., 1997, Agents that reduce work and information overload, *in* J. M. Bradshaw, ed., Software Agents: Menlo Park, CA, AAAI Press, p. 145-164.

Malone, T., K. Grant, and K.-W. Lai, 1997, Agents for information sharing and coordination: a history and some reflections, *in* J. M. Bradshaw, ed., Software Agents: Menlo Park, CA, AAAI Press, p. 109-143.

Nwana, H. S., 1996, Software agents: an overview: The Knowledge Engineering Review, v. 11, p. 1-40.

Pinkwart, N., 2003, A Plug-In Architecture for Graph Based Collaborative Modeling Systems, *in* U. Hoppe, F. Verdejo, and J. Kay, eds., Shaping the Future of Learning through Intelligent Technologies. Proceedings of the 11th Conference on Artificial Intelligence in Education: Amsterdam, IOS Press, p. 535-536.

Wooldridge, M. J., and N. R. Jennings, 1998, Agent theories, architecture, and languages: a survey, *in* M. J. Wooldridge, and N. R. Jennings, eds., Intelligent agents: Berlin, Springer-Verlag, p. 1-39.