Web-Based Collaborative Software Modeling

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics
Software Design

presented by
Haidar Osman

under the supervision of
Prof. Michele Lanza
co-supervised by
Fernando Olivero

January 2013

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Haidar Osman
Lugano, 30 January 2013

To Dima, my friend, my partner, my love, my life, the woman who knows how to make her man reach beyond limits . . .

To Rima, the joy of my life . . .

To my father, mother, and sister for their unconditional love and unlimited support . . .

A clever person solves a problem.
A wise person avoids it.

Einstein

Abstract

Software modeling is a fundamental process in any software development methodology. This process aims at identifying the core elements in the system and the relations among those elements. It is also the phase where many important design decisions are made and those decisions will affect the quality of that software system through all the rest of the development phases. Software development is a social activity and includes extensive collaboration throughout the software lifecycle including the modeling phase. In modern software industry, many development teams are geographically spread out, making collaboration an important issue.
In this thesis, we present a new approach to support development teams carry on modeling sessions even when the members are far away from each other. Our approach is based on real-time change-based collaboration that makes team members work together on the same model at the same time taking into consideration the importance of awareness among team members. To demonstrate our ideas, we built a tool, Sawa, a web-based collaborative software modeling tool that allows team members to work together on the same model at the same time. Sawa also increases team awareness by its highlighting system that makes all team members know who is doing or has done what. It also allows users to replay the building process of a model to get the full picture of how a model has reached its current state.
We ran an evaluation experiment on Sawa and gathered qualitative feedback that supports our belief that our approach increases the productivity of modelers and helps them solve conflicts as they happen to avoid future complications, thus leading to better design decisions and better models in general.

Acknowledgements

First of all, I'd like to thank is my advisor, Prof. Michele Lanza for the brilliant ideas and unlimited support. He was not only my scientific advisor, but also my mentor.
Also I want to thank Fernando Olivero for his valuable advices and guidance throughout this thesis work.

I want to thank my friends Bisrat and Michel. You guys were like brothers to me and we really shared a lot during this master program.
I have to thank also my friends Luca, Remo, and Roberto for the great advices and help they sincerely offered.

Finally I want to express my gratitude to the whole REVEAL group. You are an amazing team and working with you was an unforgettable experience.

Contents

Figures

Tables

Chapter 1

Introduction

A Model is a simplified representation of the reality. Engineers model systems or phenomena to understand how a system will behave by identifying the concepts, entities, and dynamics of that system. The level of detail included in a model is strongly related to the goals of the modeling process. Models aim at finding flaws and defects, determining the cost of building the real thing, or serving as a plan. In computer science, software models usually serve as blueprints of software systems by determining their core elements, their relations, and their behavior at runtime. Omitting the modeling phase and going directly to implementation often results in a failure or an unmaintainable software system.
Software development is a social activity. It involves social interactions among team members who collaboratively make decisions, discuss problems, and solve conflicts. Since software modeling is a fundamental phase in software development, it inherently involves team work and collective decision making.
When software modelers are working in the same place, they can easily communicate and openly discuss their models. With the rise of global software engineering (GSE), collaboration among team members is an issue. When team members are geographically spread out, the existing techniques in software modeling include the use of communication channels, like instant messaging or emails, and file sharing mechanisms, like Dropbox or email attachments. This negatively affects the productivity of the modelers because they waste time in context switching, solving conflicts, merging models, and understanding what the others have done and how.
We present Sawa, a web-based collaborative software modeling tool that support geographically spread team members allowing them to work concurrently together and build software models. Our approach in Sawa provides real-time collaboration and an awareness system to facilitate the modeling process.

1.1 Contributions

The contributions of this work can be summarized as follows:

- A real-time change-based approach in collaborative software modeling to support software modelers when they are geographically far from each others.
- New ways to increase team members awareness by providing them with the necessary information about who is doing what, and how models reached a certain state.

- A web-based tool, Sawa, that implements our collaborative approach in software modeling.
- A qualitative experiment on Sawa to gather real users feedback regarding our approach.

1.2 Structure Of The Document

The rest of this document is organized as follows:

- **Chapter 2** we present the work related to our research in two main directions. The first direction is concerned with formal techniques, informal techniques, model-oriented programing, and source code based modeling. This part talks about the advantages, disadvantages, and opportunities provided by each technique. The second direction is concerned with collaboration in software development in general by demonstrating many tools and approaches that support team work and increase team awareness.
- In **Chapter 3** we formulate the research problem we are tackling and illustrate our approach justifying all our design and implementation decisions. We also describe our tool, Sawa, and detail its architecture including its design principles and patterns. We also talk about how Sawa supports collaboration and increases team awareness.
- In **Chapter 4** we detail the experiment we ran to qualitatively evaluate Sawa. We also explain our methodology to gather information and feedback and how we coped with the threats to validity.
- In **Chapter 5** we review and conclude this thesis discussing the work that has been done so far and the possible future research directions.
- In **Appendix A** we provide the handout that was used during the evaluation experiment.
- In **Appendix B** we list the raw results of the questionnaires conducted during the experiment.
- In **Appendix C** we explain how to use Sawa for building models.

Chapter 2

Related Work

Modeling is an important phase in the software development lifecycle. It includes brainstorming, drawing, communicating, and teamwork. During this phase, very important design decisions are made that affect the quality and maintainability of software. Most software development methodologies (like Waterfall [Roy87], Spiral [Boe86], and Scrum [RJ00]) include software modeling either as a separate phase or as an ongoing activity. There have been many research activities concerning better, easier, and more efficient ways and tools to model software. Because software development is actually a group work that involves many teammates, collaboration has been in the focus for many years now. Researchers are looking for better ways to integrate collaboration in all phases of the software development cycle.
In the subsequent sections, we go through the existing approaches in modeling techniques and collaboration styles showing their advantages, disadvantages, opportunities, and challenges.

2.1 Modeling Techniques

There are many techniques to model a software system. Some of those techniques are separated from the coding process and some of them are integrated with the code. Each technique has its advantages and drawbacks. In the following subsections, we go through these techniques semonstrating the state-of-the-art research in the context of software modeling.

2.1.1 Informal Techniques

Modeling, in general, serves as a way to visualize, abstract, and solve problems with a low cost. It is common that a group of developers gather for a discussion to discover new approaches to solve problems they face during the software lifecycle. The common aiding tools they use are papers, pens, and whiteboards [CVDK07]. These tools help the developers to sketch and formulate their problems and opinions in a free way. There are also CRC cards [Wil95] [BS97] where the modelers use small cards to write their views of the necessary information about classes in the system, as in Figure 2.1.
These methods are informal or lightweight. The main advantage of these techniques is that they allow the designers to draw sketches in a free and natural way without any strict rules to follow and without any prior knowledge of certain notations. The modelers can explore the problem space freely moving from one abstract view of the system to a more or less concrete

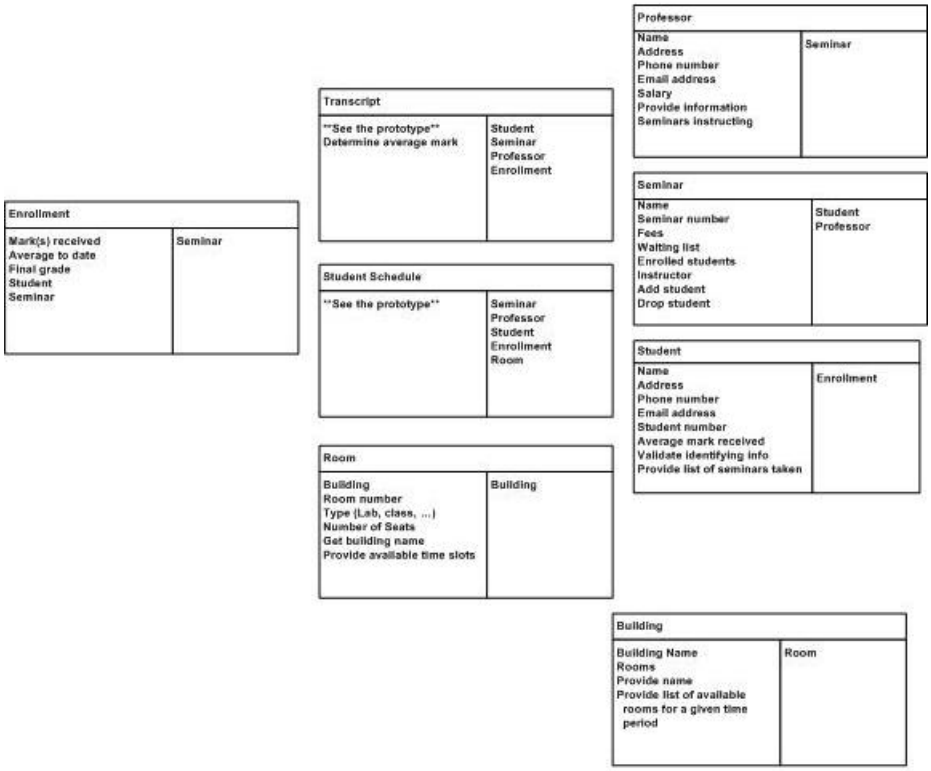
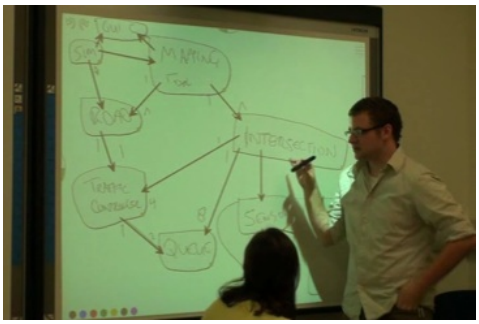


Figure 2.1. CRC model example

one [Goe91]. It is up to the modelers to decide how many details to put and also to use some formal notations if needed. Furthermore, the modelers can create new notations on-the-fly if necessary [DH07]. All these advantages support the creativity and the think-out-of-the-box mentality of the modelers which is highly sought after in any problem solving area in general and in software engineering in specific [Pet09].
Despite all the advantages of the lightweight techniques, there are a lot of drawbacks and disadvantages. First of all, the artifacts produced during the modeling sessions are not digitalized making them hard to store and almost impossible to manipulate and update. This is a major cause for the drift between documentation and models on one hand, and the real implementation on the other hand. Those artifacts are useful only at the time of the modeling sessions [SSR03]. Also it is hard for a person who was not present during these sessions to understand those sketches or the design decisions behind them. That makes those artifacts even harder to share and useless to maintain. That is why designers want to be able to manipulate and work with those sketches in a more complex way.
Because of those weaknesses, there has been research about how to overcome the lightweight modeling techniques drawbacks and many tools were introduced to aid those techniques. CREWW [BDL11], as in Figure 2.2a, is a tool that strengthens the CRC modeling process by digitizing it using software tools, a projector, and Wii remotes. What CREWW tries to do is to make it possible to record the modeling sessions and maintain the CRC cards in a digitalized

(a) CREWW Environment



(b) CALICO Environment

Figure 2.2. Using new tools to facilitate the modeling sessions with lightweight techniques

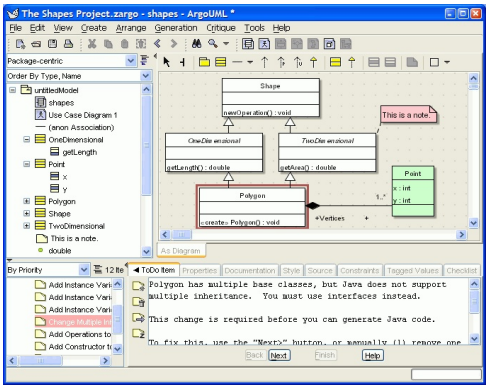
form.
Another tool, Calico [MBD+10], tries to benefit from the electronic whiteboard, as in Figure 2.2b, to keep all the advantages of modeling with whiteboards and to add the possibility to maintain, store, manipulate, and switch between the sketches.

2.1.2 Formal Techniques

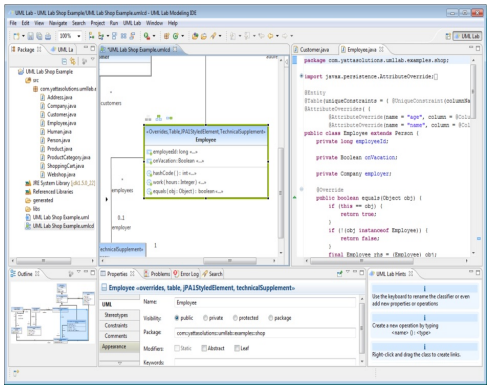
Formal modeling means the production of diagrams and sketches that comply with formal visual languages. UML is a widely-used representative of those modeling techniques. UML (Unified Modeling Language) is a visual notation for software modeling that is used as a way to visualize systems and also to serve as a clear communication standard among the designers themselves, between designers and management, and between companies and customers. UML is heavily used in industry and is recognized as a standard. UML artifacts are usually created using UML editors that are used to manipulate, save, and edit the models. These editors also ensure the consistency among different models to avoid conflicting design decisions.
UML and UML editors solve most of the problems introduced by lightweight techniques as described in Section 2.1.1. The models are digitalized and printable which makes them easy to maintain and keep updated. Also, since UML is a strict standard notation, it can easily serve as documentation and means of communication because it is easy to share the model files. Another advantage is that most UML editors can export code skeletons.
However, the formal techniques have their own drawbacks. The main drawback is the strict rules and the formalism of the standard notation. It affects the freedom and consequently the creativity of the modelers. With these techniques, the modelers focus on the right way of producing correct and clear diagrams which keeps them from putting more effort on discovering the solution space freely and examine more alternatives. Also modelers need background knowledge before starting to work with these techniques. Different UML editors have different file formats making it very hard to share model files among different platforms. There are many examples of UML editors. ArgoUML[1] as in figure 2.3a, Rational Rose[2], and Altova UModel[3] are among many of the UML tools out there. They provide all UML standard models

[1] http://argouml.tigris.org/
[2] http://www-01.ibm.com/software/awdtools/developer/rose/
[3] http://www.altova.com/umodel.html

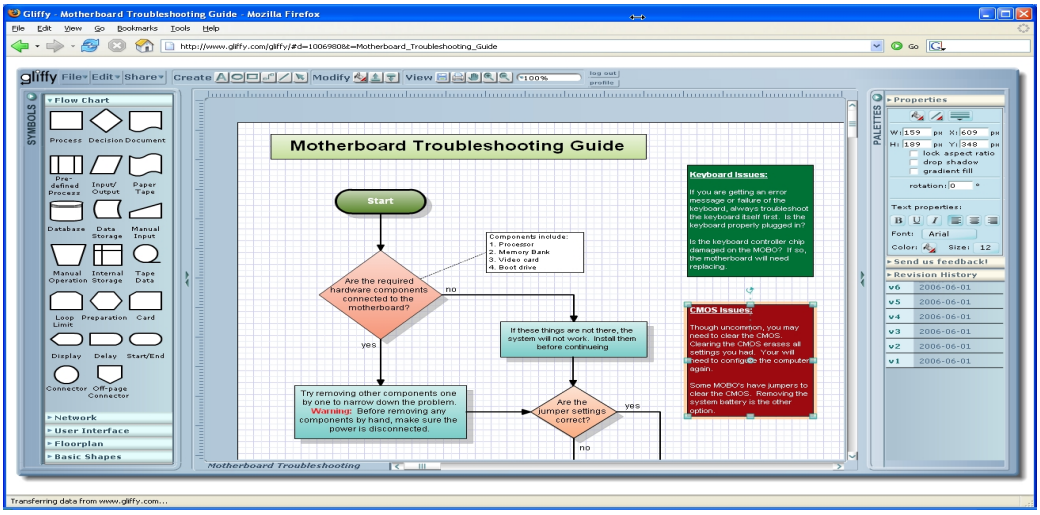
like Class Diagrams, Use Case Diagrams, Sequence Diagrams, and State Chart Diagrams. They also provide the possibility to generate code skeletons in many languages like Java, C#, and C++. These tools are model-oriented and not code-oriented. Meaning that the main focus of these tools is model creation and manipulation whereas there are many other tools that focus more on the code also like UMLLab[4] , as in figure 2.3b , which is an Eclipse plugin that integrates the code with the model. Any editing on the code will be reflected on the model using reverse engineering. All previous tools are desktop programs. They need to be installed on the local machine.



(a) ArgoUML is a stand-alone UML editor



(b) UMLLab is an Eclipse plug-in for UML editing



(c) Gliffy is a browser-based UML editor

Figure 2.3. Various UML editing environments

There are few browser-based UML editing tools that allow clients to create and edit UML

[4]http://marketplace.eclipse.org/content/uml-lab#.ULpB7COYXH0

diagrams from within the browser environment. yUML[5], web sequence diagrams[6], and gliffy[7], shown in Figure 2.3c, are good examples of the online UML editors. Although they are very simple and easy to use, they do not really take advantage of the online environment. They only allow you to share the models in a static way without any real-time collaboration support.

2.1.3 Model-Oriented Programming

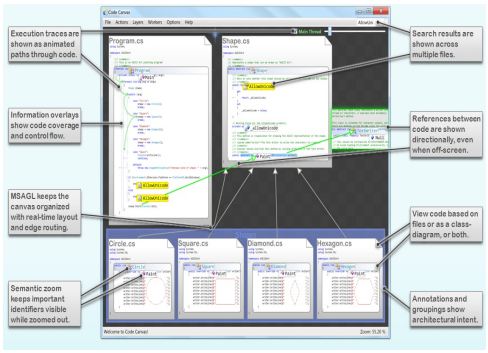
A drawback of formal modeling is the unavoidable drift between the model and the actual implementation of the system. There is a trend in research toward model-oriented programming (MOP). Model-oriented programming aims at adding modeling concepts to programming languages. With MOP, programmers can take a model-first approach or can incrementally reengineer an existing program to add modeling concepts. A MOP program is inherently human-readable text like any other programming language but it can also be represented in diagrams. With appropriate editors, changes can be made to texts and diagrams as well. This is called text-diagram duality. Because of that, there is no round-trip between codes and models. A compiler for a MOP language acts like a compiler for any other language. Programmers do not need to look at or edit the generated code.
Model-oriented programming can increase program comprehension because it raises the abstraction levels of common object-oriented programming languages [BFL12] and reduces the volume of code [FLB09]. MOP also allows users to quickly generate a fully functional prototype that exposes modeling implications on the user interface, and allows stakeholders to get a feeling of how the full system will behave [FBLS12]. Another side-advantage is the facilitations MOP provides in teaching software engineering courses regarding UML and Modeling [LMFB11].
A good representative of a model-oriented programming language is Umple [Bad10]. Umple enhances languages like Java and PHP with textual modeling abstractions. It was designed to bridge the gap between textual and graphical modeling. It adds abstractions such as associations, attributes and state machines derived from UML to object-oriented programming languages such as Java, PHP and Ruby. One write Umple code to generate the diagrams or draws the diagrams to generate the Umple code. Umple can be used to generate complete systems (not only code skeletons) in many languages like Java, PHP, or Ruby. There is no need to add or edit the generated code because all the needed algorithms and methods can be included in the Umple code. There is a nice and easy browser-based Umple editor[8] where the user can write code, draw diagrams, and generate complete programs in many other programming languages.

2.1.4 Source Code Based Modeling

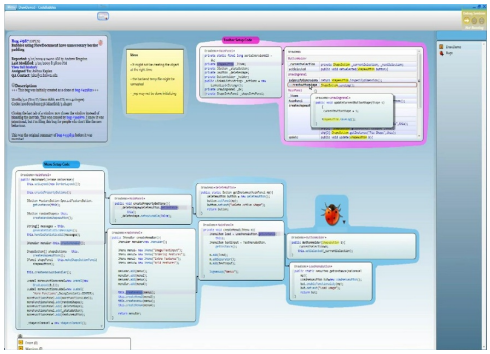
Most of the modern IDEs, like Eclipse[9], Visual Studio[10], and XCode[11], are file-based. They focus on the manipulation of source code and provide tools to support the process. The main disadvantage of those IDEs is the lack of support for code comprehension. This is a drawback

[5] http://yuml.me/
[6] http://www.websequencediagrams.com/
[7] http://www.gliffy.com/uses/uml-software/
[8] http://cruise.site.uottawa.ca/umple/
[9] http://www.eclipse.org/
[10] http://www.microsoft.com/visualstudio/
[11] https://developer.apple.com/xcode/

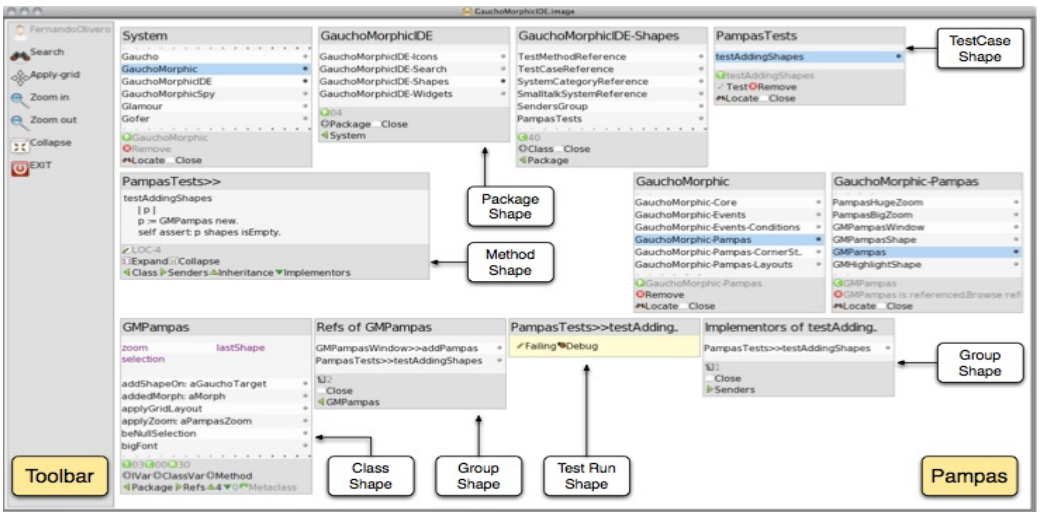
since we know from previous researches that software maintenance takes 90% of the total software cost [Erl00], 60% of which is spent on understanding the software [Cor89]. Research has been carried out to integrate software comprehension in the IDE by radically changing the IDE file-based style to new metaphors.



(a) Code Canvas: An IDE that supports the semantic zooming principle



(b) Code Bubbles IDE: Modeling by grouping



(c) Gaucho: An IDE based on direct manipulation of objects in a dynamic OOP system

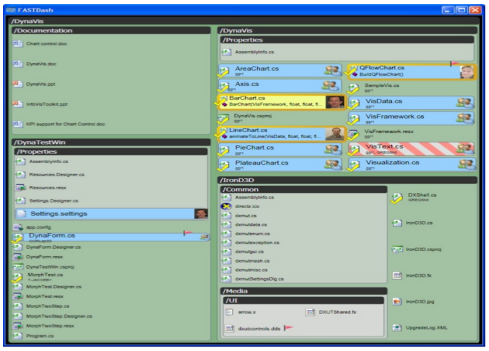
Figure 2.4. IDEs that support better comprehension of source code by visual notations

With Gaucho [OLL10], researchers have developed a simple yet effective IDE based on the direct manipulation of graphical shapes representing the software objects in dynamic OOP languages, as shown in figure 2.4c. Those graphical objects actually represent also a model of the system making use of the concept "The Code Is The Model" by the direct manipulation of live objects in that dynamic OOP system. Gaucho has been proven helpful and effective with respect to software understanding [OLDR11].
The same concerns were addressed in Code Bubbles [BRZ+10]. With Code Bubbles, the source code is fragmented into small segments, Bubbles, and arranged and grouped according to the relations among these bubbles, as shown in figure 2.4b.

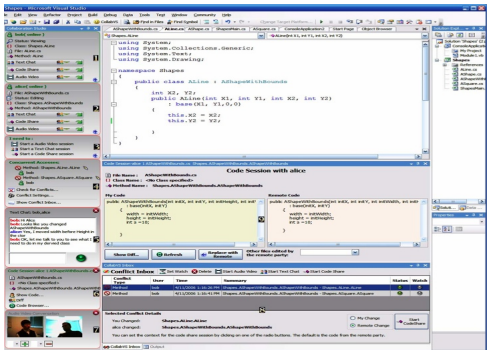
Code Canvas [DR10] which provides infinite zoomable canvas that shows the content of a software project with the necessary related information the developer needs. It allows multiple layers of visualization of the project's documents and each layer shows a certain level of details about the project. This is called semantic zooming, as shown in figure 2.4a. So when you zoom out one level you get a more abstract view and when you zoom in you get a more concrete view of the system in hand.

2.2 Collaborative Engineering and Modeling

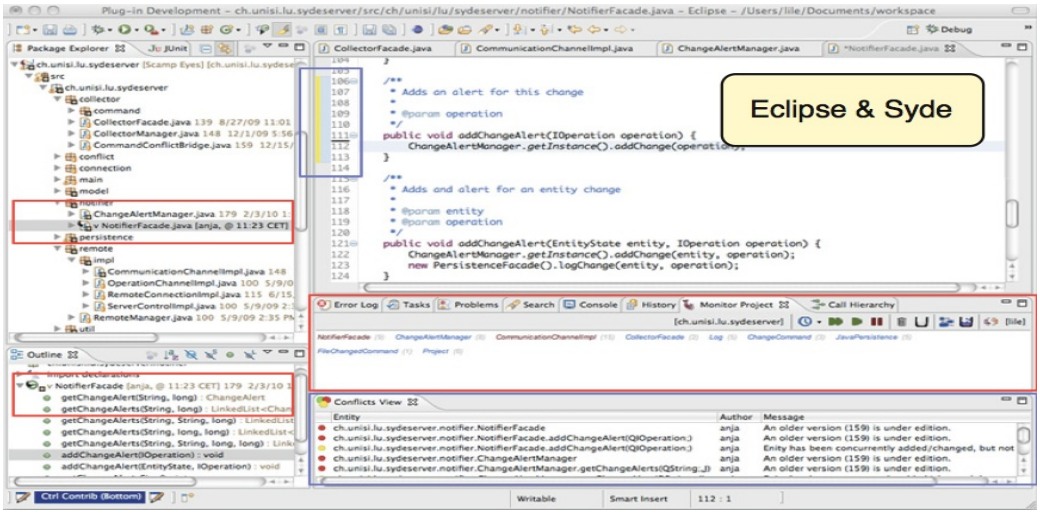
Team collaboration is essential for the success of a software project. Informal interactions and communications among team members in the workspace environment plays an important role for team coordination and awareness. Coordination can be seen as the process of managing dependencies among activities [MC94]. It is becoming an important topic in software engineering [DISK07]. Awareness is defined as an understanding of the activities of others to give a context for one's activities [DB92]. These two are the main aspects of team collaboration. When team members are located in the same workplace, they can collaborate easily and effectively using the informal communication techniques, and in the context of software modeling, they can use the lightweight modeling techniques as described in section 2.1.1. But awareness becomes a big issue in global software engineering, especially team members are geographically distributed [SvdH06].
The current software configuration management (SCM) systems provide a very structured and organized environment for team collaboration. But they suffer from a major drawback with respect to awareness because of the strategy of change propagation: Only when a team member checks in his changes, his/her teammates can access those changes and only when the synchronize the project with the repository they can actually see those changes.
Tools were developed to address awareness problems. FASTDash [BCSR07] is an interactive visualization that seeks to improve team activity awareness using a spatial representation of the shared code base that highlights team members' current activities, shown in figure 2.5a .
A richer and more complex tool is CollabVS [HD08]. It is a Visual Studio extension that augments the user experience with functionality aimed at collaborative, distributed development, shown in figure 2.5b. It allows developers to work together whether in an intentional or an ad-hoc manner. It allows the user to know what the other team members are doing, facilitates finding relevant information and people, and provides support for collaborative software construction. CollabVS even provides different communication channels like instant messaging and video conferencing.
Another tool is Lighthouse [dSCVdW$^+$06], an Eclipse plug-in that focuses on detecting conflicts as they occur by using a conflict avoidance approach to alert developers of potentially conflicting implementation changes as they occur, indicating where the changes have been made and by whom.
The aforementioned tools capture the changes at the file level, meaning that they capture added and deleted lines and propagate these changes to the team members whereas in Syde [HL10], the researchers have tackled the problem of providing awareness information by adopting a change-centric approach [RL07]. In Syde, which is an Eclipse plug-in as shown in figure 2.5c, the object-oriented systems are dealt with as abstract syntax trees and the changes are tree operations making the approach effective in solving the trade-off between providing relevant information about the activities of the team members, and avoiding overloading de-



(a) FASTDash: a tool for visualizing team members' current activities on the shared code base



(b) CollabVS: A Visual Studio extension for collaborative and distributed development



(c) Syde: An Eclipse plug-in for increasing awareness in collaborative software development

Figure 2.5. Different tools to support collaborative software engineering

velopers with irrelevant information.
At this point, all the coordination and awareness features mentioned previously were added features and functionalities to already existing IDEs that were designed without collaboration in mind. This is the main source of limitations regarding those tools. Whereas in Ronda [OLD12]. the researchers have developed a novel IDE from ground up to fully embrace the collaborative nature of software development. Ronda is built up on Gaucho and it is a change centric environment based on the canvas metaphor [OLDR11].

2.3 Summary

In the previous sections we have described the work related directly and indirectly to our research. We analyzed the state of the art in the context of software modeling and collaboration that influences our approach. One can see that collaboration among software modelers, when

they are apart from each other, is currently not investigated. The goal of our research is to find a new approach for facilitating collaboration among software modelers by combining the modeling techniques advantages, exploiting the work done in collaborative software development, and taking advantages of the new web technologies. In Chapter 3, we present a new approach that allows software modelers to build models together in a real-time collaboration manner. We also demonstrate a new web-based tool, Sawa, that implements that approach.

Chapter 3

Web-Based Collaborative Software Modeling

3.1 Problem Formulation

The problem we are tackling is **how to make modeling collaborative with geographically spread out teams**. To illustrate this problem best, we look at two scenarios: modelers are in the same place and modelers are far from each other.
When team members are having modeling sessions within the same room, collaboration is not an issue since they can easily communicate, discuss, solve conflicts, and make decisions together. They can use the whiteboard, CRC cards, pens & Papers, or even UML.
But when modelers are away from each others, modeling sessions will involve the use of some standalone software modeling tool, communication channels, like Skype or emails, and file sharing mechanisms, like Dropbox or email attachments. That will negatively affect their productivity because of the frequent break of flow they are forced to handle. Also every modeler will not be able to see the changes the others have done until they share them, Leading to many conflicts and inconsistent changes that need to be resolved.
Our approach tries to address these problems by supporting team members work together on the same model in real-time, and by providing mechanisms to increase their awareness of each other's activities.

3.2 Our Approach

Our approach aims at making software modeling easier, faster, and more natural by allowing multiple geographically spread-out teammates to work on the same sketch of class diagrams together at the same time in a way that improves each designer awareness of what the other designers are doing and who did what in the past, leading to less conflicts and to a more consistent model.

3.2.1 Modeling Technique

As we discussed in chapter 2, there are different modeling techniques, environments, and collaboration styles that we have to decide upon. Regarding the modeling techniques, we decided to go for a point in the middle between formal and informal techniques. We need some formality because when geographically spread-out users want to collaboratively work on the same model, sharing some standard notations is helpful so everybody can agree on the same meanings and principles behind the visual notations. Also we need some informality to support creativity and not to overwhelm modelers with rules they have to follow and details they have to provide. Our tool, Sawa, supports diagrams that are very similar to class diagrams in UML (formal) but without the constraining rules about types and class and method signatures. The diagrams represents entities (class shapes) with attributes and methods, and also association and generalization relations among these entities.

3.2.2 Collaboration

We want our tool to work inside a bowser making it platform-independent. This was particularly desired because we need the tool to support collaborative modeling. In this way there is no need for every team member to have the same platform or install any programs.
Regarding the collaboration style, our approach allows multiple accesses of the same model by different users. The changes made on the models are captured at the level of entities (model, view, class, attribute, method, relation), treated as entity operations (add, delete, update, move), then broadcasted to all clients. So now whenever a modeler makes a change to a model, this change will take effect in a real-time manner and all other modelers will see it immediately. We believe that real-time change-based collaboration is the best way for teamwork. It is more natural and helps solve problems the moment they arise. Also to support coordination, we want to provide a chat communication channel for each model to allow teammates to discuss problems, solve conflicts, and coordinate tasks on the fly.

3.2.3 Awareness

Since increasing awareness in the team is an important part of collaboration, we proposed the concept of client intention. Whenever a client double-clicks an item in the model, he/she is expressing the intention to do something with that specific item. So we catch this intention and broadcast it to all the other clients. The color of that item will be changed and the user name of the client working on that item will be displayed for everybody to know that this person is working on that item. This will naturally prevent people from working on the same item leading to less conflicts.
Also we provided the concept of views. One can have as many views as one wants in the same model. Views are basically different arrangements of the model objects. Each view can have part of the model or the model as a whole with the possibility to have objects appearing in any number of views and the tool will take care of keeping everything consistent. These views help modelers focus on different parts of the system and also see the model from different angles.
A client can work on one view at a time. In that case he/she will miss the changes made to other views of the same model. We added the change highlighting feature so all the changes that happened in the not-in-focus views will be highlighted along with the names of clients responsible for the changes. In this case, the client can miss the happening of a change, but

will always tell what happened and who did what.
Another interesting scenario is when a client goes away for some time and gets back later to continue working on the model. The model might be changed dramatically to the extent that the aforementioned client can't recognize it anymore. For that problem we propose the playback feature that allows the client to replay step by step all the changes that have happened since that last time he/she was online. In this case, that client can understand the evolution of the model and know exactly how the model reached the current state. The playback feature can even replay the building process of a model from the very beginning to support modelers who join the modeling sessions later after the process has already started.

3.3 Sawa in a Nutshell

To demonstrate our approach, we developed a new tool, SAWA[1], which is a web-based collaborative software modeling tool. Sawa has the following features:

Real-Time Collaboration

Sawa supports real-time collaboration where multiple users can work together on the same model at the same time. Whenever a user makes a change to the model, this change will be propagated to all online users, making them feel they are drawing on a shared whiteboard.

Highlighting System

Sawa support the concept of intention display. Whenever a client double-clicks on any item in the canvas, this means that he/she is showing the intention to change that item. This intention is propagated to all other clients by highlighting that specific item. The highlighting is made visible by changing the color of that item and displaying the name of the user working on it. This notifies all other users that this item is under editing by this specific user.
Sawa supports the concept of views where any model can have different views. A view is a different arrangement of the model classes. It can be a subset of the whole set of classes. A user can work only on one view at a time and cannot see the changes made to the other views. So to cope with this fact, whenever a change is made to a view other than the one in focus, this change is also highlighted with the name of the user responsible for that change.

Replay

When a user logs in to a model, he/she has the ability to replay step-by-step the building process of that model either from the beginning or from the last time time that user was online. In this way, the user can know how that model reached the current state.

Browser-Based

Sawa is a web-based tool that works inside the browser environment. In this way, we can relieve ourselves from making assumptions about the platforms or operating systems the team members are working on.

[1] Sawa page: che.inf.usi.ch:2128/

Complementary Services

Sawa has a textual console that logs every change made to the model along with the user responsible for that change. Also Sawa has an embedded chat service where users can discuss problems and resolve their modeling issues.

3.4 User Interface Design

The main principle behind our design of the user interface is "KISS: Keep It Simple & Stupid". The user interface is simple and straightforward to the extent that new users open the tool and start using it immediately without any instructions.

Figure 3.1. Sawa login page: It has login form, register button, and textual explanation about the tool.

When the user first opens the tool, he/she will be forwarded to the login page, as in figure 3.1. In this page, there is a textual description about the tool and its goals and features so the user gets some hints about what he/she is going to see and do. There is also a "register" button that allows new users to register themselves. It also contains the login form where the user enters the user name and password to move to the main page. One can see, in Figure 3.2, the user interface is basically divided into three sections: Two columns in the left and the right, and the canvas area in the middle. The left column contains the list of classes and the list of views. The user can switch from one view to another by clicking on the view names. Each class in the class list can be right-clicked and a menu will appear with two options: Add to the view or

delete. This is the only menu in the tool and every other action is preformed by manipulating the graphical drawings directly.

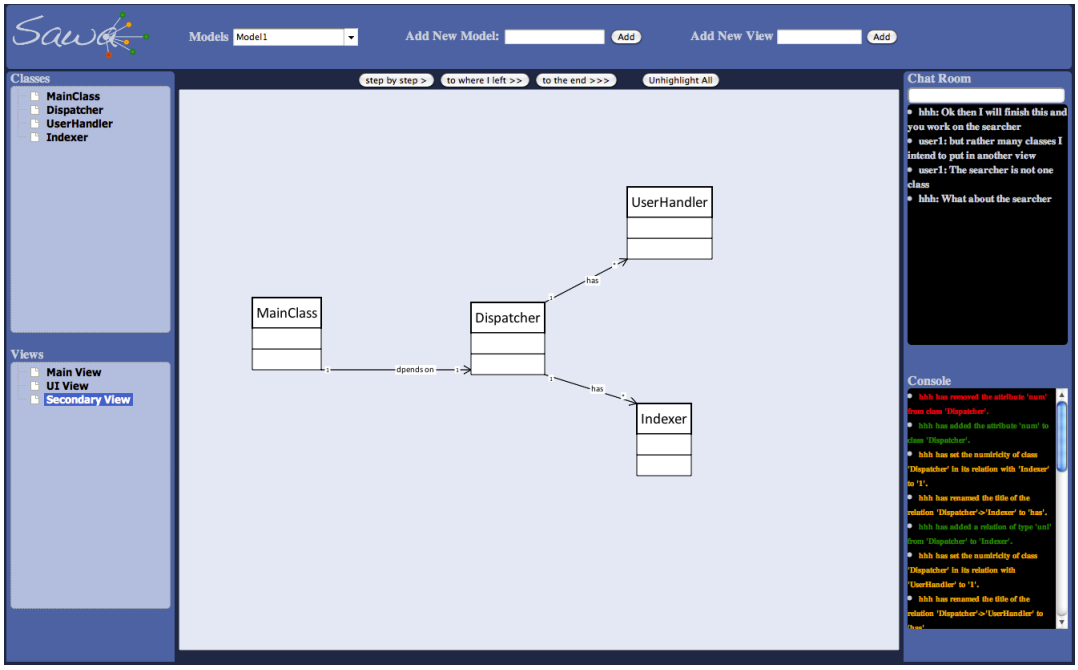


Figure 3.2. Sawa user interface

The right column contains the chat box, where users working on the same model can have discussions, and the textual console that explains everything going on on a model by text. The size of the text refers to the tree level of the change. In other words we have a tree of four levels. Each model has many views, each view has many classes. Each class has attributes, methods, and relations. So the smaller the text the lower the level of the change. Also there is a color indication of the nature of the change where green means addition, orange means update or edit, and red means deletion.

The middle area is the canvas area where users draw and manipulate the graphical objects. The user can double-click on any empty area in the canvas to create a new class. This class is added to the model and to the current active view. This class shape can be dragged and dropped anywhere in the canvas area. There are three sections in the class shape exactly like the UML notation: The class name, attributes, and methods. Each section can be double-clicked to be edited or deleted. There one empty attribute and one empty method areas that can be used to add new attributes and methods. When the user puts the mouse over a class shape, too side hooks and one top hook will appear. These hooks are used to draw relations among classes where the side hooks are for association relations and the top hook is for generalization relations. The relation is drawn by clicking on a hook then clicking again on another class. Each relation has a title and cardinality texts. They can be altered the same way the attributes and methods are. The relation is deleted when the title is deleted.

One can see in figure 3.2 that there are four buttons above the canvas area. The leftmost three buttons are used in replay mode where the user can replay the building process of a model as also described in figure 3.6. The right button is to unhighlight the current view. Unhighlighting

the view means removing the highlights of changes that happened while the current view was not in the focus of attention of the user, as explained in section 3.2.3. The highlights the user

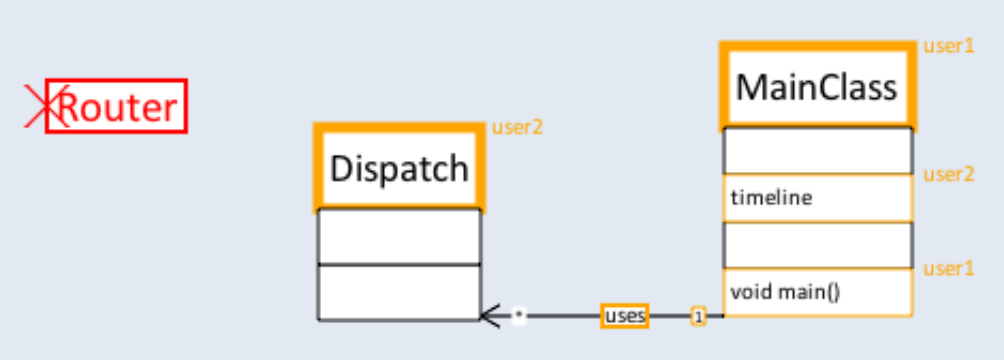


Figure 3.3. Highlighted objects to increase user awareness of what happened in his/her absence

might encounter are explained in figure 3.3. These highlights tells the user what was deleted, what was changed and which user did the change. The red highlight indicated that the object was deleted while the orange indicates that the object was altered. The orange highlights might occur even when the view in the user focus of attention but it then indicates that another user is working on that specific highlighted object. We believe that the highlighting system, the real-time change effect, the chat box, and the textual console increase user awareness and enhance the overall collaborative user experience.

3.5 Architecture

3.5.1 Implementation-Level Decisions

Framework

In Sawa, we need to draw shapes and to be able to manipulate them, so we need a canvas with user interactions events. We are inside the browser environment and we want it to work on all platforms, so we need a scripting language. We want to establish communication channels between the client and the server, so we need some kind of sockets. All these requirements lead us to decide first between two choices: HTML5 or Flash.
To decide between the two, we have to consider that Flash[2], in spite of its success, is still not regarded as a standard because it is a proprietary technology provided by Adobe[3]. Whereas HTML5[4] is going to be the replacing standard of HTML4[5] and XHTML[6] and it is being developed by World Wide Web Consortium (W3C)[7]. So we argue the future of web development should not be in the hands of an independent supplier and we are in favor of HTML5.

[2] http://www.adobe.com/products/flash.html
[3] http://www.adobe.com/
[4] http://www.w3.org/html/wg/drafts/html/master/single-page.html
[5] http://www.w3.org/TR/REC-html40/
[6] http://www.w3.org/TR/xhtml1/
[7] http://www.w3.org/

Client-Side

Since we decided on HTML5, we need a scripting language. We chose JavaScript which is an ECMAScript standard[8]. It is very powerful and has a lot of support by the open-source community. There are thousands of excellent free libraries that can be used easily and effectively. We know that JavaScript suffers from weaknesses and drawbacks but we believe that these drawbacks are shared among all the web scripting languages because they share the source of problems which is the Document Object Model (DOM)[9].
For canvas and drawings we decided to use the Kinetic library[10] which is a well-known framework that enables high performance animations, transitions, node nesting, layering, filtering, caching, and event handling of the HTML5 canvas.

Server-Side

On the server side we decided to go with NodeJS[11] which is a platform built on Chrome's JavaScript runtime for building fast, scalable network applications. We chose NodeJS because it uses an event-driven, non-blocking I/O model that makes it lightweight, efficient, and perfect for data-intensive real-time applications that run across distributed devices. Also it makes sense to use the same language and libraries in both client and server machines because that saves time and troubles developing consistent ways of communicating data.
We know that there is no problem sending data and requests from the client to the web server but the opposite is not true. HTTP is a connectionless protocol making it impossible for web servers to initiate the connection. Connection must be initiated by clients. For that reason, we chose the Socket.io[12] library that takes advantages of the WebSocket[13] standard (part of the HTML5 initiative) to make it possible to establish a communication channel between the server and the client. Socket.io not only uses WebSockets, but also can decide among alternative protocols in case the browser doesn't support WebSocket without affecting the API.

3.5.2 Overall Architecture

Sawa follows the client-server architecture style, as shown in figure 3.4. The client-server architectural style describes distributed systems that involve a separate client and server system, and a connecting network. The simplest form of client/server system involves a server application that is accessed directly by multiple clients. We combined the client-server style with the 3-tier style. The project is separated into segments of functionalities with each segment being a tier that can be located on a physically separate host. We chose this combination of styles because Sawa is working inside the browser environment making it inherently a client-server application. Also the server will not be a performance bottleneck because it has only to do routing and saving. All the functionality of drawing and manipulating the models will be handled in the client-side.
We also followed the command design pattern. In object-oriented programming, the command

[8] http://www.ecmascript.org/
[9] http://www.w3.org/DOM/
[10] http://kineticjs.com/
[11] http://nodejs.org/
[12] http://socket.io/
[13] http://www.websocket.org/

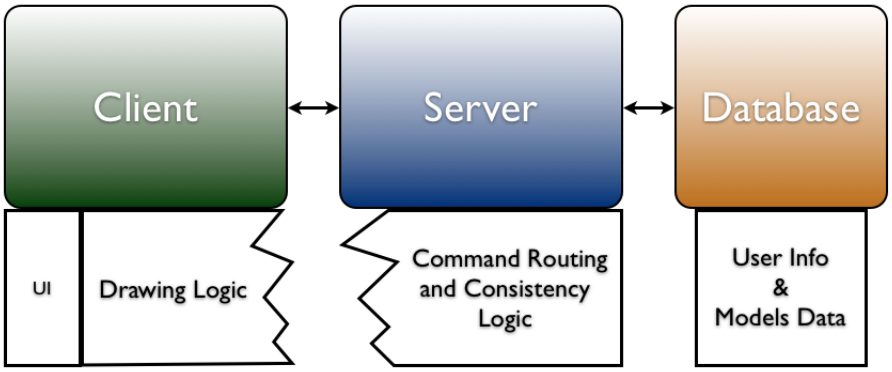


Figure 3.4. The overall architecture of Sawa

pattern is a behavioral design pattern where we can encapsulate, in an object, all the information needed to call a method at some point during runtime. This information includes the method name, the objects that owns the method, and the values of the method parameters. In other words, One encapsulates a request as an object, thereby letting him/her parameterize clients with different requests, queue or log requests, and support undoable operations [GHJV95]. In Sawa all user interactions with the user interface are captured and encapsulated in command objects to execute them only in the right time.
The collaboration style in Sawa is inspired by Croquet [SKRR03]. Croquet is a computer software architecture built from the ground up with a focus on deep collaboration among teams of users. In Croquet, a new collaboration architecture/protocol called TeaTime has been developed to enable this functionality. In Sawa, we implemented TeaTime tailored to our needs, as shown in Figure 3.5.
The steps of the algorithm is as follows:

a. Every user interaction with Sawa is encapsulated in a command and sent to the server.

b. The server adds a timestamp and a sequence number to the command.

c. The server stores the command in the database.

d. The server broadcasts the command to all clients (including the command issuer client).

e. The clients then executes the command.

When a client tries to do something, his/her interaction will not take effect until the command is sent to the server and received again with a timestamp and a sequence number. With this implementation we guarantee that all clients reach the same state.
The overall functionality of Sawa is explained in Figure 3.6. One can see that the client has three phases. The first phase includes login, establishing a session, and send a request to ask for the commands. The second phase is between getting the commands from the server and executing them. The client has three options in this phase. They are step-by-step execution of commands, go to the last known command, and execute all. The third phase is the actual collaborative modeling. This is where our Teatime implementation is in effect.

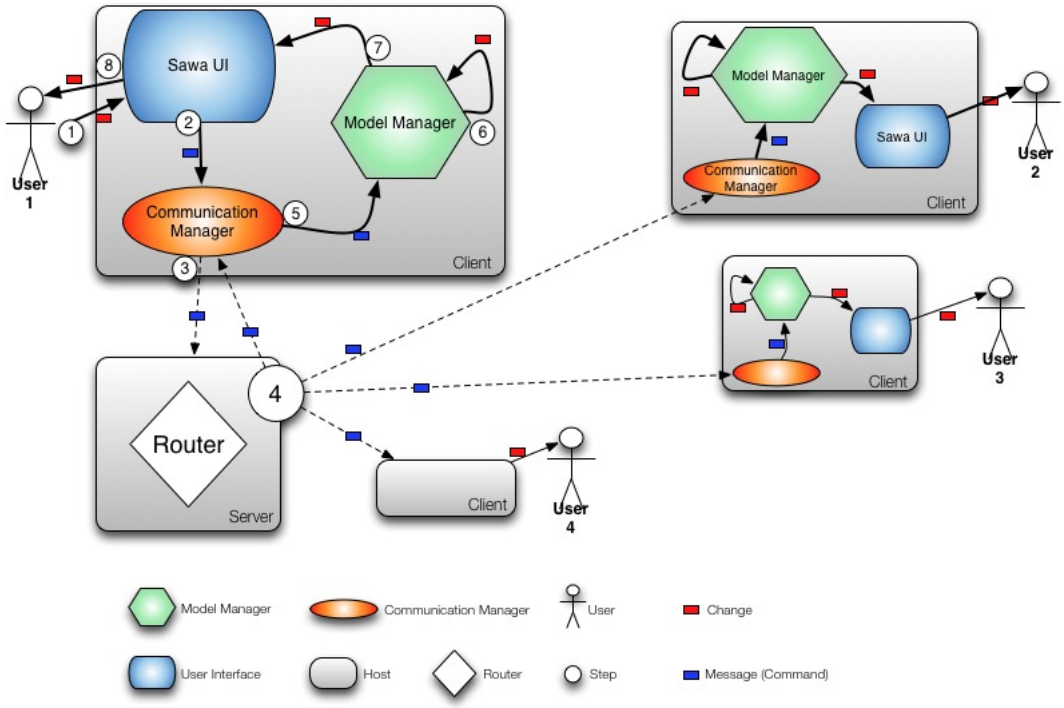


Figure 3.5. The implementation of TeaTime in Sawa

3.5.3 Server-Side Implementation

The server mainly consists of two components: The web server component and the router component, as shown in Figure 3.8. The web server component is responsible of accepting the HTTP requests and sending back the proper responses. The client can ask for the web pages, CSS files, images, and JavaScript source files. This component is also responsible for creating the web sessions and authenticating users. The router component is responsible for receiving the commands from the clients, adding the timestamps and sequence numbers to them, saving the commands in the database, and broadcasting the commands to all online clients as shown in figure 3.7.

As we said earlier, the server is implemented in JavaScript using NodeJS[14]. And to make things more organized and well-maintained, we used the Express[15] framework to build the web server component. Express is a minimal and flexible NodeJS web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications. With Express, we can define routes for web requests and handle each rout independently and asynchronously.

For the router component, we used the Socket.io[16] library that handles the initiation and maintaining of persistent connections between the server and the clients that will be used to exchange messages (commands). There are many communication protocols implemented in

[14] http://nodejs.org/
[15] http://expressjs.com/
[16] http://socket.io/

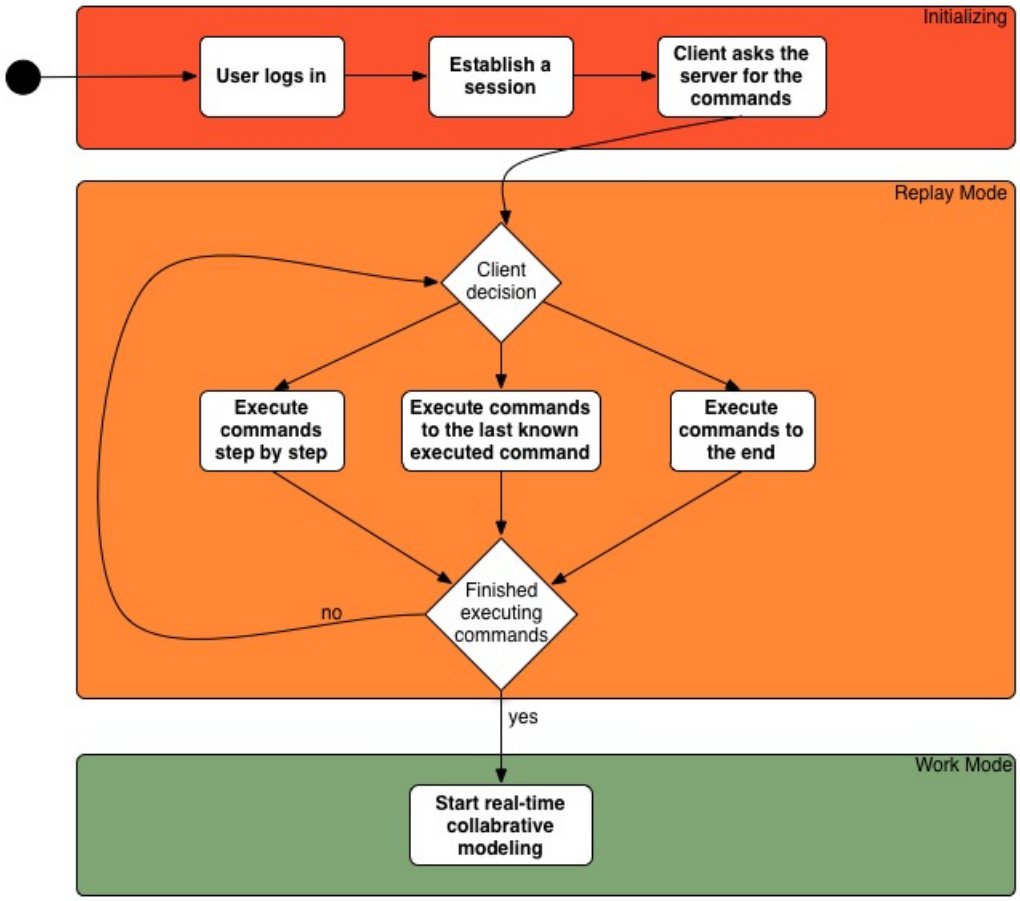


Figure 3.6. The overall functionality of Sawa explaining the three phases the client goes throw

Socket.io. The best protocol the client browser supports will be chosen. The protocols are chosen in this order:

a. WebSocket.

b. Adobe Flash Socket.

c. AJAX long polling.

d. AJAX multipart streaming.

e. Forever Iframe

f. JSONP Polling.

So whatever browser the client has, Socket.io will manage to establish the needed connection. This makes Sawa a highly platform-independent tool. The only necessary thing needed is the browser support of HTML5 canvas. This is something that cannot be replaced or altered.

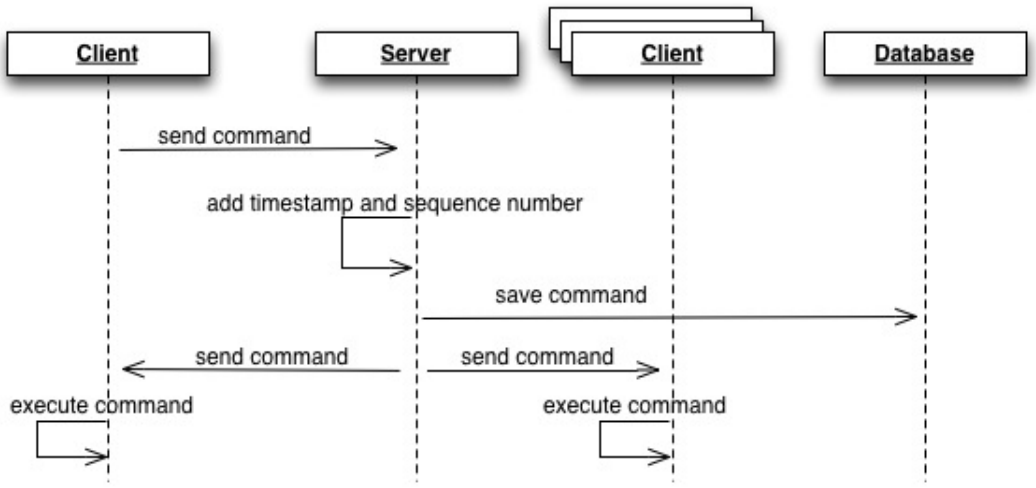


Figure 3.7. The process of routing the commands in Sawa

The two main components of the server expose two interfaces. The first one is HTTP and the other one is the Socket.io protocols. These two interfaces can be viewed as connectors as shown in figure 3.8.
All the commands that are created and sent by the client are JSON objects. It is an easy way to formulate, communicate, and execute commands. That was the main reason to use MongoDB[17] as the database system. MongoDB is a scalable, high-performance, open source NoSQL database that features Document-Oriented Storage. It stores JSON-style documents with dynamic schemas resulting in simplicity and power. This document-based database is the most suitable for our tool because we are able to store the commands JSON files as they are without considering any kind of translation or transformation. Besides, there is no sacrifice made in choosing the document-based database over the normal ordinary SQL databases. Document-oriented databases in general and MongoDB in specific offer all the needed features like full index support, replication, and querying.
The web server component is responsible for handling client HTTP requests. It consists mainly of three sub-components: The request handler, the user manager and the route manager as shown in figure 3.9. The request handler receives the request and pass it to the right component. Basically the request handler represents a facade of the web server component. The user manager handles user registration and authentication. It creates new user credentials, saves them in the database, and verifies user login information. The route manager takes requests from the request handler and sends the client the appropriate responses (files). When a request comes, the request handler decides which component is responsible for carrying it out based on the type of the request. Then it forwards it to the right component in an asynchronous, callback manner then returns to listen to requests. Then the specified component carry out the request and sends back the response itself, as shown in figure 3.10. For each client, the use of the web server component ends when the user is at the index page (the canvas) because after that, the router component will be the one responsible for listening to clients.

[17] http://www.mongodb.org/

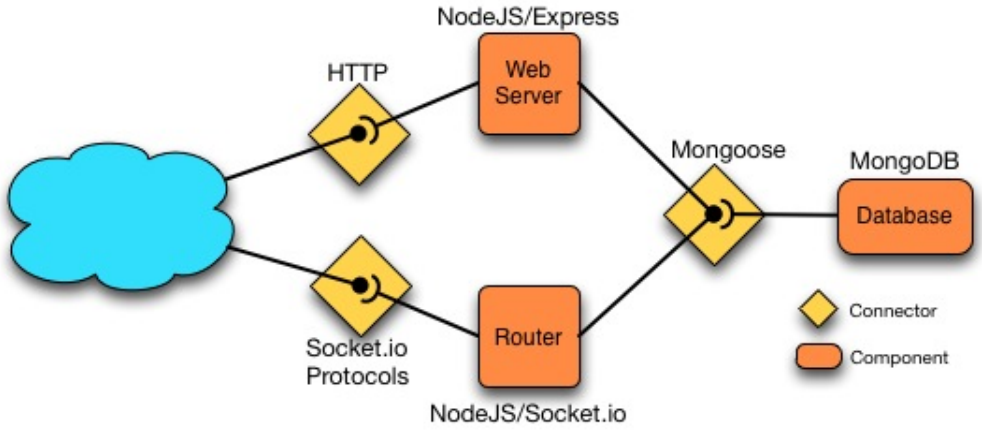


Figure 3.8. The architecture and the main components of the server in Sawa

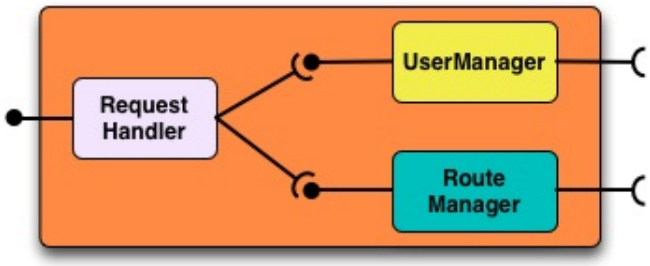


Figure 3.9. The sub-components of the web server component

The router component is is responsible for many functionalities. The first functionality is to listen to requests coming from the clients. These requests are not HTTP but they belong to one of the protocols of Socket.io mentioned in 3.5.3. The design of the router follows an event-driven architecture. Each request triggers an event based on the type of the request, as described in figure 3.11. The "getCommands" request is initiated by clients in two cases: when first the client logs in and when the client disconnects and reconnects again. The router sends back the list of commands and the client should handle them in the proper way. The "intent" request is sent from the client when the client shows an intent to do something, meaning that whenever the client double-click on something in the canvas, that means he/she wants to either delete it or edit it. This is perceived as an "intent" of action and is sent to the router that broadcasts it to the other clients to warn them that this particular item is about to be changed by this particular client. This increases team awareness of who is doing what and decreases the chance of conflicts happening. The "command" request is sent by the client when he/she actually does something. This command is then broadcasted to all clients, including the initiator, who executes it. After a client executes a command, he/she sends back a confirmation to the router in the form of "setLastExecutedCommand" request. The router then saves it in the database. This piece of information is helpful to track if the users are on sync and also helps in the replay

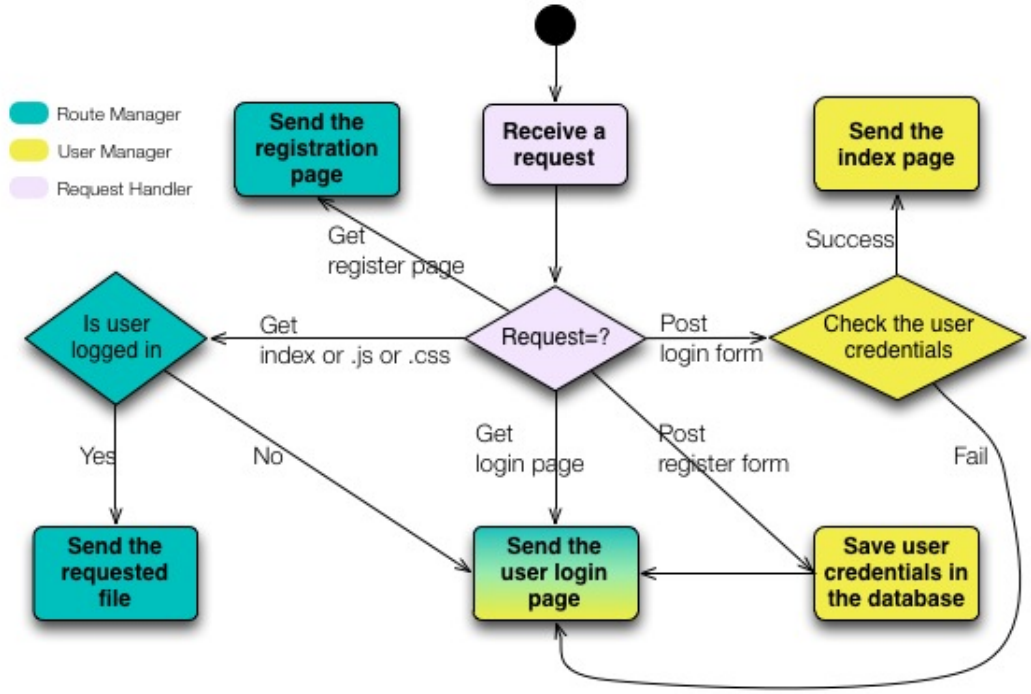


Figure 3.10. The detailed process of handling the HTTP requests showing the component responsible for each step

mode and the disconnect/reconnect scenario.
There are four kinds of connectors in the server. The first one is the regular HTTP requests where the exposed interface of the web server component is a listener to HTTP on a specific port. This called the "Web" connector. In the same way, the exposed interface of the router component is a listener for incoming connections on one of the supported Socket.io protocols. This type of connector is called the "Stream" connector. The other connector is between the router and the web server, and the database. It is written using "mongoose" which is a MongoDB driver that is suitable to the asynchronous callback nature of the NodeJS. This is basically a simple "Procedure Call" connector.

3.5.4 Client-Side Implementation

Since we decided on the client-server architecture, it makes sense to put as much functionality as possible on the client-side. This prevents the server from becoming a performance bottleneck. The client mainly consists of four components, as shown in figure 3.12.
The Communication Manager component is responsible for sending requests and receiving responses from the server. These requests and responses are in JSON format. The Model Manager component handles the binary representation of the models, views, classes, and relations. It also initiate the requests to the Communication Manager when necessary and gets back the commands from it and executes them with the help of the UI Updater and the Canvas. The UI

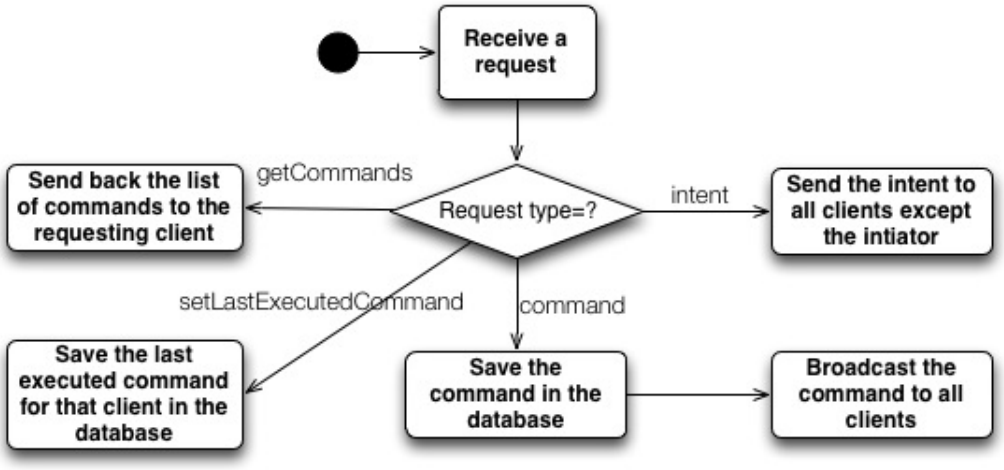


Figure 3.11. The type of requests and the corresponding triggered events in the router component

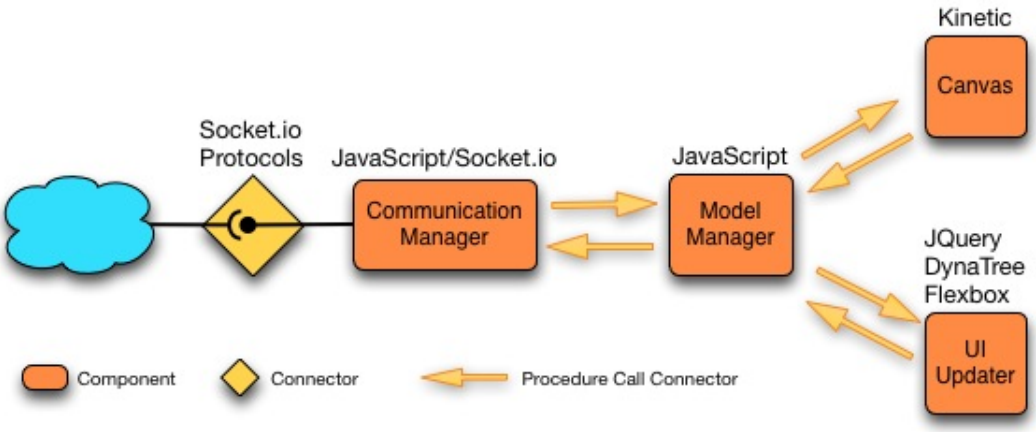


Figure 3.12. The main components of the client

Updater changes the HTML elements of the client web page. It is implemented using JQuery[18], DynaTree[19], and Flexbox[20] libraries. These elements are the models dropbox, the class tree, the view tree, the textual console, and the chat box. Figure 3.13 shows the main user interface of Sawa. The Canvas component is responsible for the drawing functionality. It is implemented using the Kinetic[21] library. This component handles the HTML5 canvas element and allows users to draw the classes, attributes, methods, and relations, and to move objects around.

[18] http://jquery.com/
[19] http://code.google.com/p/dynatree/
[20] http://code.google.com/p/dynatree/
[21] http://kineticjs.com/

All these interactions are encapsulated into commands that are passed to the Communication Manager through the Model manager, and sent to the server commands requests. When the command is received, the Canvas component is responsible of making the appropriate changes to the drawings.
The server accepts requests as HTTP or as one of the Socket.io protocols. The Communication Manager is implemented using the Socket.io library and sends requests of its protocols. There is no specific component in the client-side that initiates HTTP requests because this is the task of the client web browser.

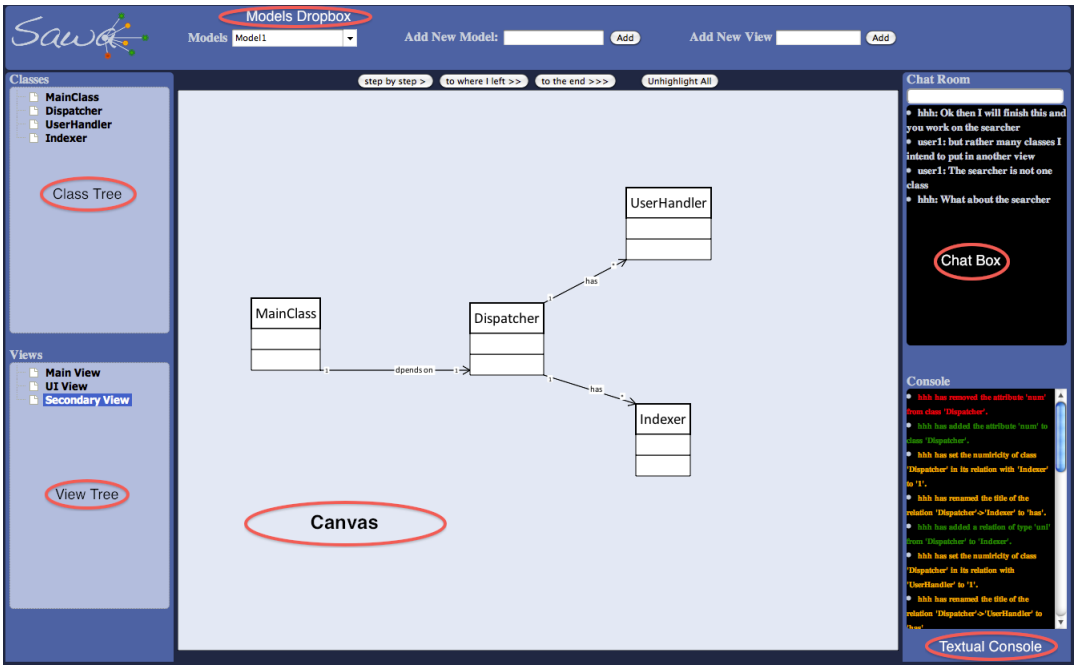


Figure 3.13. The main user interface of Sawa. One can see the HTML elements that change frequently as the coming commands are executed

The communication manager component is implemented using Socket.io in the same way as the Router component is implemented in the server-side. it is responsible for sending the requests explained in figure 3.11. In the same way, it is implemented using an event-driven approach. Whenever a message comes, it triggers the corresponding event, as explained in figure 3.14.
There are four possible types of messages that can be received by the Communication Manager. The "connect" message is sent from the server when the client first connects to the server. This message confirms that a connection has been established. The "intent" is sent from the server when it receives an "intent" itself from an another client and broadcasts it to other clients. In the same way, the client receives a command when the server broadcasts a command with a sequence number and a timestamp to all clients. The "commands" message is sent from the server whenever the clients asks for it. The clients asks for the "commands" message in two cases. The first case is when the client first connects to the server which is the normal case. The second case is when the clients faces a problem, disconnects from the server, and reconnects again. Each case is handled differently by the client as you see in figure 3.14.

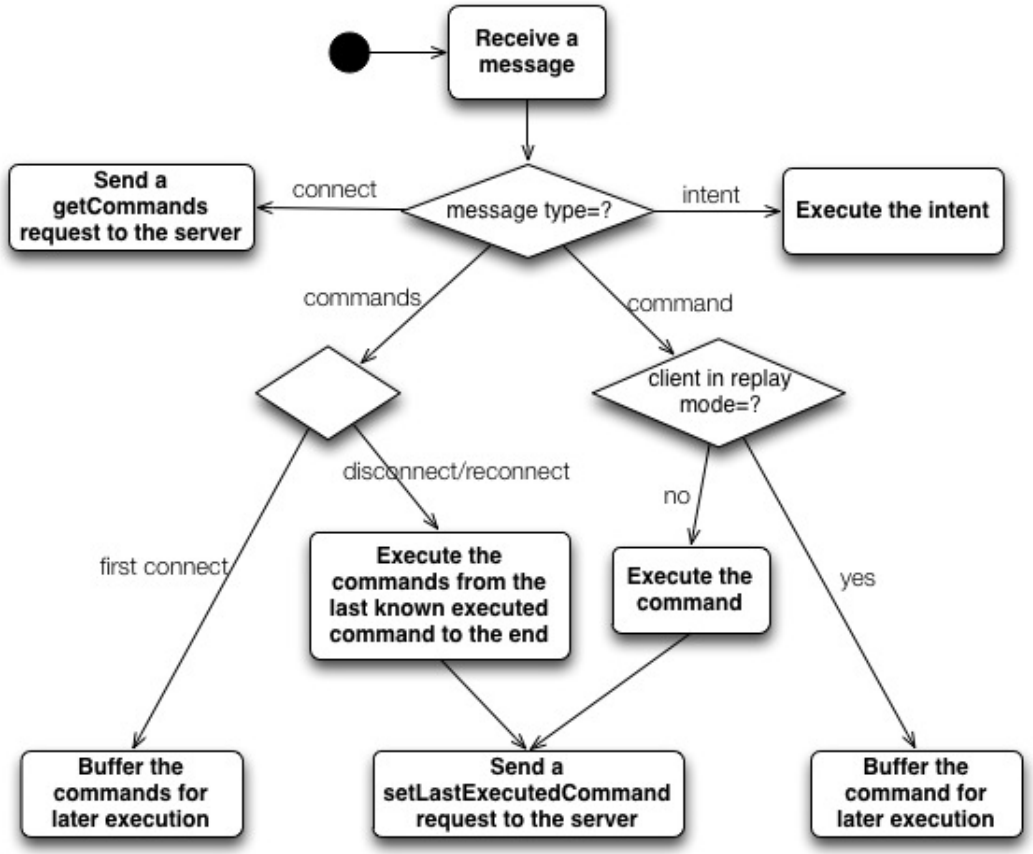


Figure 3.14. The types of messages received by the Communication Manager and the corresponding events

The Model Manager has some kind of hierarchical structure. It has many models, each model has many views, each view is a Canvas component. Also each model has the representation of all the relations among its classes. The Model Manager is responsible for keeping everything consistent and handling user input mistakes like adding al already existing class for example. It also gets the messages from the Communication Manager, formulates the commands, and process them in the right way. And whenever a user interaction is captured, the Model Manager will encapsulate it as a command message and gives it to the Communication Manager to be sent to the server.
This component decides whether the user is in replay mode and how the buffered commands are executed. It also decides how the incoming commands should be executed. When command execution effect is not in the user focus of attention (different model or different view), then the Model Manager highlights the resulting effect making it easier for the user to recognize when he/she switches to the place where the change took effect.
The Canvas and UI Updater components are the ones the user interacts with. They are responsible for capturing user interaction and passing it over to the Model Manager. They also execute

the commands and transform them into real changes that the user can recognize. So drawing, coloring, editing , adding, deleting, highlighting, and unhighlighting are all functionalities provided by the Canvas and the UI Updater components.

3.5.5 Network Problems

Since we are dealing with the web, there is always a chance for unintended disconnect. Our tool handles the rebuilding of the connection by itself but it is not enough since the client application needs to rebuild also the graphical representations of the models. In the case of connect/reconnect, the client application asks again for the list of commands and executes them from the last known executed command. Thanks to the sequence numbers attached to each command message and to the confirmation messages of the last executed commands explained in figure 3.14 , the client will know for sure where exactly to carry on the execution as explained in figure 3.15.

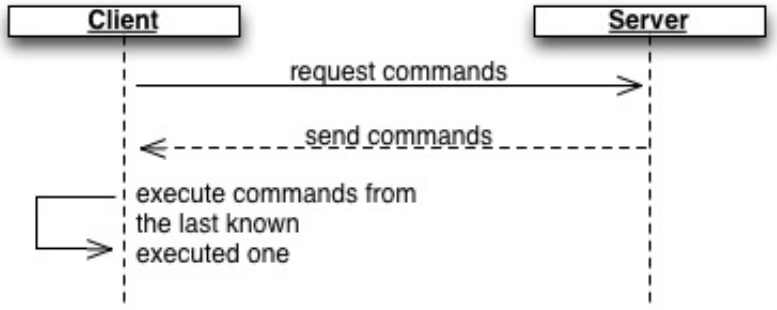


Figure 3.15. Explanation of the disconnect/reconnect scenario

Another problem originating from the internet network is that there is no guarantee that messages sent from the server will arrive in the same order to the client especially when there are many clients and they are all generating commands at high rates. Sequence numbers are again used. When the client application receives a message, it will know if this message is in the right order so it either executes it or waits till the other missing messages arrive as explained in figure 3.16.

Another important problem when dealing with distributed systems like ours is keeping all peers consistent and preventing them from diverging states. Our implementation of the TeaTime algorithm [SKRR03] guarantees that all clients will eventually reach the same state as explained in section 3.5.2. One can ask: what if two clients generates two commands at the same time and these two commands change the name of a class for example, which order will the commands be executed in? How to determine which command to execute first? The answer is that the order is determined by the router component in the server as the first command received is the first command to be executed. So in the situation of conflicting commands, the first one will take effect and the other one will fail to be executed. It makes sense if you consider the router as the timeline of clients' events. Even in life, the sense of "after" and

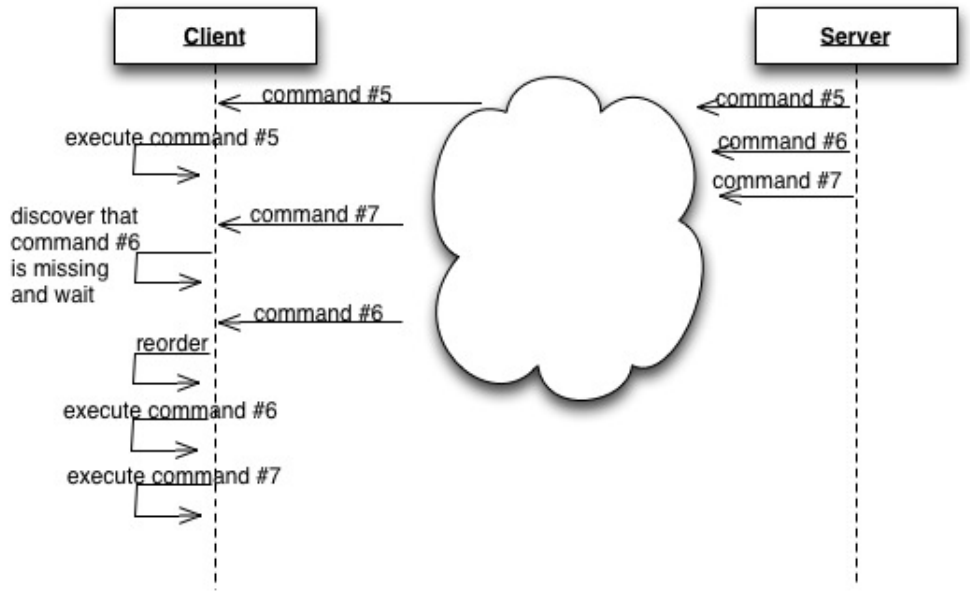


Figure 3.16. The client knows the right order of the commands and will execute them only in the right order

"before" of events cannot be determined unless all those events are measured against a central shared coordinate. In our case, this shared coordinate is the router component receive time.

Chapter 4

Evaluation

In chapter 3, we presented Sawa, our modeling tool and explained its architecture, design principles, and goals. We built Sawa believing that it is a helpful tool for software modelers when they are geographically separated. To support our claims and ensure that we are on the right path, we conducted an evaluation experiment to gather data, feedback, and suggestions we can use to spot the weaknesses and strengths of our approach to improve it and lead the future work in this specific area. Although the experiment results are not statistically significant because we had only eight participants, we can still use the results as a qualitative indication of the quality of Sawa's features and design decisions.

4.1 Experiment Design

The experiment was designed to get feedback from the participants regarding collaborative modeling and to test our new approach against existing tools and techniques. Our experiment is divided into three phases: Pre-experiment questionnaire, modeling tasks, and debriefing questionnaire.

4.1.1 Pre-experiment questionnaire

We conducted a small questionnaire before starting the experiment. The aim of this phase was to gather miscellaneous information about the participant like personal information, favorite modeling technique, and skills and experience in the fields we are working on. The results of this questionnaire are listed in tables 4.1 , 4.2 , and 4.3. We had eight participants and one can see from the results that those participants were mostly experienced modelers. Our aim is to evaluate the collaborative side of our tool and not to evaluate a new modeling technique or method.

4.1.2 Modeling Tasks

The pre-experiment questionnaire was conducted and the results were collected before running the experiment because we needed those results for the modeling tasks. The modeling tasks are two examples of software systems taken from the course "Software Architecture & Design"[1]

[1] Software Architecture & Design Course: http://www.master.inf.usi.ch/msd-course-detail?id=2289

Table 4.1. Pre-Experiment Questionnaire Personal Data Results

ID	Gender	Age	Position	Favorite Modeling Mean
P1	Male	21	Bachelor Student	UML Editors
P2	Male	24	Master Student	Whiteboard
P3	Female	28	Master Student	Whiteboard
P4	Male	28	Teacher Assistant	Whiteboard
P5	Male	27	Teacher Assistant	No Modeling
P6	Male	30	Teacher Assistant	UML Editors
P7	Male	28	Teacher Assistant	Whiteboard
P8	Male	29	Teacher Assistant	Whiteboard

Table 4.2. Pre-Experiment Questionnaire Results showing the number of participants in each skill level in the fields we are working in.

	None	Beginner	Intermediate	Advanced	Expert
OOP				3	5
Software Modeling		1	2	1	4
Using ArgoUML	2		1	2	3
Team Work	1	2		4	1
Using Skype					8

Table 4.3. Pre-Experiment Questionnaire Results showing the number of participants in each experience level (measured in years) in the fields we are working in.

	<1 Year	1 to 2 Years	3 to 5 Years	6 to 9 Years	10 Years or More
OOP			4	4	
Software Modeling	1	1	3	2	1
Using ArgoUML	3	1	4		
Teamwork	2	3	3		
Using Skype			3	5	

which is taught by Prof. Cesare Pautasso[2] at the master level at the University of Lugano[3]. The participants were divided into two groups taking into consideration that both groups are similar regarding experience level, age, position, and favorite modeling mean. So the first group G1 has the participants P1, P3, P7, P8 and the second group G2 contains P2, P4, P5, P6. We let group members know each other's emails and Skype accounts to use them in the experiment.
The first task was about building a software model of a chess game with specific requirements. Both groups were handed the same task description and requirements. G1 was allowed to use only Sawa, whereas G2 was allowed to use ArgoUML, emails, Dropbox[4], and Skype. No participant was allowed to talk to anybody during the modeling session.
Both teams were given one hour to finish the models. To simulate the real world, we pulled P3 and P4 from the modeling session after 15 minutes from the kick off. Then we returned them back to their groups after five minutes. Then we pulled P1 and P2 and returned them back after five minutes. We continued pulling participants out of the session in the same way until every participant had his/her turn to be out of the modeling session for a little while. After that, all participant continued working together till the end of that task.
After finishing the first task, the two teams switched places for the second task. Both teams were asked to model a travel agency system with the same description and requirements but with G1 using ArgoUML and G2 using Sawa. We followed the same procedure again regarding pulling participants out and returning them back. Both teams worked on the second task also for one hour and the experiment was over.
In this experiment all participants went through the same circumstances, experienced the same tools, and worked on the same problems. This was helpful in gathering objective feedback with minimal biases. This feedback was collected by means of a post-experiment questionnaire.

4.1.3 Post-Experiment Questionnaire

All participants were asked to revisit their experience in the modeling tasks phase and fill out the post-experiment questionnaire. This questionnaire aimed at getting qualitative feedback on Sawa against ArgoUML. Some general questions were asked regarding usability, model manipulation, and teamwork modeling experience. Some other specific questions were asked regarding conflicts, awareness, and models completion percentages. After that, we asked some questions where participants all answer with free text about positive and negative aspects of Sawa, required and recommended features, suggestions, and ideas.

4.1.4 Threats to Validity

There are many factors that might influence the effectiveness and generalizability of this experiment. In this section, we address the threats of validity to our experiment and we explain how we tried to neutralize them.

[2] http://www.pautasso.info/
[3] http://www.usi.ch/
[4] https://www.dropbox.com/

Subjects

We had the pre-experiment questionnaire to gather information about the participants background in the field of interest regarding the modeling tasks to reduce the threat of having subjects with no competence who will influence the experiment results. We found that we have five subjects with skill level above intermediate (advanced and expert) in all fields and only one participant with no experience at all in teamwork. To mitigate the influence of that particular participant, we put him in the group that includes the expert participant in teamwork. Other than that, there was no particular under-skilled subjects that might have influenced the results. In our experiment, we wanted to evaluate the effectiveness of our collaborative approach in software modeling. This is the reason why most of the subjects were above intermediate and they all at least knew the basics of UML.

Baseline

We have chosen ArgoUML as the contender of our tool Sawa because it is a well-known UML editor with good reputation among UML fans. We made that choice because if the subjects do not like the chosen base tool, they might unconsciously favor Sawa ignoring its defects and over-emphasizing its features.

Training

We had only two participants with no prior experience in ArgoUML and all the other participants had worked with ArgoUML before. The threat of that situation affecting the results was eliminated by providing enough quick instructions on how to use both tools and we didn't start the experiment until we were sure that all participants had the necessary knowledge about both tools on order to work on the provided tasks.

Experimenter Effect

Since the experimenter is one of the authors of the approach and the developer of Sawa, there is a risk of being unfair and biased during the evaluation of the results. To mitigate this threat, we did not evaluate the created models but rather let the participants themselves evaluate their work regarding model completion percentage. Also the experimenter did not interfere in any phase during the experiment and was playing the role of a neutral observer and did not engage in any private interviews. The data was collected only by means of questionnaires and observations.

4.2 The Outcome

The outcome of the experiment we did comes from three sources. The first source is the post-experiment questionnaire where participants described their experience with Sawa and ArgoUML during the modeling sessions. The second source is the participants' opinions and suggestions in a free text manner. And the third source is our observation notes during the modeling sessions.

4.2.1 Questionnaire Results

Although our experiment aimed at getting a qualitative evaluation of our approach, we can summarize the results using numerical means. The results are not statically significant but they can serve as a good starting point for more experiments.
We asked the participants to rank Sawa and ArgoUML with respect to usability aspects including usability in general, model navigation, entities manipulation, model recognition, and collaborative teamwork. The participants chose the difficulty level for each of those aspects among "very difficult, difficult, intermediate, easy, and very easy". Those choices correspond to a scale from 1 to 5.

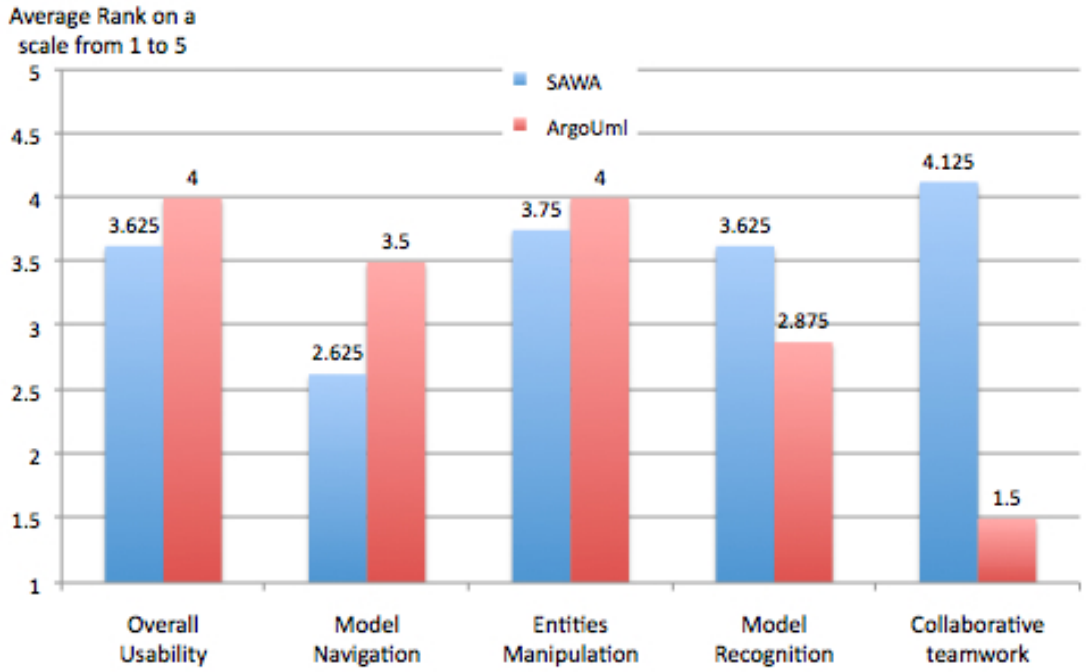


Figure 4.1. Comparison between ArgoUML and Sawa in many usability aspects where participants ranked these aspects on a scale from 1 to 5.

One can see from diagram 4.1 that ArgoUML is better than Sawa in the overall usability, model recognition, and entities manipulation. This is because ArgoUML has been under development and maintenance for many years while Sawa was developed in only four months. Another reason is that ArgoUML has many more features regarding UML design, coloring, themes, editing, and it has a huge canvas with a zoomable user interface (ZUI) making it more convenient and natural for users. But regarding model recognition and collaborative teamwork, Sawa showed better results.
Also the participants were asked some questions regarding collaboration and awareness like:

a. How often did you feel the need to talk to your teammates?

b. How often did you encounter conflicts?

c. How often were you aware of what your teammates are doing?

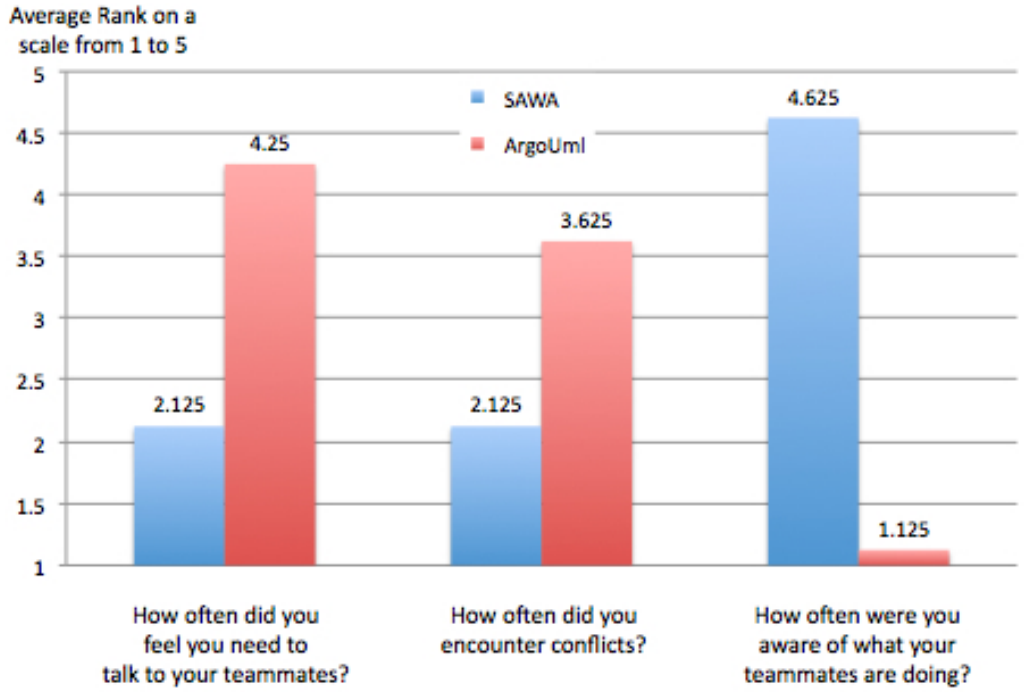


Figure 4.2. Comparison between ArgoUML and Sawa regarding conflicts, the need to talk, and awareness. Participants ranked the frequency of those events occurrences on a scale from 1 to 5 according to their experience during the experiment.

The answers to those questions were on a scale from 1 to 5 where 1 means never and 5 means always. Diagram 4.2 shows an advantage of Sawa over ArgoUML. Participants stated that they encountered less conflicts and had almost full awareness of what the others are doing. Also with Sawa, they showed less needs to talk to their teammates. These results are due to the highlighting system, the real-time change effect, and the replay feature included in Sawa.
Another question the participant were asked is about how complete their models were in both Sawa and ArgoUML. We allowed each team to evaluate their own models and the other teams model and we asked them to be objective and honest in their evaluation. We did not want to evaluate the models ourselves to avoid the threat of being unfair or subjective in our judgement. There is a threat that the two teams compete with each other and each team might rank its own model as more complete than the other team's model. This threat is eliminated because both teams worked with both tools on similar tasks. Also the diagram 4.3 shows consistent results in both tasks meaning that there was no competition in the evaluation. You can see that in both tasks, models were more complete with Sawa than with ArgoUML. Th reason is because of the intensive context-switching when working with ArgoUML. ArgoUML team members needed to chat a lot on Skype, share files on Dropbox or by emails, get back to the models, and keep the models consistent. This context-switching interrupts the workflow of people affecting their productivity. Whereas with Sawa, collaboration was easy and did not need any external communication channels. Again this was because of the real-time change effect, the textual console, and the chat system in Sawa.

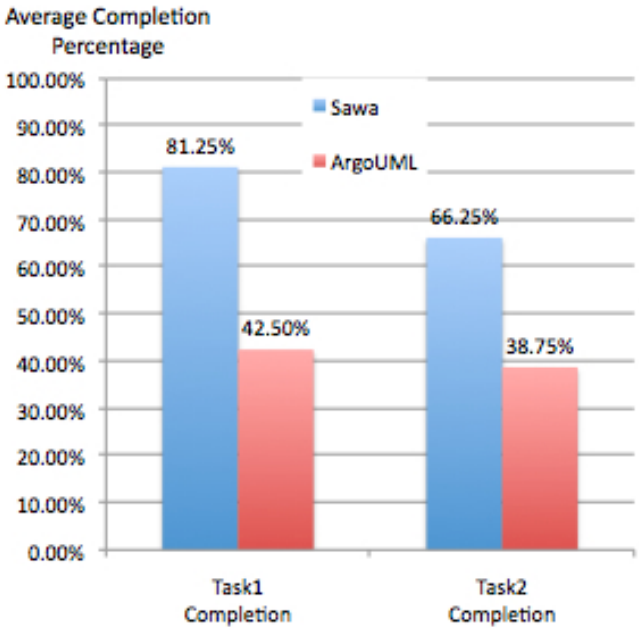


Figure 4.3. Comparison between Sawa and ArgoUML regarding task completion

4.2.2 User Feedback

The users feedback was collected by free text questions about opinions, suggestions, and ideas. We grouped these feedbacks in three categories.

Collaboration

All participants were enthusiastic about how they make a change to the model and everybody gets to see it immediately. They all gave positive feedback on how easy collaborative modeling is with Sawa. Here are some quotes from the participants:

"*It's so easy to work with others on the same models*" **P1**
"*I can instantly see the others changings to the model. This saves a lot of time*" **P5**
"*It was a very good way to benefit from the collective intelligence of the group*" **P6**
"*We wasted a lot of time discussing, when we worked with ArgoUML*" **P6**

All those feedbacks showed that the real-time collaboration in modeling is very well-appreciated by users when they are not present in one place making this approach the natural alternative of modeling sessions on a whiteboard.

Awareness

Only one participant stated that the highlighting system was annoying.

"*I hated it when i saw the color of some items changing to orange. It was annoying*" **P2**

While all the others appreciated the fact that with the highlighting system, they were totally aware of what items are under editing and by whom.

"*When I saw the orang color, I knew that I should wait to touch that item*" **P6**

"*Seeing the name next to the highlighted item made me know who to chat with about the change when necessary*" **P6**

"*When I switch to other views, I can easily see what happened*" **P8**

Regarding the replay feature, everybody said that it was very useful when they had to go out of the session and get back again after a while during the task modeling phase. We think that following that procedure equally with the team working with Sawa and the team working with ArgoUML made people realize how important it is to know how the model reached the current state in order to catch up with the teammates.

"*It was a nightmare when I had to leave the group for a while when I was working with ArgoUML but it was so easy to catch up when it happened with Sawa*" **P3**

"*Actually I used the replay feature once while I was with the group to remember how things got to be like that*" **P7**

Features

Most participants stated that Sawa needed to have more features that were, in their opinions, crucial to a modeling tool like Sawa. The most important feature they asked about was to have a zoomable user interface like the one in ArgoUML. That feature was pointed out by three participants (P2, P5, P6). Another feature requested by one participant (P6) was to include the package principle to the model where users can group classes together in different packages exactly like the package principle in UML. Also one participant (P8) said that it would have been nice to have a code generation mechanism.

4.2.3 Observations

During the two-hour modeling sessions we were watching, observing, and taking notes of what people are doing. We found out that when the team G2 was working with ArgoUML, they spent a noticeable amount of time in managing each member's task. They also agreed that P6 is the group leader and he is the one responsible for integrating other people work. The same thing happened with G1 but around thirty minutes after the start. G1 members agreed on P7 to do the task of integrating the models and keeping everything consistent. While with Sawa, there was no need to elect a leader and people were using the chat service to make decisions and solve conflicts collectively.
We also noticed some frustration on the faces of the teams when they working with ArgoUML, especially when a member left the team for a while and got back again. While with Sawa, team members kept excitedly working with full concentration on the tasks.
Another thing we found out is that ArgoUML team used a bigger amount of textual chatting than Sawa team in both tasks. We think this is basically because of the high level of awareness in Sawa.

4.3 Reflections

The results of our experiment are not statistically significant and we are aware of the fact that we need many more experiments to asses our approach in collaborative software modeling. Still, we believe that this experiment serves as a good starting point and gives a good enough indication about the success of our approach. Also we can benefit from the users feedback and include some more features and enhancements for further experiments.
Furthermore, the observations taken during the modeling session can help us improve the experiment itself and include more detailed questions in the post-experiment questionnaire in future experiment runs.
The evaluation work in this chapter concludes this thesis. In chapter 5, we summarize the work done so far and point out the possible research directions and future work in this field.

Chapter 5

Conclusions

This chapter ends this document by summarizing the work that has been done so far and describing our contribution with respect to collaborative software modeling. After that we explore the future work and the possible research points that can be further investigated.

5.1 Contributions

We know from literature that software development is a social activity. Modeling is a fundamental phase in any software lifecycle. In this research we tackled the problem of how team members can collaboratively work on a model when they are geographically far away from each other. There are many tools, methods, and frameworks that can be used to facilitate the modeling sessions but they all require the modelers to be present at the same room as explained in section 2.1.1.
Our tool, Sawa, is a web-based collaborative software modeling tool that enables many modelers to work on the same model at the same time in a real-time manner. It allows many users to collaboratively build a UML-like class diagram. Sawa comes with many features and design principles that serve the productivity of teamwork as follows:

- **Sawa is web-based:** It works inside the browser environment making it platform-independent. The only requirement the client should have is a modern web browser that supports the HTML5 canvas element. In this way, a client can go to the tool webpage and start working with the others immediately without having to install anything or worry about any kind of configurations.

- **Sawa is collaborative:** We designed and developed Sawa from the ground up to be collaborative. We took into consideration that Sawa should support multiple users working online at the same time where they can immediately see each others changes as if they were working together on the whiteboard in the same room. Sawa was designed as a scalable distributed system by adapting the TeaTime protocol to our needs to make it faster and more effective.

- **Sawa comes with a team awareness support system:** Awareness is an important aspect of teamwork. Teammates should have the ability to see who is doing what or who has done what. Sawa supports this through its highlighting system and replay feature.

Whenever somebody double-clicks on an item in the model, he/she basically shows the intention to edit it. This intention is captured and broadcasteded to all other online clients and that specific item is highlighted with the name of that user so everybody sees who is doing what. Also when a change happens outside the user's focus of attention (different view or different model), that change is also highlighted so the user gets to see later. Another awareness support feature is the replay mode that allows users, when they first logs in, to replay the building process of that model so they can tell what happened when they were absent.

- **Sawa has supplementary collaboration and awareness gadgets:** It has a textual chatting service that allows users to discuss their model problems, conflicts, and decisions on the fly. Sawa also has a textual console that displays all events in textual manner for people to read whenever they want.

We ran a preliminary evaluation experiment and got very positive feedback about our approach. That experiment results served as a good indication that we are on the right path of solving collaborative software modeling problems and also gave us very valuable information about what further can be done with Sawa and how to improve the experiment itself for the next runs.

5.2 Future Work

Adding Features

From the feedback we got from our first evaluation, we have some clues about missing features in Sawa. First, people seemed to admire the zoomable user interface (ZUI) as a flexible way to interact with the canvas. We should take into consideration changing our static canvas to ZUI infinite canvas with full support of zooming, expanding, and object locating mechanisms. Another feature we should include is the package support since users tend to like the idea of grouping classes together in different packages or groups although we didn't think that it was a necessary detail in software modeling in general but rather an implementation detail. Also we should add the code generation support to our tool in many programming languages where users can generate skeleton code derived from the model.

Other Models Support

Sawa support only the UML-like class diagrams but we should explore the possibility to add support to more important models like component models, state chart diagrams, and sequence diagrams. We should even examine the possibility to support free-style drawing where users can freely draw whatever they want on the shared canvas simulating the whiteboard technique. Or maybe we can add support the lightweight techniques mentioned in chapter 2.1.1 so their collaborative nature can be preserved and performed on the web.

Enhance Awareness Even More

There are many ways to enhance team awareness. For instance, there should be more teams organizing mechanisms like there should be a model creator (owner) who invites other modelers, by emails or usernames, to work on his/her model. Also we can display the list of users

working on that model with details regarding their status (online, offline, or away). Also the design of Sawa user interface should be revisited to display only the necessary information. Maybe it's better to display only the canvas and make the other page elements, see figure 3.2, more controllable with respect to display allowing the user to hide them, show them, pin them, or make them appear or disappear on certain events like on mouse-over or on status-change. Of course, we have to take into consideration that increasing the amount of displayed information doesn't necessarily mean increasing awareness. As a mater of fact, sometimes increasing the amount of displayed information can result in overwhelming the user with unnecessary details that can negatively affect the overall productivity. Awareness is providing the right amount of information at the right place in the right time.

5.3 Epilogue

Modeling is a fundamental activity in any software development methodology and it is also a social activity as much as software development is. In this thesis, we provided the necessary background knowledge regarding modeling and came up with a novel and innovative approach to support collaborative software modeling taking advantages of the new rising technologies. This work came from our belief that all software development activities are based on teamwork and modeling is no different and we should provide the necessary support not for single developers but rather to the whole team as a unit.

Appendix A

Experiment Handout

In this Appendix we present the experiment handout that was given to the eight participant to fill out. It includes the pre-experiment questionnaire, the modeling tasks descriptions, and the post-experiment questionnaire. As explained in section 4.1, those are the three phases of the experiment and they are consecutive in time as their names suggest.

SAWA Validation Experiment

Participant:	

Introduction:

This experiment aims to evaluate our tool, SAWA. It is a collaborative web-based UML designing tool that should make it easy for spread-out team members to build designs together in real time.

The experiment goes as follows:

There will be two teams. The first one will work on SAWA and the second one will work on ArgoUML. Both teams will work on modeling the same problem. One person from each team will start working 15 minutes after the rest of his/her teammates start. Then another member will go offline for 10 minutes then rejoin his team and work like this till the end of first task. Then the two teams will switch tools for the second task.

You are kindly asked:

- Not to talk to anybody during the experiment
- To work until you are satisfied about the model you built with your teammates
- Don't return to previous tasks as that will affect the experiment results

The experiment starts with a pre-experiment questionnaire to collect some information about yourself and your expertise in topics relevant to the experiment. Then you will go through the tasks and fill in the post-experiment questionnaire and have a little talk with the experimenter.

Thank you very much for your time.

Haidar Osman, Michele Lanza, Fernando Olivero

Pre-experiment questionnaire

E-mail Address:

_____

Gender:

❑ Male ❑ Female

Age: (for statistical purposes only)

_____ Years.

Nationality: (for statistical purposes only)

_____

Affiliation: (i.e., University, Company, etc.)

_____

Job Position: (i.e., Developer, Project Manager, MSc Student, etc.)

_____

What is your favorite mean to model software systems?

❑ Whiteboard / Paper Sketches.

❑ UML Editors.

❑ No modeling, I directly start programming in the IDE.

❑ None of the above: _____

Experience level in: (A subjective assessment of your skills)

	None	Beginner	Intermediate	Advanced	Expert
OOP	❑	❑	❑	❑	❑
Software Modeling	❑	❑	❑	❑	❑
Using ArgoUML	❑	❑	❑	❑	❑
Team Work	❑	❑	❑	❑	❑
Using Skype	❑	❑	❑	❑	❑

Number of years of: (the years you spent in acquiring your experience)

	< 1 Year	1 to 2	3 to 5	6 to 9	>10
OOP	❑	❑	❑	❑	❑
Software Modeling	❑	❑	❑	❑	❑
Using ArgoUML	❑	❑	❑	❑	❑
Team Work	❑	❑	❑	❑	❑
Using Skype	❑	❑	❑	❑	❑

First Task (Chess Game)

Given the following informal requirements and the usage scenarios, please create the software model that you think is the most appropriate. Please do not only model the class entities, but for each entity add the details (i.e., methods, fields and relationships) which you find important.

The requirements:

You and your Team should collaboratively

1. The game will be running on a very powerful machine in processing power, RAM, and hard disk.
2. The game should be able to adapt to the configurations and resources of the machine. The more powerful the machine, the bigger prospection the application can reach. In other words, when a user plays against the computer, the computer gets "smarter" as it gets more resources.
3. The user interface should be separated from the logic allowing us to support three different kinds of user interfaces:
    a. Command-line interface.
    b. Regular interface (desktop application).
    c. Web-based interface.
4. The games can be saved and loaded. And "epic" games can be replayed.
5. Inside the game, undo and redo should be supported.
6. The administrator of the system should be able to set the maximum number of concurrent active games.

The scenarios:

1. Learning: Each user should be able to study the strategies of other players. The user can load the last game of a certain player and replay it move by move. Also the user can watch other games online.
2. Training: Each player can play against the computer with three different difficulty levels; medium, hard, and master. Each player also can play against other players online.
3. Tournament: All games are locked. No players are allowed to play online. Only the players in the tournament can play with a real world chessboard and the jury will enter their moves into the system. Everybody can watch the tournament games and switch among games.

Second Task (Travel Agency)

Given the following informal requirements and the usage scenarios, please create the software model that you think is the most appropriate. Please do not only model the class entities, but for each entity add the details (i.e., methods, fields and relationships) which you find important.

System Description

Your job is to help migrating a set of legacy applications used in a travel agency to the web. The system is currently operated by workers answering phone calls from clients. Workers would ask for the relevant information to pull the client's file, ask for the destination, and list the choices to the client. The client would then be billed, and the tickets sent by traditional mail. To accomplish these tasks, the workers would interact with four different applications, sharing a central database containing customers' data. Each application runs on mainframe and is accessed with a terminal:

- An account management application, storing and managing information about clients and addresses. New client accounts would be created from here.
- A travel search application, in which the possible ways to reach a destination would be listed, along with their dates, duration and price.
- A billing service, communicating with credit card agencies. It checks if a credit card number is valid, and can charge credit cards.
- A ticketing system that prints and sends the tickets to the client after payment validation.

Design an integrated system in which the applications are accessible via the web, enabling the clients to search for their trips and buy the tickets themselves. Investigate how much of the existing functionality you can reuse, and how you will reuse it. In addition, the following new requirements should be considered:

- Authentication: Anybody can access the platform now, not only the workers, so the security has to be higher.
- Better communication between the legacy applications: In the new system, switching from one application to the next should be transparent: The user should feel there is only one application. In particular, each legacy application would need the account number of a client to access its information (the billing service to access the card number, the printing service to access the customer, address, ...). The workers would copy and paste it from one application to the next, but the integrated version should provide it automatically.

- A history of the trips of each client should be added, to propose discounts or promotions.
- Electronic tickets should be supported: The client can optionally print the tickets on his home printer, instead of having to wait for the tickets to be sent.
- Clients should be able to cancel their reservations.

Post-experiment questionnaire

Please revisit your experience with SAWA and ArgoUML and share with us your thoughts by completing this questionnaire:

	Very difficult	Difficult	Intermediate	Easy	Very Easy
Overall Usability					
SAWA	❑	❑	❑	❑	❑
ArgoUml	❑	❑	❑	❑	❑
Model Navigation					
SAWA	❑	❑	❑	❑	❑
ArgoUml	❑	❑	❑	❑	❑
Entities Manipulation					
SAWA	❑	❑	❑	❑	❑
ArgoUml	❑	❑	❑	❑	❑
Model Recognition					
SAWA	❑	❑	❑	❑	❑
ArgoUml	❑	❑	❑	❑	❑
Collaborative team work					
SAWA	❑	❑	❑	❑	❑
ArgoUml	❑	❑	❑	❑	❑

How often did you feel you needed to talk to your teammates?

SAWA: ❑ Never ❑ Rarely ❑ Regularly ❑ Frequently ❑ Always

ArgoUML: ❑ Never ❑ Rarely ❑ Regularly ❑ Frequently ❑ Always

How often did you encounter conflicts during the design session?

SAWA: ❑ Never ❑ Rarely ❑ Regularly ❑ Frequently ❑ Always

ArgoUML: ❑ Never ❑ Rarely ❑ Regularly ❑ Frequently ❑ Always

How often were you aware of what other members of your team are doing?

SAWA: ❑ Never ❑ Rarely ❑ Regularly ❑ Frequently ❑ Always

ArgoUML: ❑ Never ❑ Rarely ❑ Regularly ❑ Frequently ❑ Always

How hard was task 1 (Chess Game)?

- ❑ Very difficult
- ❑ Difficult
- ❑ Intermediate
- ❑ Easy
- ❑ Very Easy

How hard was task 2 (Travel Agency)?

- ❑ Very difficult
- ❑ Difficult
- ❑ Intermediate
- ❑ Easy
- ❑ Very Easy

On a scale from 0 to 100%, how much do you think your models are complete for task1 (Chess Game) and task2 (Travel Agency)?

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Task1											
SAWA	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
ArgoUml	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
Task2											
SAWA	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑
ArgoUml	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑	❑

Please answer the following optional questions to help us improve SAWA

What are the positive and negative aspects of SAWA compared to ArgoUML?

What are the improvements or new features that SAWA should provide?

Do you have any other suggestions, ideas, comments or critiques you want to share with us?

Appendix B

Experiment Raw Results

In this chapter we present the raw results of the questionnaires in Appendix A. The numbers inside the tables "Experience level" and "Number of years", as well as the numbers in the post-experiment questionnaire mean how many participants gave that specific answer to that specific question.

Pre-Experiment Questionnaire Results

Statistical Data:

Id	Gender	Age	Country	Affiliation	Position	Favorite mean
P1	M	21	SY	Tishreen University	BSc Stud.	UML Editors
P2	M	24	SY	Tishreen University	MSc Stud.	Whiteboard
P3	F	28	SY	Tishreen University	MSc Stud.	Whiteboard
P4	M	28	SY	Tishreen University	T.A.	Whiteboard
P5	M	27	SY	Tishreen University	T.A.	No Modeling
P6	M	30	SY	Tishreen University	T.A.	UML Editors
P7	M	28	SY	Tishreen University	T.A	Whiteboard
P8	M	29	SY	Tishreen University	T.A	Whiteboard

Experience level in**:** (A subjective assessment of your skills)

	None	Beginner	Intermediate	Advanced	Expert
OOP				**3**	**5**
Software Modeling		**1**	**2**	**1**	**4**
Using ArgoUML	**2**		**1**	**2**	**3**
Team Work	**1**	**2**		**4**	**1**
Using Skype					**8**

Number of years of: (the years you spent in acquiring your experience)

	< 1 Year	1 to 2	3 to 5	6 to 9	>10
OOP			**4**	**4**	
Software Modeling	**1**	**1**	**3**	**2**	**1**
Using ArgoUML	**3**	**1**	**4**		
Team Work	**2**	**3**	**3**		
Using Skype			**3**	**5**	

Post-Experiment Questionnaire Results

	Very difficult	Difficult	Intermediate	Easy	Very Easy
Overall Usability					
SAWA		**1**	**3**	**2**	**2**
ArgoUml	**1**			**4**	**3**
Model Navigation					
SAWA	**1**	**2**	**4**	**1**	
ArgoUml		**1**	**3**	**3**	**1**
Entities Manipulation					
SAWA			**3**	**4**	**1**
ArgoUml			**3**	**2**	**3**
Model Recognition					
SAWA		**1**	**2**	**4**	**1**
ArgoUml		**2**	**5**	**1**	
Collaborative team work					
SAWA			**2**	**3**	**3**
ArgoUml	**5**	**2**	**1**		

How often did you feel you needed to talk to your teammates?

SAWA:	[1] Never	[5] Rarely	[2] Regularly	[0] Frequently	[0] Always
ArgoUML:	[0] Never	[0] Rarely	[1] Regularly	[4] Frequently	[3] Always

How often did you encounter conflicts during the design session?

SAWA:	[2] Never	[4] Rarely	[1] Regularly	[1] Frequently	[0] Always
ArgoUML:	[0] Never	[0] Rarely	[3] Regularly	[5] Frequently	[0] Always

How often were you aware of what other members of your team are doing?

SAWA: [0] Never [0] Rarely [1] Regularly [1] Frequently [6] Always

ArgoUML: [7] Never [1] Rarely [0] Regularly [0] Frequently [0] Always

How hard was task 1 (Chess Game)?

- [2] Very difficult
- [4] Difficult
- [1] Intermediate
- [1] Easy
- [0] Very Easy

How hard was task 2 (Travel Agency)?

- [3] Very difficult
- [1] Difficult
- [3] Intermediate
- [1] Easy
- [0] Very Easy

On a scale from 0 to 100%, how much do you think your models are complete for task1 (Chess Game) and task2 (Travel Agency)?

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Task1											
SAWA								1	5	2	
ArgoUml				1	5	1	1				
Task2											
SAWA							4	3	1		
ArgoUml				4	2	1	1				

Appendix C

Sawa Quick Manual

Sawa[1] is a very simple easy-to-use tool that can be used without any tutorials. But still, here are some few instructions to know exactly what to do to build and manipulate models.

1. To create a new model, enter the model name in the text box next to "Add New Model" and press the button "Add".

2. Now you need to add a view to start drawing diagrams. Enter the name of the new view in the text box next to "Add New View" and press the button "Add". Of course you can add as many views as you want.

3. Now you can start building models by selecting a view from the view list on the left and work inside the canvas area in the middle.

4. Double click on any free spot in the canvas and a new class text box will show up. Enter the name of the class and press enter and a new class shape will be added to the canvas and its name will appear in the class list on the left.

5. You can drag and drop class shapes anywhere you want in the canvas area.

6. Every class shape is divided into three main sections: class name, attributes, and methods exactly like the class notation in UML. You can double-click on the empty attribute or method and enter some text and press enter to add an attribute or a method. You can also double-click on any attribute or method, change the text and press enter to alter that specific attribute or method. Or you can delete the text and press enter to delete that attribute or method.

7. In the same way, you can change the name of the class or delete it from that view. Remember that deleting a class from the view will not delete it from the model and that class will remain in the class list on the left because any class can appear in as many views as you want.

8. To delete a class entirely from the model you can right-click on its name on the class list and select delete.

[1] Sawa page: che.inf.usi.ch:2128/

9. To add an existing class to another view, switch to that specific view by selecting it from the view list then right-click on the class name in the class list and select add to view.

10. When you put the mouse over any class shape, you will see three handles appearing on the right, left, and top sides of that class shape. These handles are for drawing relations. To create an association relation, click on the left or right handles of the first class then click again on the second class. To create a generalization relation, click on the top handle of the first class and click again on the second class and a generalization relation will be added from the first class to the second class.

11. Each association relation has a title and two cardinality texts from both sides. You can change the text inside each one the same way you change an attribute or a method. You can delete a relation by deleting the title of that relation.

12. Remember that you can cancel any kind of operation (changing an attribute, method, relation title or cardinality, or building a relation) by clicking the escape button.

13. You can always switch between models by selecting the model name from the top drop box next to "Models".

14. When you first logs in, you first select the model you want to work on, then you have three possibilities. First you can replay the model building process step by step by clicking the button "step by step >" or you can go immediately to where you last left the modeling session by clicking the button "to where I left »" then you can go step by step again. Finally you can go to the current state of the model by clicking the button "to the end »>".

15. When you are working with a group, you will see many highlighted items as people are also changing things, especially when you switch among models or views. If you want you can always remove the highlightings by clicking the button "Unhighlight all".

Bibliography

[Bad10] Omar Badreddin. Umple: a model-oriented programming language. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 337–338, New York, NY, USA, 2010. ACM.

[BCSR07] Jacob T. Biehl, Mary Czerwinski, Greg Smith, and George G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1313–1322, New York, NY, USA, 2007. ACM.

[BDL11] Felix Bott, Stephan Diehl, and Rainer Lutz. CREWW: collaborative requirements engineering with wii-remotes (NIER track). In *ICSE '11: Proceeding of the 33rd International Conference on Software Engineering*. ACM Request Permissions, May 2011.

[BFL12] Omar Badreddin, Andrew Forward, and Timothy C. Lethbridge. Model oriented programming: an empirical study of comprehension. In *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '12, pages 73–86, Riverton, NJ, USA, 2012. IBM Corp.

[Boe86] B Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, August 1986.

[BRZ+10] Andrew Bragdon, Steven P. Reiss, Robert Zeleznik, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola, Jr. Code bubbles: rethinking the user interface paradigm of integrated development environments. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 455–464, New York, NY, USA, 2010. ACM.

[BS97] David Bellin and Susan Suchman Simone. *The CRC Card Book*. The Addison-Wesley series in object-oriented software engineering. Addison Wesley, 1997.

[Cor89] Thomas A Corbi. Program Understanding: Challenge for the 1990s. *IBM Systems Journal ()*, 28(2):294–306, 1989.

[CVDK07] Mauro Cherubini, Gina Venolia, Rob DeLine, and Andrew J Ko. Let's go to the whiteboard: how and why software developers use drawings. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Request Permissions, April 2007.

[DB92] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, CSCW '92, pages 107–114, New York, NY, USA, 1992. ACM.

[DH07] Uri Dekel and James D Herbsleb. Notation and representation in collaborative object-oriented design: an observational study. In *OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*. ACM Request Permissions, October 2007.

[DISK07] Daniela Damian, Luis Izquierdo, Janice Singer, and Irwin Kwan. Awareness in the wild: Why communication breakdowns occur. In *Proceedings of the International Conference on Global Software Engineering*, ICGSE '07, pages 81–90, Washington, DC, USA, 2007. IEEE Computer Society.

[DR10] Robert DeLine and Kael Rowan. Code canvas: zooming towards better development environments. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 207–210, New York, NY, USA, 2010. ACM.

[dSCVdW+06] Isabella A. da Silva, Ping H. Chen, Christopher Van der Westhuizen, Roger M. Ripley, and André van der Hoek. Lighthouse: coordination through emerging design. In *Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*, eclipse '06, pages 11–15, New York, NY, USA, 2006. ACM.

[Erl00] L. Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17–23, May 2000.

[FBLS12] Andrew Forward, Omar Badreddin, Timothy C. Lethbridge, and Julian Solano. Model-driven rapid prototyping with umple. *Softw. Pract. Exper.*, 42(7):781–797, July 2012.

[FLB09] A. Forward, T.C. Lethbridge, and D. Brestovansky. Improving program comprehension by enhancing program constructs: An analysis of the umple language. In *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, pages 311 –312, may 2009.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[Goe91] Vinod Goel. *Sketches of thought: a study of the role of sketching in design problem-solving and its implications for the computational theory of the mind*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 1991.

[HD08] R. Hegde and P. Dewan. Connecting programming environments to support ad-hoc collaboration. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 178–187, Washington, DC, USA, 2008. IEEE Computer Society.

[HL10] Lile Hattori and Michele Lanza. Syde: a tool for collaborative software development. In *Proceedings of the 32nd ACM/IEEE International Conference on*

Software Engineering - Volume 2, ICSE '10, pages 235–238, New York, NY, USA, 2010. ACM.

[LMFB11] T.C. Lethbridge, G. Mussbacher, A. Forward, and O. Badreddin. Teaching uml using umple: Applying model-oriented programming in the classroom. In *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*, pages 421 –428, may 2011.

[MBD+10] Nicolas Mangano, Alex Baker, Mitch Dempsey, Emily Navarro, and André van der Hoek. Software design sketching with calico. In *ASE '10: Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM Request Permissions, September 2010.

[MC94] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, March 1994.

[OLD12] Fernando Olivero, Michele Lanza, and Marco DÕambros. Ronda: A fine grained collaborative development environment. In Yuhua Luo, editor, *Cooperative Design, Visualization, and Engineering*, volume 7467 of *Lecture Notes in Computer Science*, pages 155–162. Springer Berlin Heidelberg, 2012.

[OLDR11] F. Olivero, M. Lanza, M. D'Ambros, and R. Robbes. Enabling program comprehension through a visual object-focused development environment. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 127 –134, sept. 2011.

[OLL10] Fernando Olivero, Michele Lanza, and Mircea Lungu. Gaucho: From Integrated Development Environments to Direct Manipulation Environments. In *Proceedings of FlexiTools 2010 (1st International Workshop on Flexible Modeling Tools)*, 2010.

[Pet09] Marian Petre. Insights from expert software design practice. In *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering on European software engineering conference and foundations of software engineering symposium*. ACM Request Permissions, August 2009.

[RJ00] L. Rising and N.S. Janoff. The scrum software development process for small teams. *Software, IEEE*, 17(4):26 –32, jul/aug 2000.

[RL07] Romain Robbes and Michele Lanza. A change-based approach to software evolution. *Electron. Notes Theor. Comput. Sci.*, 166:93–109, January 2007.

[Roy87] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[SKRR03] D.A. Smith, A. Kay, A. Raab, and D.P. Reed. Croquet - a collaboration system architecture. In *Creating, Connecting and Collaborating Through Computing, 2003. C5 2003. Proceedings. First Conference on*, pages 2 –9, jan. 2003.

[SSR03] Martina Schütze, Pierre Sachse, and Anne Römer. Support value of sketching in the design process. *Research in Engineering Design*, 14(2):89–97, 2003.

[SvdH06] Anita Sarma and Andre van der Hoek. Towards awareness in the large. In *Global Software Engineering, 2006. ICGSE '06. International Conference on*, pages 127 –131, oct. 2006.

[Wil95] Nancy M Wilkinson. *Using CRC Cards — An Informal Approach to Object-Oriented Development*. SIGS Publications, Inc., 1995.