THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

Department of Computing

BA(Hons) Degree in Computing

Final Year Project Report (2001/2002)

A Framework for an agent-based development environement with jini/javaspace (Real-time collaboration support)

Supervisor Name	: Dr. Stephen, Chan
Co-examiner	: Dr. Jane, Wong
Student Name	: Yim Wai Hin
Student ID	: 98133594d
Submit Date	: 19/4/2002

Abstract:

Software development requires the aid of many development tools to accomplish different tasks. It is impossible that all the tools are available or supported by an integrated development environment.

Based on Jini network technology, we propose a distributed software development platform for software development tools. Tools can be plugged into the platform dynamically and become members of distributed development environment.

Besides, we will also address the lack of support for real-time collaboration on current distributed systems. This project, we will extend a single user UML modeling tool to support real-time collaboration, as one of the tools of the propose development environment. Integration of the collaborative UML editor into the platform will also help to validate the functions of the propose system.

The result of the project is a distributed development environment, which contains a collaborative UML editor using versioning algorithm.

Acknowledgement

I would like to give my sincere thanks to my project supervisor, Dr Stephen Chan, who gave me genuine support and valuable advice throughout the development process of the whole project. He has been gentle and provided comments on the project so that I could finish the project smoothly.

I would also like to give my special thanks to my project co-examiner, Dr Jane Wong, for spending her valuable time on this project.

Last but not least, I would like to express my appreciation to my families, classmates, friends and those who supported my work in intangible ways.

Table of Contents

list of figures

1. Introduction

1.1 Problem statement

There are some problems in the current Integrated development environment:

1) No IDE can provide all features that satisfy every developer. This is the normal case that developers have to change their develop practice according to the IDE used. This make the development time longer.
2) In addition, most of modern IDE tools are standalone at local area network. This leads to location limitation. For example, the users need to download and install the program locally, which is very inflexible. Besides, nowadays the developer need to work on many difference machines and work on large number of tools, maintains these tools on many machines need extra administration.
3) Besides, most of these tools do not support real-time group work communication, which deduce the productivity of the team.

1.2 Proposed system

The proposed system is an online distributed development environment in order to break down the location boundary in software development.

1) There are some fundamental problems with today's network programming architecture, which does not fit in the distributed computing environment. For example, the protocol oriented communication is a not object oriented, non-standard ways of network communication; lack of failure handling and resource allocation is other problems. Sun Microsystem have proposed a jini network technology to solve these problems[5]. Jini is a more distributed environment ready network programming architecture, we will discuss more later. The proposed system will be based on this programming model.
2) The system is open. It means that it facilitates for integrating different tools, which is developed by different parties, in a dynamical way. In another words, the proposed system will focus on investigating the possibility on developing a framework for distributed tools integration.
3) Real-time collaboration is also a big challenge in distributed computing. The proposed system will support collaborative development work in the Internet environment by using multi-version currency control (MVCC) technique in the project, we will discuss more about the advantage of MVCC later.

1.3 Objectives

The projects will focus on developing a distributed development environment for Java Application. The main goal is a network base, dynamic and loosely coupling development environment. Which contains the following feature:

1.3.1 Infrastructure implementation

The project will make use of some open source development environment component such as a UML editor, text editor and compiler to migrate into a set of jini service. Then, we will build some infrastructure level components so that the system will be easy to extend and plug other component dynamically (i.e.: without restart/reload). The next step is implement a generic user interface framework and a human searching component framework. Our final objective is building a framework for developer easy to separate the model and view. As well as users can easy to find and use the tools. And the model stays at the network and shareable to the others.

1.3.2 Collaboration support on development tools

Collaboration support is an important feature for multi-user application. A standalone UML editor will be modified to become a collaborative multi-user application, then try to develop a generic collaboration framework base on this. Versioning technique will be used.

From database systems experience, locks are the slower mechanism used to maintain concurrency control and data consistency. Multi-version model become a more popular technique in implementing database concurrency control. For example, in PostgreSQL, a version is like a snapshot of the data at a distinct point in time. The current version of the data appears whenever users query a table. Naturally, a new version appears if they run the same query again on the table and any data has changed. Such changes happen in a database through UPDATE, INSERT, or DELETE statements. Thus, performance is much better by eliminating the unnecessary waiting in locking.

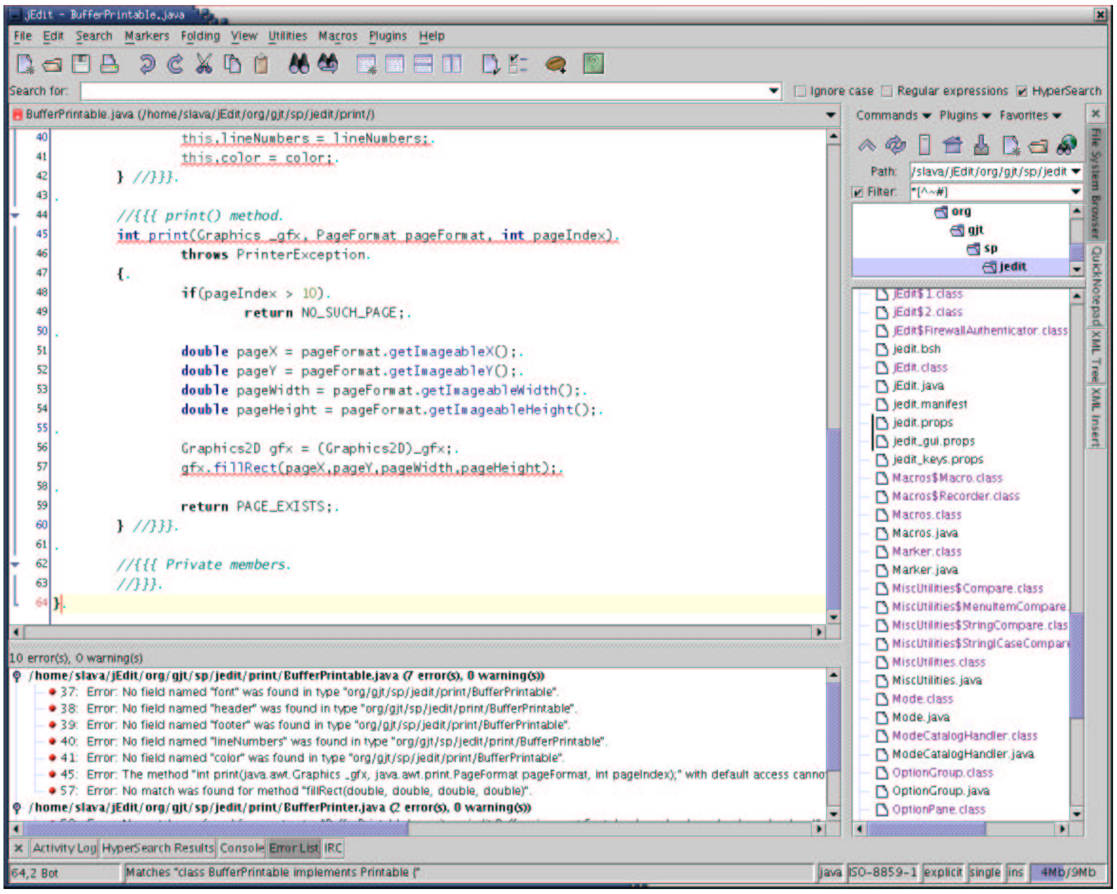
My group consist of 2 member, I will focus on the collaboration support and my partner will focus on the infrastructure implementation.

2. Literature reviews

Before the system design and implementation, we have reviewed some other related works and projects. Here is the projects and approaches we have reviewed and considered.

2.1 Revision of related works

One approach is build the project from scratch, with reference from existing developments environment. The code of some existing opensource IDE like netbeans[6] and jedit[7] are reviewed. Then base on this to evaluate if we can build similar but smaller project. However there are 2 problems. It is found that if the project develop from scratch, the project is probably a duplication of other development environment. Besides, there are so many basic components need to implement, there are not enough time to do this.



```
            this.lineNumbers = lineNumbers;
            this.color = color;
      } //}}}

      //{{{ print() method
      int print(Graphics _gfx, PageFormat pageFormat, int pageIndex)
            throws PrinterException
      {
            if(pageIndex > 10)
                  return NO_SUCH_PAGE;

            double pageX = pageFormat.getImageableX();
            double pageY = pageFormat.getImageableY();
            double pageWidth = pageFormat.getImageableWidth();
            double pageHeight = pageFormat.getImageableHeight();

            Graphics2D gfx = (Graphics2D)_gfx;
            gfx.fillRect(pageX,pageY,pageWidth,pageHeight);

            return PAGE_EXISTS;
      } //}}}

      //{{{ Private members
      //}}}
}
```

```
10 error(s), 0 warning(s)
/home/slava/jEdit/org/gjt/sp/jedit/print/BufferPrintable.java (7 error(s), 0 warning(s))
    37: Error: No field named "font" was found in type "org/gjt/sp/jedit/print/BufferPrintable".
    38: Error: No field named "header" was found in type "org/gjt/sp/jedit/print/BufferPrintable".
    39: Error: No field named "footer" was found in type "org/gjt/sp/jedit/print/BufferPrintable".
    40: Error: No field named "lineNumbers" was found in type "org/gjt/sp/jedit/print/BufferPrintable".
    41: Error: No field named "color" was found in type "org/gjt/sp/jedit/print/BufferPrintable".
    45: Error: The method "int print(java.awt.Graphics _gfx, java.awt.print.PageFormat pageFormat, int pageIndex);" with default access canno
    57: Error: No match was found for method "fillRect(double, double, double, double)".
/home/slava/jEdit/org/gjt/sp/jedit/print/BufferPrinter.java (2 error(s), 0 warning(s))
```

```
Matches "class BufferPrintable implements Printable {"
```

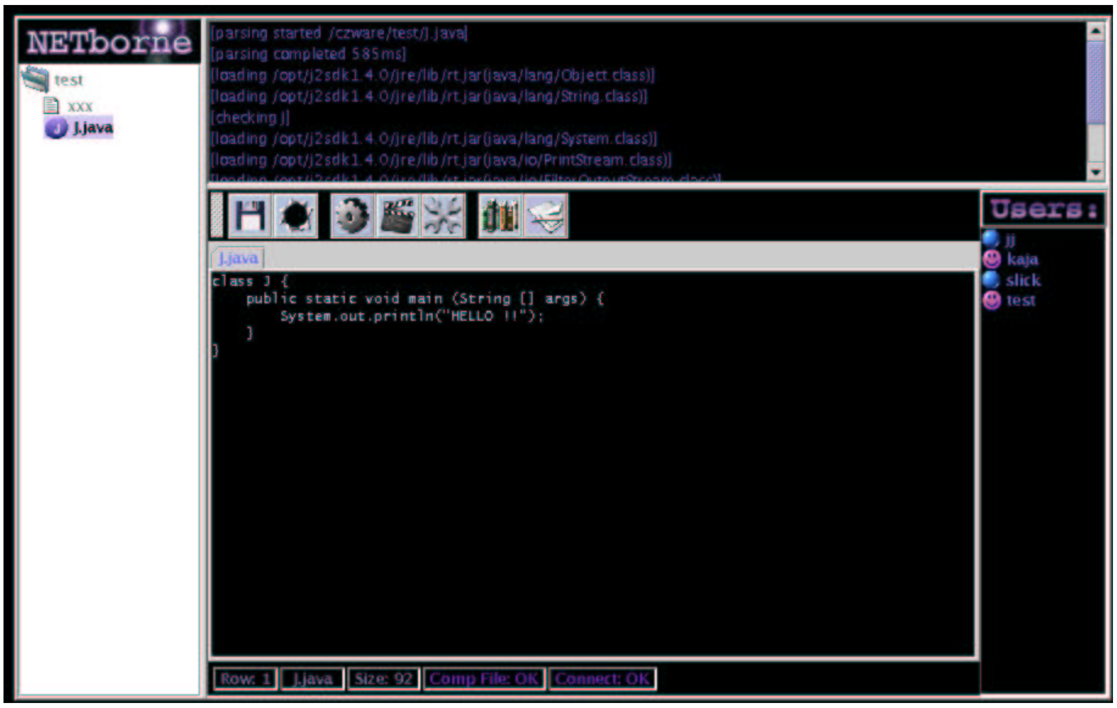
The screen shot of Jedit in Linux, a typical IDE extended from an editor.

Besides, most of the exist development environment are not accessible from the network. i.e.: Most of the components are bounded to the development environments statically, and the development environment is not able to run without any of the components. The user need to download and install most of the component to the client machine. But the proposed project is highly distributed that no similar implementation can be a reference. Thus this approach is rejected.

2.2 Relevent previous projects

Another approach is to develop the project base on some existing project. There are some opensource program are reviewed, like NETborne[8]. NETborne is an applet IDE using client-server architecture, which is better fit in our project, and there are some nice features. However, after investigating more about this tools, some problem are found:

- The author of this program need us to share the our result to him and he does not have licensing his program yet. So there is potential licensing issue of using this tool.
- Most features of NETborne is not related to the proposed system, like applet-servlet communication protocol of IDE. Other than that, some design problem are found (no package, duplication between disk storage and database storage without sync.)
- jini do not use traditional client and server architecture. The traditional client-server architecture is not suitable for the proposed projects, we will discuss the advantages of jini in the later section.



```
[parsing started /czware/test/J.java]
[parsing completed 585ms]
[loading /opt/j2sdk1.4.0/jre/lib/rt.jar(java/lang/Object.class)]
[loading /opt/j2sdk1.4.0/jre/lib/rt.jar(java/lang/String.class)]
[checking J]
[loading /opt/j2sdk1.4.0/jre/lib/rt.jar(java/lang/System.class)]
[loading /opt/j2sdk1.4.0/jre/lib/rt.jar(java/io/PrintStream.class)]
```

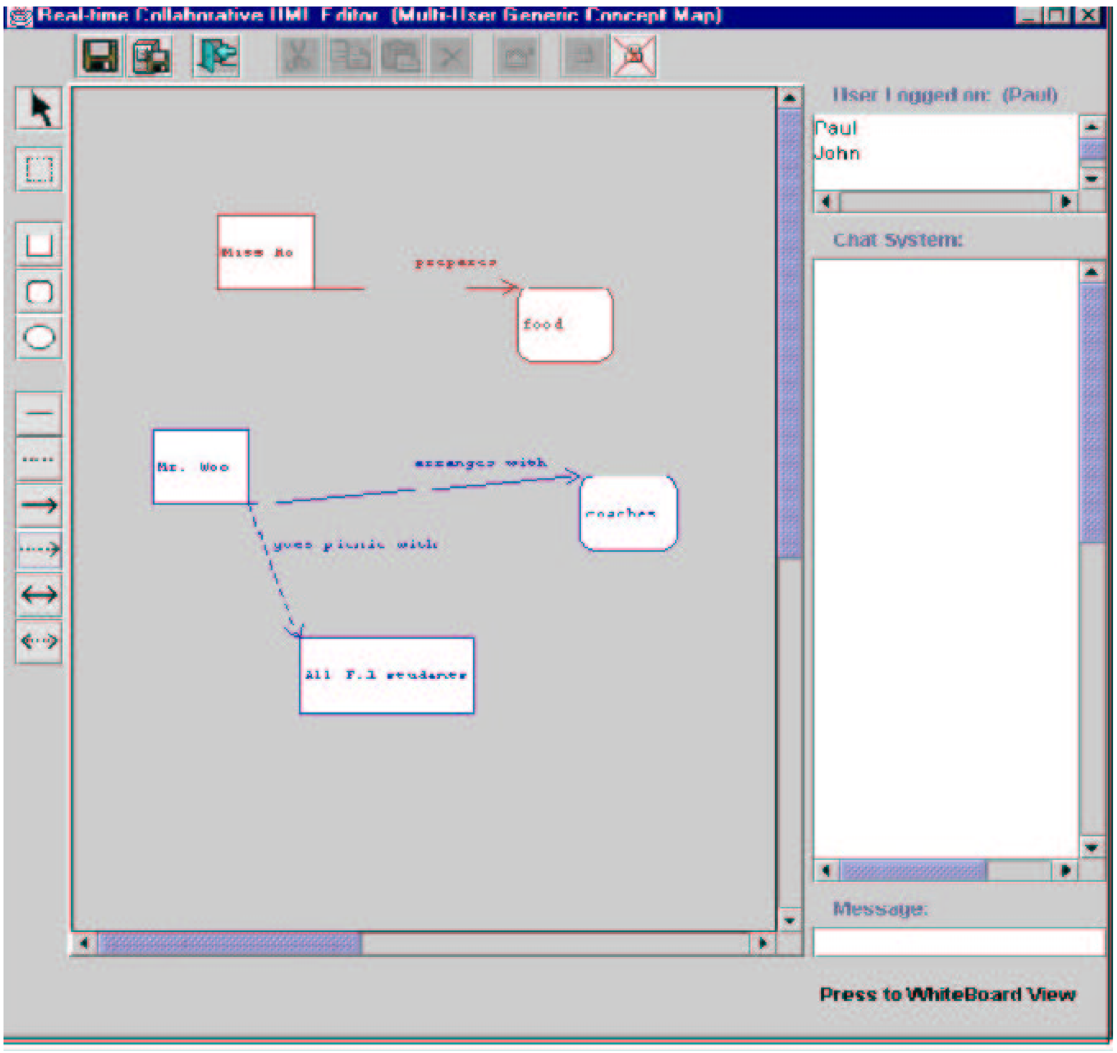
```
class J {
    public static void main (String [] args) {
        System.out.println("HELLO !!");
    }
}
```

Row: 1 | J.java | Size: 92 | Comp File: OK | Connect: OK

The screenshot of NETborne, an applet base IDE.

Other than public available opensource program, Some past final year projects are reviewed. For example, a collaborative uml editor[9]. This software allow multiple users edit a single

UML class diagram in same time. It is a very good project, however, it suffer from some limitation that we can make use of this.



The screenshot of the collaboration UML editor.

- The uml editor cannot extend without a heavy work, mostly because the source is highly coupling which it is hard to identify a code to edit.
- The system also use client and server architecture using RMI method call. Which is hard to port to jini.
- The system use locking as collaboration methodology but we would like to use versioning. Changing locking base communication code to versioning base communication code is another problem.

2.3 Proposed approach

After reviewing several difference tools and technology, the final decision is extending ArgoUML[22] to have versioning supports. The reason are follow:

1) The tools/technique must be ready for distributing environment, as computer network because larger and larger scale that we should build our system ready for distributing environment for the future, and ArgoUML have highly modularized structure that enable me to port it as a distributed tools
2) ArgoUML is an opensource project have a strong community supports, this is a high volume list. We believe that we can benefit from this.
3) This is the most advance open source UML editor available, receive good comment from the press [10].
4) This tools save the UML in a XML format call XMI which save much of our work for save our work.
5) This tools provide fully support in UML editing, so the project can concentrate on the collaborative support without wasting time of UML editing, which is the core of our projects. Hopefully we can have better collaboration support finally.

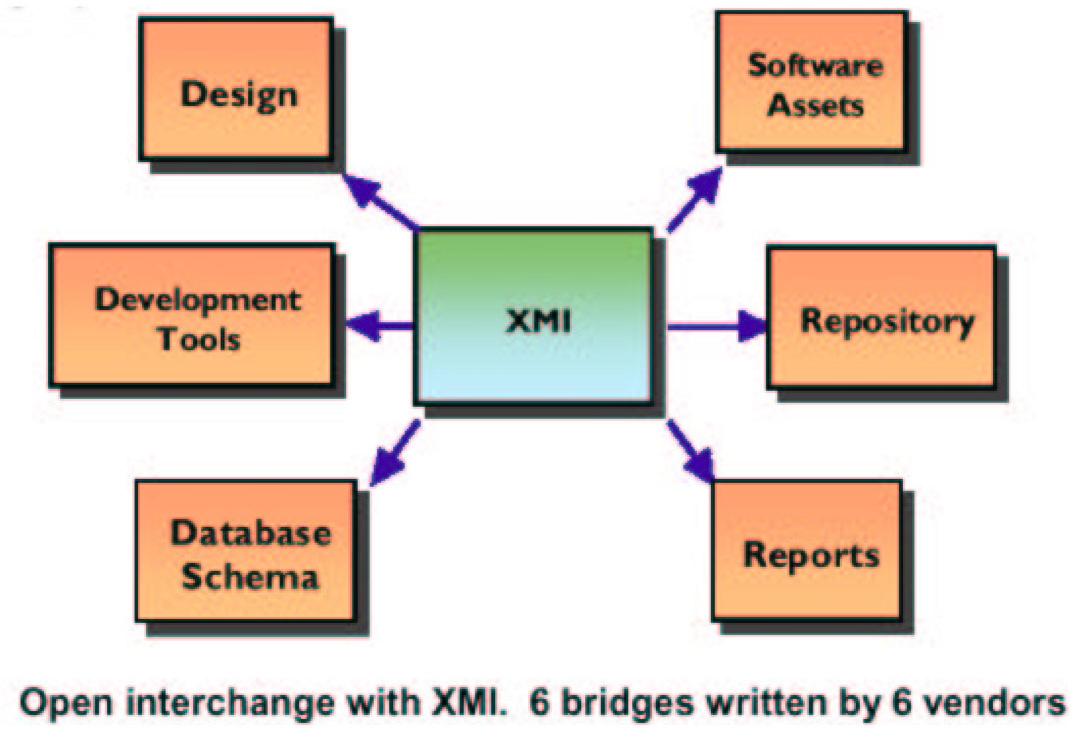
After the implementation, we will discuss the advantages and disadvantages of using ArgoUML in the project.

3. Technology to evaluate

In this project, there are some technology of this topic will be evaluated, we will find out is these technology suitable for future projects. Here is the brief introduction of these technology.

3.1 XML Metadata Interchange(XMI)

XMI is an very important technical specification of UML editing, because OMG, the biggest object oriented standard maker, only recommend XMI to archive UML. The software design diagram can be interchanged between difference modeling tools via XMI. Besides, XMI is that standard propose by OMG, and supported by most UML modeling tools[12].



Open interchanged with XMI

XMI integrates three key industry standards:[36]

1. XML - eXtensible Markup Language, a W3C standard
2. UML - Unified Modeling Language, an OMG modeling standard
3. MOF - Meta Object Facility, an OMG metamodeling and metadata repository standard

3.1.1 Advantages

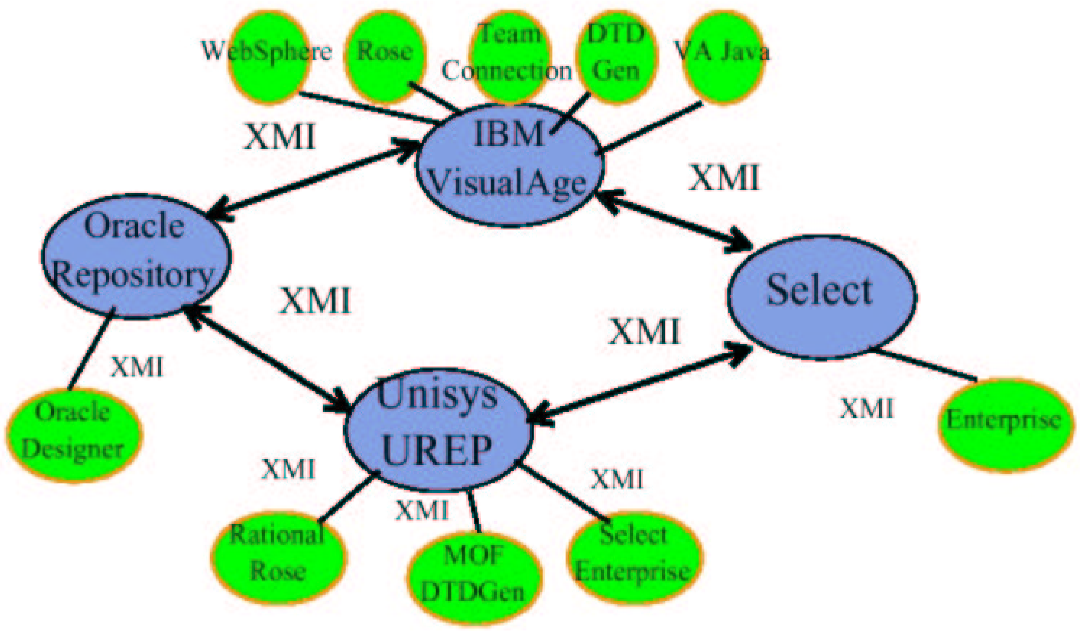
XML format provide the following benefits[15]:

1. Object model archive is hard to manage, because java object archive can only be managed when serialize back to object in memory. That mean object model archive cannot be managed without java.
2. Even after serialize back to object in memory, the structure of that object is unknowns. It need to cast back to exactly type manually, the original type need to be recorded first, in most case, it is easy to make mistake.
3. There are many support tools on XML that help us the manage the data format, like XML data store.
4. There are existing tools that analysis XML. These can be used to analysis XMI design. For example there are some existing tool help us to extract some tag content, which can be used to extract all the comment of the project.
5. XML is a popular technology that most developer familiar with.

When compare with other similar XML format, like UXF[12] and uml-xchange[13]. XMI have the following advantage:

1. XMI is adopted in more applications by more vendor support XMI, like IBM, oracle, togetherJ and ArgoUML, which make our resulting application can communicate with more program. Even UXF author agree that XMI is the mainstream
2. None of the above full support complete UML 1.4 specification, Fully support open standard is important, because this make our application extendable.
3. XMI is readily port to other format through XSLT, project is existed on transferring XMI to HTML[14].

A diagram of how XMI have IBM Visual Age, Unisys UREP and Oracle respostory work together, show the scalability of XMI:



Visual Age, Oracle repository and Unisys UREP work together with XMI

3.1.2 Disadvantages [36]

Nothing is prefect, so as XMI, in fact there are some problem of using XMI:

1. XMI do not contain any information of how the UML diagram display, it only concern the structure of UML diagram, because this is the part that XMI address. However, this bring many problems in real use. For example, if 2 UML modeling tools use difference way to store the presentation, the design diagram cannot share between these 2 tools.
2. It is too complicated to learn, the specification of XMI is very long, so this may not be a good choice of final year project. May be some smaller specification like UXF is better.
3. Unlike Java, Rational does not release a standard validation test suit to valid XMI. In fact, difference vendor have difference validation tools, IBM have it own XMI toolkits, Rational Rose have it own XMI plugin. This may cause possible incompatible.

3.1.3 Summary

In this project, other than using ArgoUML, the system will also using NSUML package, an opensource UML metamodel. This package fully support the important software design meta method standard, e.g.: MOF, XMI and JMI. We will explain this package more in later section. After the implementation of the project, we will make a conclusion of is it good to use XMI in the under-graduate projects.

3.2 Evaluation of different collaborative algorithms

Some one will refer locking to pessimistic approaches and versioning to optimistic approaches. Here is a reference that explain the difference very detail:

3.2.1 The different consequences of using locking or versioning[1]

The locking approach is good for keeping model valid, but can be perform very bad for concurrency. Especially, it guarantees that that no 'lost updates' occur, because the editing model will be locked to prevent other to update. However, the poor performance of concurrency is a side effect.

The can be a serious problem because it is possible that an user hold the lock of the object for a unexpected long time. There are many reasons for this, one is the user really need to edit that object for a long time. However, the really problem is this can be a results of common unexpected situation, e.g.: The user editing a model at first, but when the office time finish, it is very easy for the developer to forget to release the lock before leaving office; another problem is if the client have been killed for some reasons, the system many hold the lock forever, may be the server need to restarted to fix this.

Of course, these can be solved by setting a timeout of locking, however, the other potential problem is dead lock, which is a very common problem in concurrency system, especially when doing a batch job. I do not plan to explain dead lock in this text very detail, there are many papers to discuss this. In fact, the past UML modeling tool[9] is a very good reference.

So it is very clear that locking may have many problems when the system work in a big scale system.

The versioning approach aim to solve the problem of locking, because it do not need to lock any object. However, the problem is we need more complex algorithm to keep the model valid. The "versioning" definition usually says that expectations of update clashes are rare, but in fact it is a normal occurrences in a high access system. The basics are that any changes between time of access and time of update must be detected and taken into account. This is often done by comparing timestamps, but one must be sure that the timestamp is always changed for an update/commit. Will discuss the versioning algorithm more later.

3.2.2 Example to show the advantage of versioning enable system[2]

Quoted from reference, it is a simple example of selecting data from one table shows the difference between traditional row-level locking and multiple versioning concurrency control (MVCC) powered database management system:

SELECT headlines FROM news_items

This statement reads data from a table called news_items and displays all the rows in the column called headlines. For data systems that use row-level locking, the SELECT statement will not succeed and will have to wait if another user is concurrently inserting (INSERT) or updating (UPDATE) data in the table news items. The transaction that modifies the data holds a lock on the row(s) and therefore all rows from the table cannot be displayed. Users who have encountered frequent locks when trying to read data know the frustration this locking scheme can cause, forcing users to wait until the lock releases.

In MVCC enable DBMS, however, users can always view the news_items table. There is no need to wait for a lock to be released, even if multiple users are inserting and updating data in the table. When a user issues the SELECT query, PostgreSQL displays a snapshot,or version, of all the data that was committed before the query began. Any data updates or inserts that are part of open transactions or were committed after the query began will not be displayed.

3.2.3 Summary

As collaborative UML editor using locking algorithm already done in past final year project, this system will focus on versioning support of UML editing. After the implementation of the project, we will make a conclusion of is it good to use versioning algorithm in the under-graduate projects.

3.3 Jini network technology

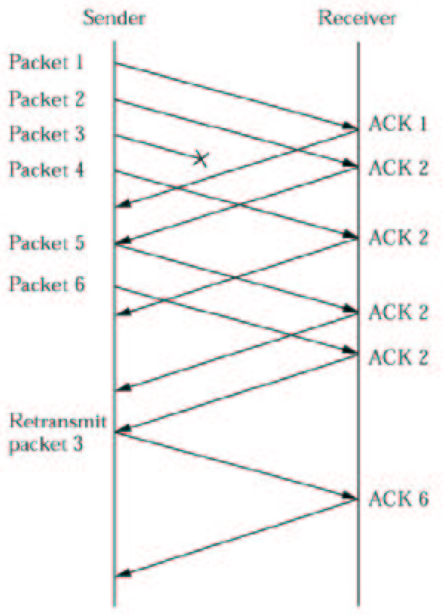
3.3.1 Advantages of using Jini

For more information about how Jini help in the development environment, please refer to my partner's report[38]. Here only the benefit that jini bring to use in the collaborative system.

3.3.1.1 The problems of existing network architecture

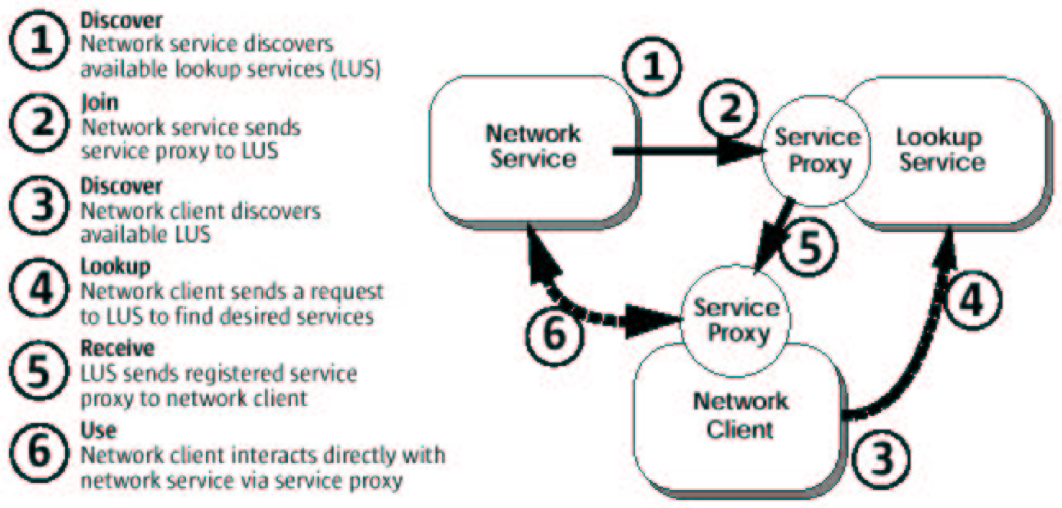
The existing client server architecture is not suitable in large scale system

1. When the system require to model a one to many or many to many communication, client-server architecture require the programmer group many one to one connections to model the system. For example, if we want to model a system have one server and many clients, we need to keep a list of socket connection in the server. If we need to model and many to many connection system, we need to have a groups of one to many connection to handle them. For some complex network architecture, it is hard to model and hard for the maintainer to understand the system.
2. Lack of persistence support. If a client send something to the server, the server need to handle it now. If the server are busy, the client need to wait for the server. A more worse case is the server will deny the request of the clients.
3. Possible for dead-lock. Refer back to 1), if the server maintaining a list of client to communicate, because there may be 2 client call the server in same time. The server need to synchronize some methods, the synchronization block may have dead-lock or other performance problems occur.



3.3.1.2 How jini helps in these problems

Here is the diagram show how jini work.


1 Discover
Network service discovers available lookup services (LUS)
2 Join
Network service sends service proxy to LUS
3 Discover
Network client discovers available LUS
4 Lookup
Network client sends a request to LUS to find desired services
5 Receive
LUS sends registered service proxy to network client
6 Use
Network client interacts directly with network service via service proxy
Network Service
Service Proxy
Lookup Service
Service Proxy
Network Client

The flow diagram of jini architecture

1. Jini have a very clear way to model network architecture. In jini, everything is service, the user do not need to build any network connections in order to communicate with other network component. Everything now are managed by the lookup service. If you want to communicate with other network component, you just need to find it from lockup service, then get the proxy of the service and work with it. It work mostly same with the local component, so we are more easy to model and understand a large system.
2. Jini provide default persistence support in the network, javaspace. The server do not need to response to request immediately, the client is not need to wait for server response. In java space, the clients can put an item to space and let the servers handle any time he like. If the server do not manage that item for a long time, jini leasing service will free the resource automatically.
3. Jini can prevent dead-lock problem in many way because of the help of javaspace. In response to the last example, when 2 client try to communicate with the server in same time, the servers do not need to synchronize any method. Because the clients do not communication directly to the server, the clients just put the items into javaspace and the server just pick them up and handle them and put it back to javaspace. You can think now there are no server, everything is clients that work with the persistence javaspace. This architecture simplify many programming problems.

In fact, the boardcast of the versioning engine mentioned in later section will make use of this architecture, we will show it later.

3.3.2 Summary

Jini sound great, it seen to be a very robust network infrastructure that solve most networking problem, but is it really that perfect? The system will build on top of jini, after the implementation, we will find out the good and the bad of using Jini, and is it good to use in an under-graduate project.

4. Proposed collaboration algorithm

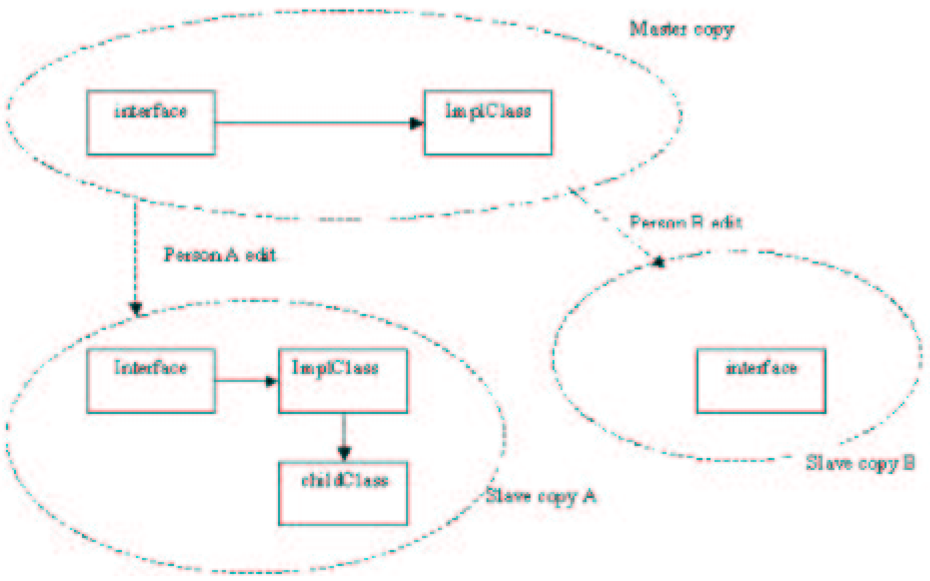
After designing the approach of this project, next step is to figure out a algorithm of versioning collaboration. The first collaborative algorithm was inspired from “Efficient Version Model of Software Diagrams”[3] and “A flexible object merging framework”[4].

4.1 Concept of the proposed algorithm

“An Efficient Version Model of Software Diagrams” have present an idea of how to model the diagrams and the modifications of the diagrams in Mathematics way, where “A flexible object merging framework” introduce when will conflict arise and how to resolve in simple data.

Summarize the above idea from the papers and get the following Versioning algorithm:

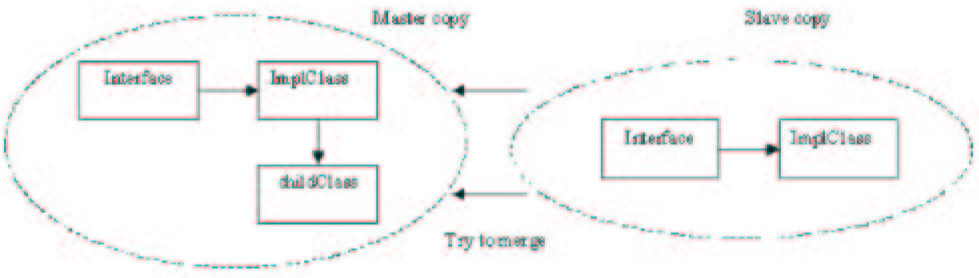
1. Model UML and modification into some data structure according to [3]
2. The system will consist of a versioning engine and a master copy that is always valid.
3. The difference clients work on same UML diagram will merge the data structure of it with the master copy for every operation. The system will identify if there are any conflict before merge 2 difference diagram.
4. We find that conflict is actually some operation break the **dependence[4]** between item, e.g.: *class I extend class II, if slave copy have an action to delete class II, then the inheritance relationship between class I and class II is broken.*
5. Consider the following example, there are 2 clients, A and B modify some class diagram. Originally that class diagram consist of an interface and a class implement that interface, ImplClass. Client A try to extend a child class from ImplClass; meanwhile client B try to deletes ImplClass. conflict arise in this case because A copy and B copy cannot merge together.



An example of conflict arises

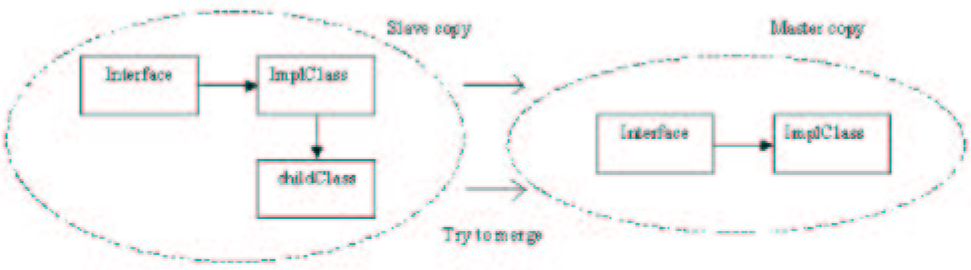
If there is not conflict, the system will merge the UML diagram of master copy and client copy (slave copy), then update both copy to the latest version. If conflict find the system will take the do take some action that prevent the master model being corrupted. The conflict can be resolved into 2 category:

1) *Slave copy deletes some entities required for master copy – We can just reject the client modification and as the client to restore to the originally state.*



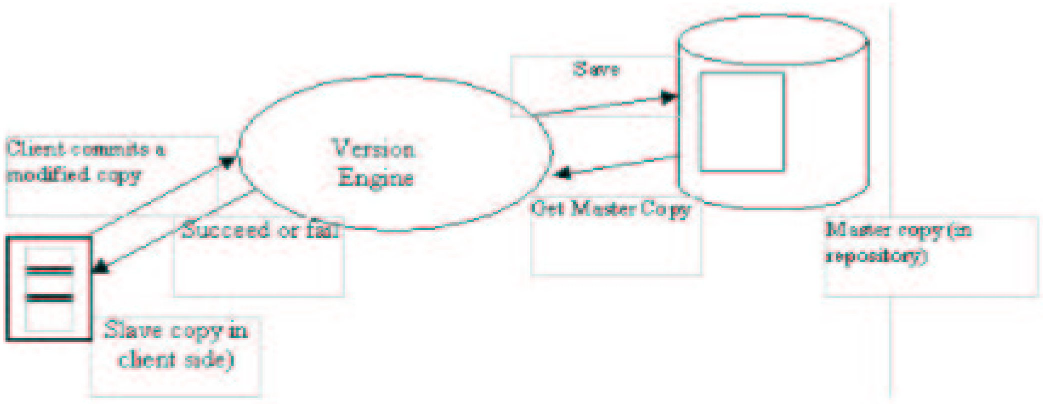
Case 1 of conflict: slave copy delete some required for master copy

2) *Master copy deletes some entities required for slave copy – In the case we need to recover the component that client need and then merge 2 diagram.*



Case 2 of conflict: master copy delete some required for slave copy

The big picture the of the system is follow:


Save
Version Engine
Client commits a modified copy
Succeed or fail
Get Master Copy
Master copy (in repository)
Slave copy in client side)

The system architecture of first version of versioning engine

The above algorithm can be extended from object-object relationship to object-features relationship and features, because there are dependence relationship of object and features. e.g.: every operations depend on some class, we can forbid or recover delete an object that have one or more operation depend on it; we can also forbid or recover a operation that have any other operation inheritance relationship with this. We can do all these with the same algorithm as stated above.

4.2 Detailed implementation plan

1. Use just one model.
2. One server that owns the model and diagrams, the master copy. The server consist of a versioning engine that guarantees it is valid at all times.
3. All clients work against the same model and diagrams, but having their own copy on the UML modeling tool instance.
4. Clients do modifications of the model or diagrams and show views of the model. A interface needs to be defined that can get all needed information from the model and diagrams for the client to merge it copy to the master copy. The interface must also contains methods to modify the master copy.
5. Every modifications initiated by the client will merge with the master model. The modification can fail with errors like: Operation cannot be removed - it does not exist (i.e. some other client just removed it), or Operation cannot be added - Class does not exist (i.e. some other client just removed it). All these kinds of errors must be defined in the versioning engine. All errors will have 2 handler, because there are 2 kind of conflict, slave delete master required and master delete slave required.
6. The default case of error handling is reject the client of the client. If the versioning engine encounter some cases cannot handles. It will reply something like: "You cannot do that because somebody else just did something that made your modification impossible." Where something is the most recent modification, and somebody is the client name do that modification.
7. If there is no conflict, master copy will merge with the change of client copy, then the clients UML modeling tool will get a copy of master copy and replace the local copy.

4.3 Other references

Here is some other paper about this topic reviewed for future reference, but they are not used because:

1. They are not easy to merge with the algorithm discussed.
2. The algorithm is very complex and not suitable for real time collaboration.

Abstract State Machines:UML State Machines[16]

ASMs are used to give semantics for UML state machines, as a basis for constructing an automated tool for verifying properties of UML state machines.

UMLAUT: an Extendibles UML Transformation Framework[17]

UML Transformation Framework allowing complex manipulations to be applied to a UML model

Modeling Versions in Collaborative Work[18]

Discuss a basic version model a domain model capturing the idea of a version and the relationships between versions

Version control for asynchronous group work.[19]

This paper looks at the issue of version control comparing single and multiple user situations. The aim is to focus on requirements for version control that will assist asynchronous distributed group writing.

vUML[20]

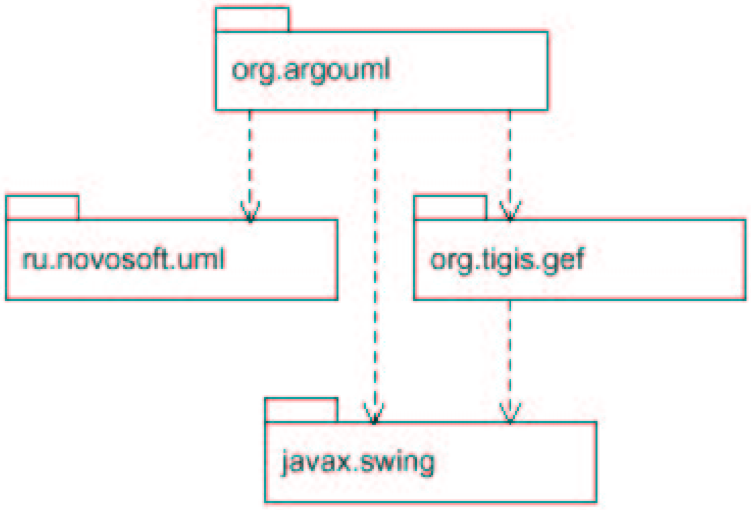
vUML is a tool that automatically verifies UML models, UML validation is very important in our project so I take reference from this. However, the mechanism of this papers is too complicated that do not suitable for use.

5. Implementation

Before implementation, we need to understand the flow and structure of the UML modeling tools, ArgoUML, first. Here is a brief overview of the internal structure of ArgoUML.
Besides, ArgoUML is a highly modularized UML editor. Personally speaking, I learn many on studying this system, so I will spend a part in the report to explain the structure of ArgoUML, as well as the core components of this ArgoUML.

5.1 The architecture of ArgoUML [37]

- GEF - Graph editing framework, model the components in nodes and edges and let user edit.
- NSUML - UML meta-model implementation, contain the underlying UML structure and xmi converter
- Swing - build the GUI component



The architecture of ArgoUML

Screenshot

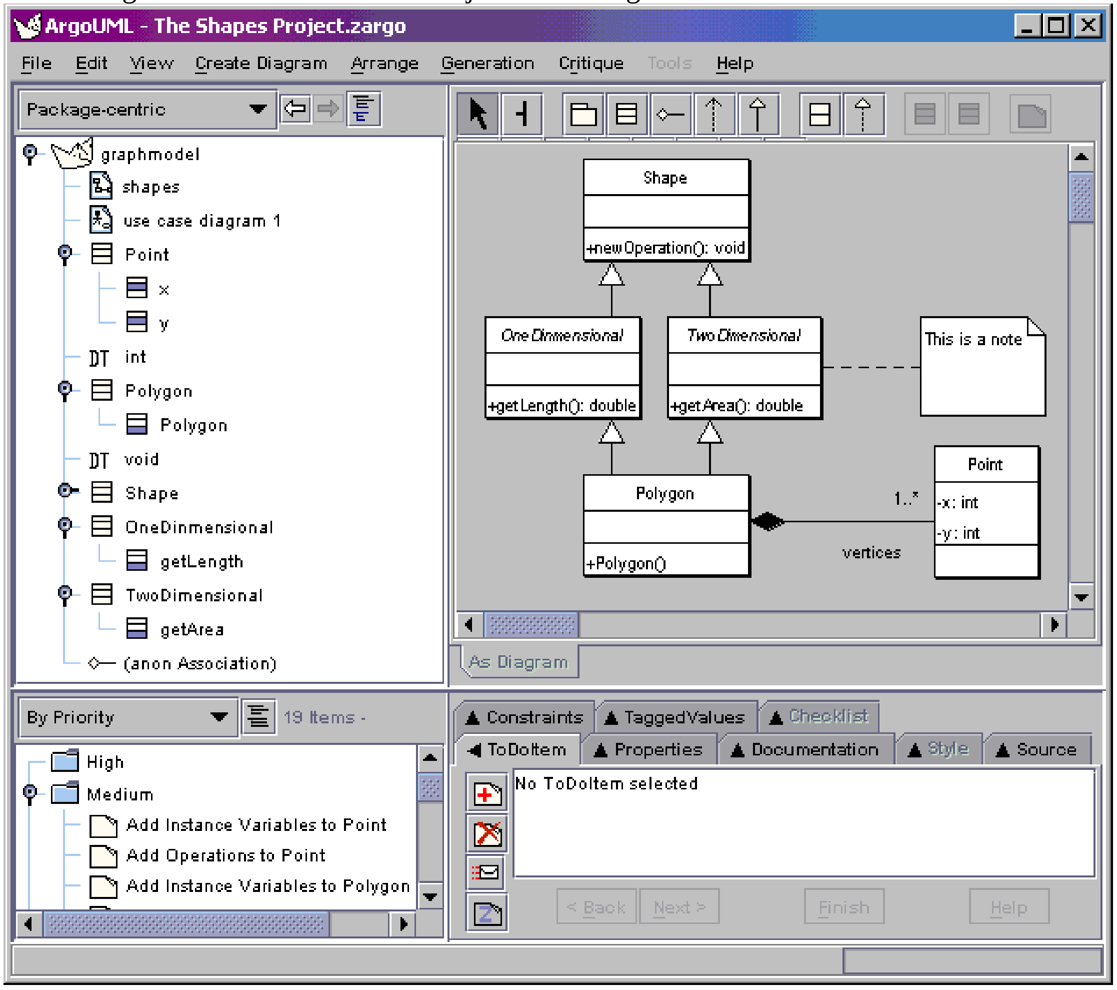
Here is a screen-shot of ArgoUML.

Top left: a hierarchical view of the current project file.

Upper right: editor(s) for the selected part of the project, in this case a class diagram.

Bottom left: the designer's "to do" list.

Bottom right: details of the selected object in the diagram or the selected "to do" item.



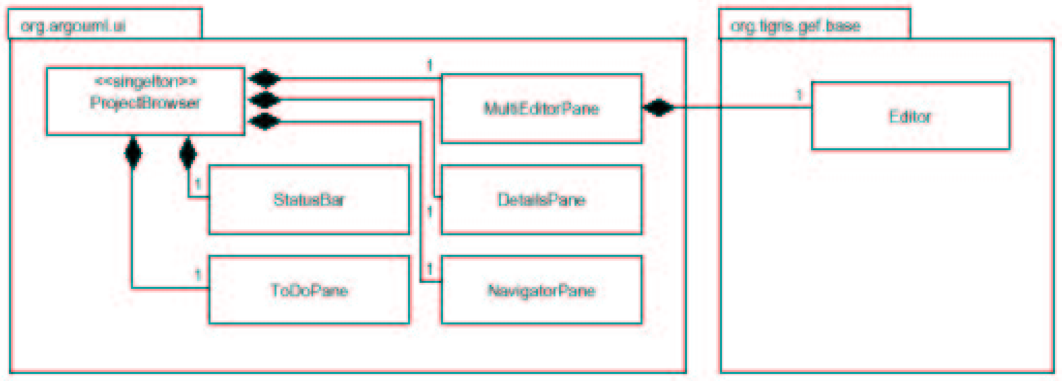
Screenshot of ArgoUML

5.1.1 Detailed description

5.1.1.1 GEF: [21]

- A simple, concrete design that makes the framework easy to understand and extend.
- Node-Port-Edge graph model that is powerful enough for the vast majority of connected graph applications.
- Model-View-Controller design based on the Swing Java UI library makes GEF able to act as a UI to existing data structures, and also minimizing learning time for developers familiar with Swing.
- High-quality user interactions for moving, resizing, reshaping, etc. GEF also supports several novel interactions such as the broom alignment tool and section-action-buttons.
- Generic properties sheet based on JavaBeans introspection.
- XML-based file formats based on the PGML standard (soon to support SVG).

Here is a diagram to show how ArgoUML relate to GEF:

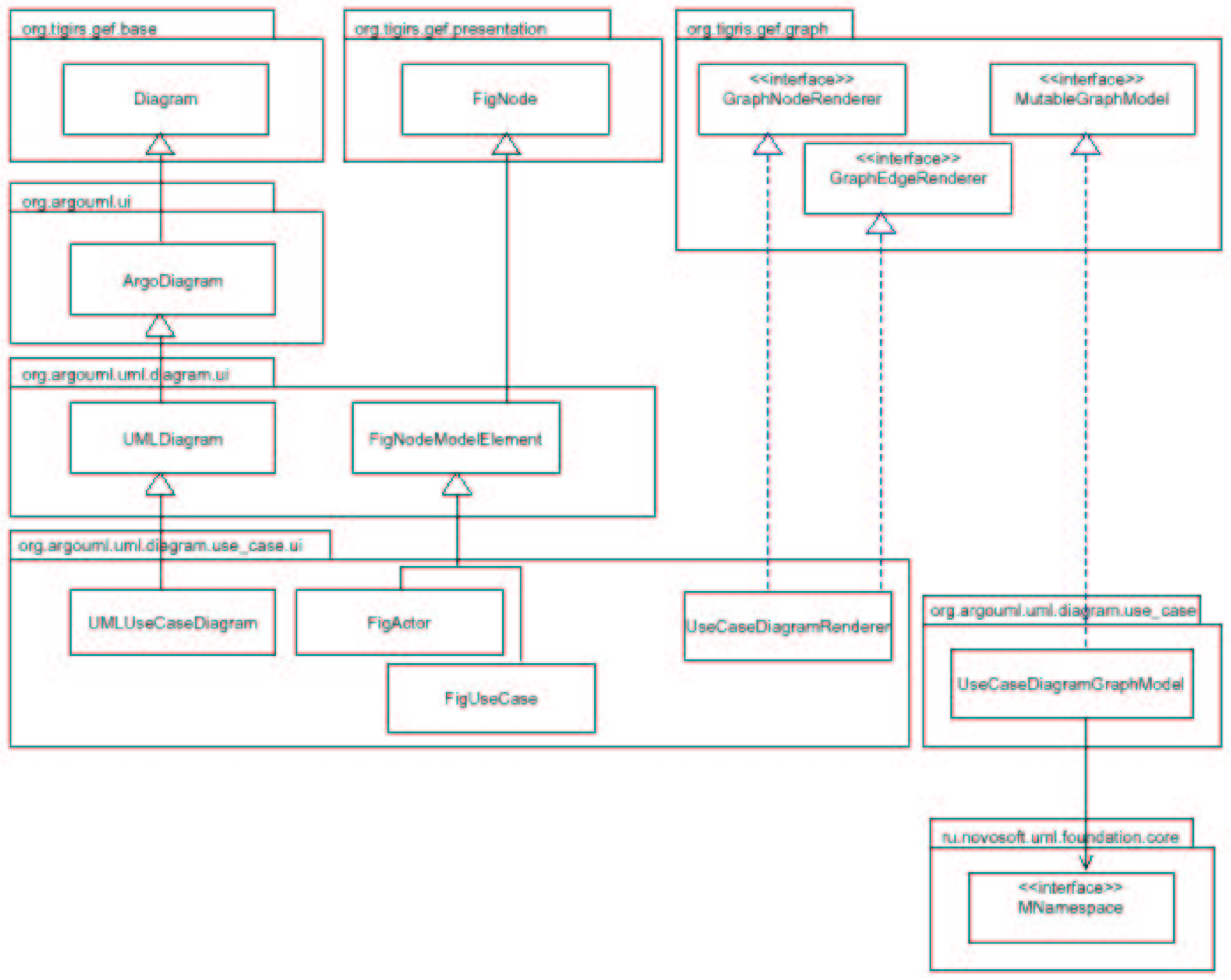


The relation of ArgoUML and GEF

5.1.1.2 NSUML: [23]

NSUML(Novosoft metadata framework) is based on JMI specification and generated classes that are required by JMI specification and also provides additional services like event notification, undo/redo support, XMI support. NSMDF is local in-memory implementation.

This package also provide code generated from UML 1.4 metamodel. Which could be used for constructing applications based on UML 1.4.


org.tigirs.gef.base
Diagram
org.tigirs.gef.presentation
FigNode
org.tigris.gef.graph
<<interface>> GraphNodeRenderer
<<interface>> MutableGraphModel
<<interface>> GraphEdgeRenderer
org.argouml.ui
ArgoDiagram
org.argouml.uml.diagram.ui
UMLDiagram
FigNodeModelElement
org.argouml.uml.diagram.use_case.ui
UMLUseCaseDiagram
FigActor
FigUseCase
UseCaseDiagramRenderer
org.argouml.uml.diagram.use_case
UseCaseDiagramGraphModel
ru.novosoft.uml.foundation.core
<<interface>> MNamespace

How GEF, NSUML and ArgoUML work together

5.1.1.3 JMI (Java Metadata Interface) [24]:

The JavaTM Metadata Interface (JMI) Specificiation implements a dynamic, platform-neutral infrastructure that enables the creation, storage, access, discovery, and exchange of metadata. JMI is based on the Meta Object Facility (MOF) specification from the Object Management Group (OMG), an industry-endorsed standard for metadata management. The MOF standard consists of a base UML model and a set of interface definition language (IDL) interfaces. The MOF specification provides a programming mechanism that allows applications to query a metamodel at run time to determine the structure and semantics of the modeled system. JMI is a Java technology mapping of the MOF IDL interfaces that will allow Java components and applications to access and manipulate metadata. Using JMI, applications and tools which specify their metamodels using MOF-compliant UML can have the Java interfaces to the models automatically generated. Further, metamodel and metadata interchange via XML is also automatically enabled by JMI's use of the XML Metadata Interchange (XMI) specification.

Advantages of JMI

JMI will increase the adoption of standards-based metadata and accelerate the creation of applications and solutions in which there are no barriers to information exchange.

5.1.2 The functions of the main packages

- Application - Application launcher, plugin helper classes and security codes.
- Cognitive - still unknown, but should not relate to our project
- Kernel - kernel components like project management and editing history management
- Language - Support code for difference computer language
- Ocl - code generation from uml supports
- Pattern - reserve for future support of design pattern
- Persistence - reserve for future support of storage and restore UML to DB
- Ui - Swing UI components
- Uml - UML manipulation components
- Util - Misc utils like logging and config loader
- Xml - Xml manipulation codes

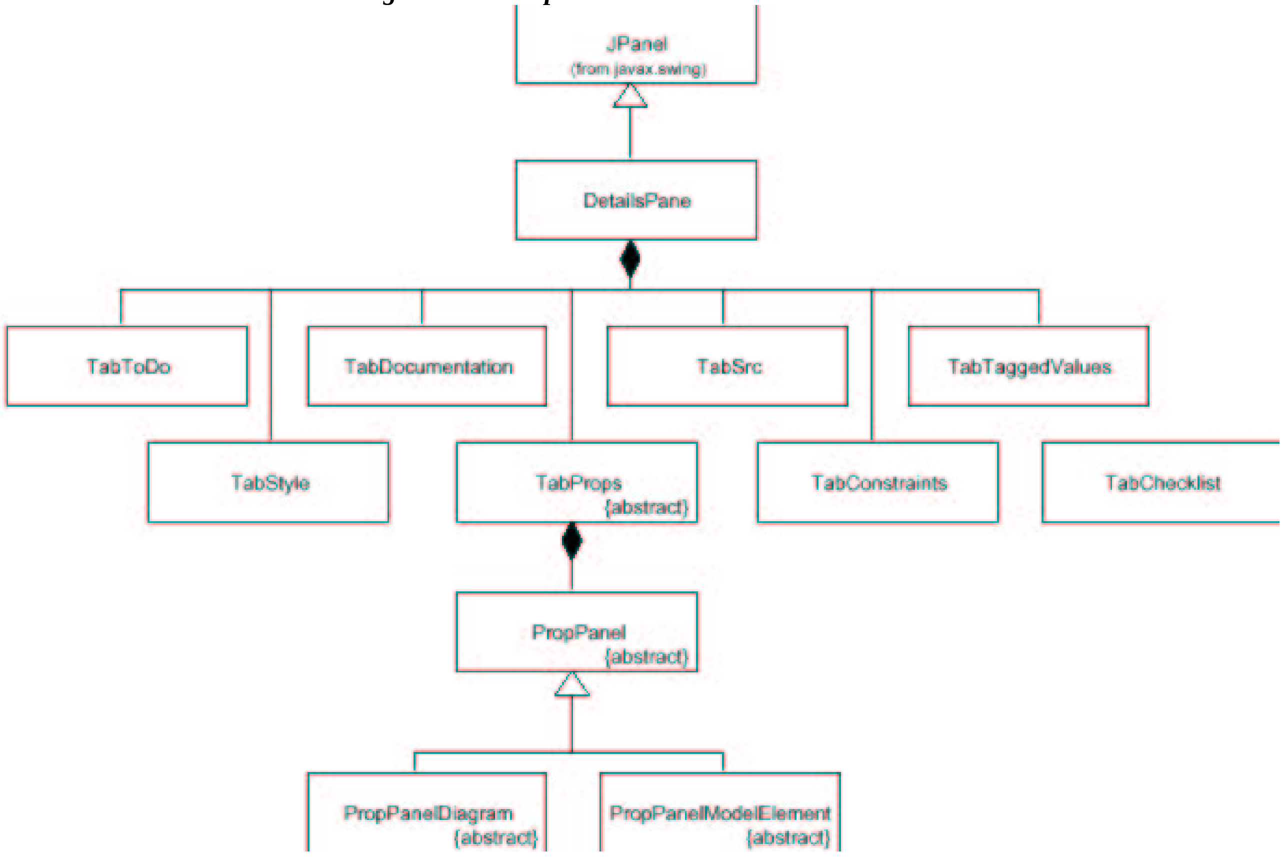
The packages we conserve most are Ui and Uml, as the Ui responsible for the for front-end swing event and swing action handling, as well as the drawing code; and Uml responsible for the backend Uml modeling code., which need to be shared with other machine.

The detail of UI of ArgoUML

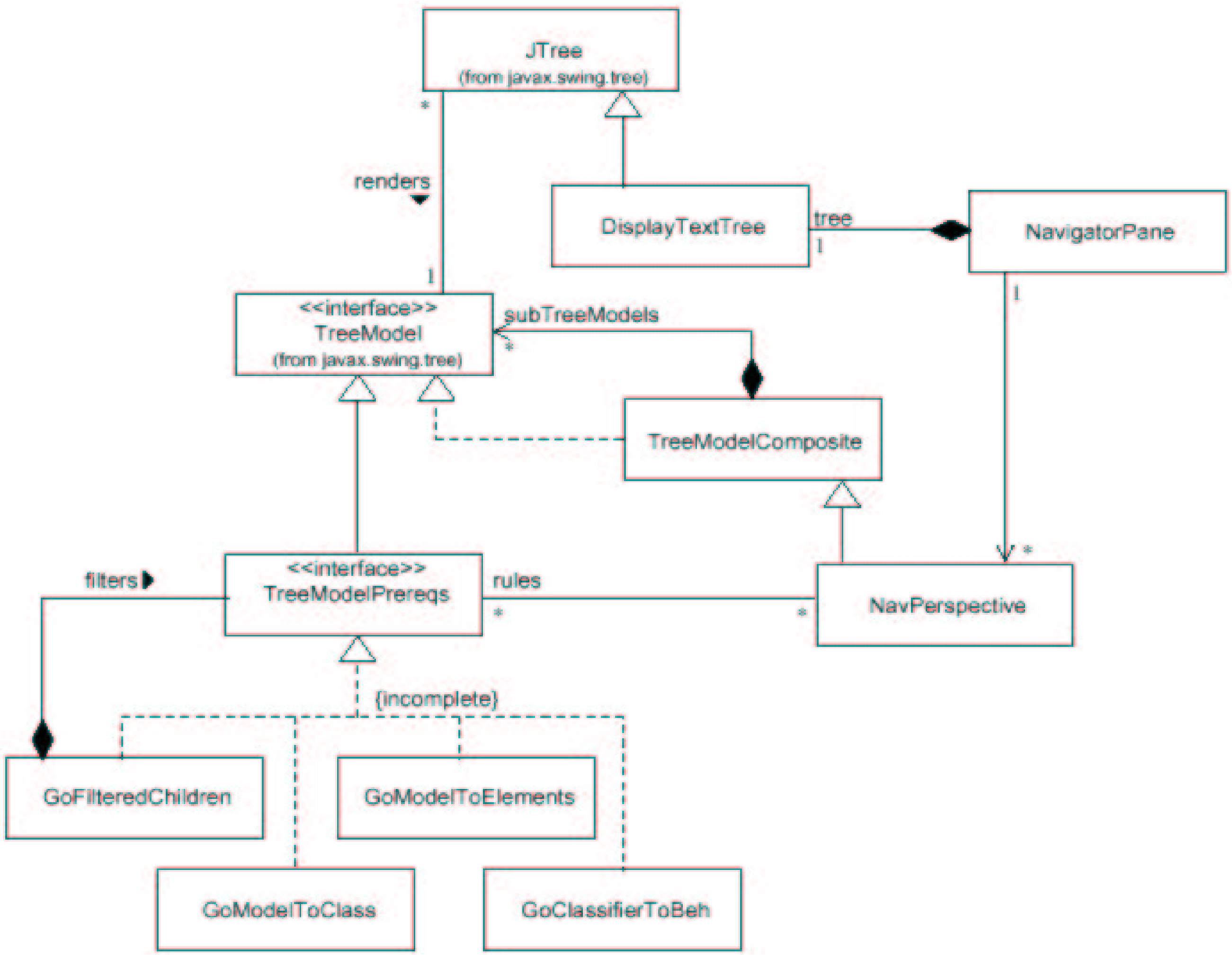
The UI of ArgoUML is very complex, but only 2 is highly possible related to our work, one is detail plane and one is tree plane, the reason will discuss later.

Here is the structure of detail plane, it is important because the project may need to modify the detail plane to add new tab related to versioning for ArgoUML.

ArgoUML detail plane classes.



And here is the tree plane, this is important because the project may need to get the current diagrams structure from the tree plane items.

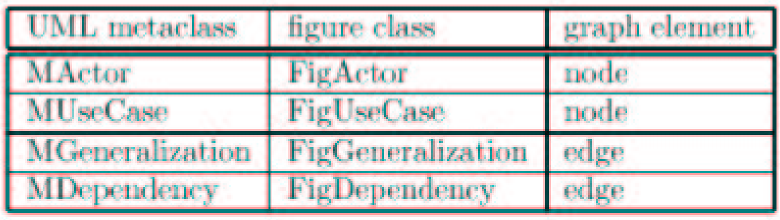


ArgoUML tree plane classes.

5.1.3 ArgoUML data structure

GEF

- library provide tools and data let user edit the diagram.
- handle the interaction from screen to model
- XML-based file formats based on the PGML standard

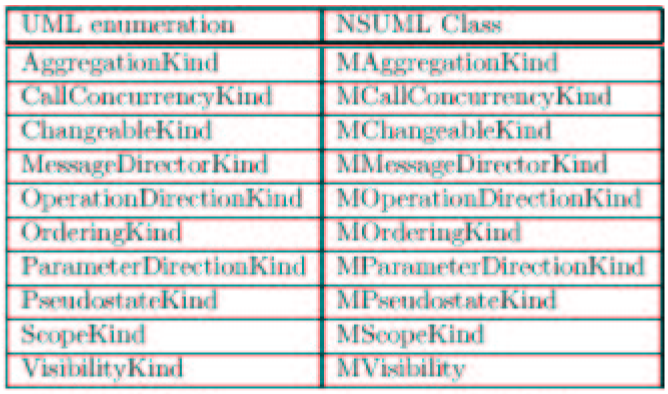


UML metaclass	figure class	graph element
MActor	FigActor	node
MUseCase	FigUseCase	node
MGeneralization	FigGeneralization	edge
MDependency	FigDependency	edge

The relationship of GEF and NSUML is usually one to one.

NSUML

- implementation of complete UML 1.3 physical metamodel,



UML enumeration	NSUML Class
AggregationKind	MAggregationKind
CallConcurrencyKind	MCallConcurrencyKind
ChangeableKind	MChangeableKind
MessageDirectorKind	MMessageDirectorKind
OperationDirectionKind	MOperationDirectionKind
OrderingKind	MOrderingKind
ParameterDirectionKind	MParameterDirectionKind
PseudostateKind	MPseudostateKind
ScopeKind	MScopeKind
VisibilityKind	MVisibility

Table 6.2.: UML enumerations and NSUML classes

- XMI loading/saving.
- NSUML is able to generate events whenever the model is modified.
- Undo/redo support

5.2 Prototype

5.2.1 Overview of the prototype design

After investigating about the data structure of the program, we need to write some prototype. The propose is to test if the structure and data flow work just as expected, and to find out if the algorithm proposed have any problems. The prototype is very simple, it just try to to share the model between 2 instance of ArgoUML.

1. There are few instance of patched ArgoUML running.
2. When there is one model added to one instance. ArgoUML will generate a model object. The patch will capture the model and send it to the server.
3. When the server receive the object of the model. It boardcast the model to all other instance of ArgoUML.
4. When the other client receive the model, it merge the model to the data structure of the diagram editing.
5. There will not be any versioning algorithm implemented in the prototype. So all the delete action are allowed to do, and the model may get into an invalid state.

5.2.2 Problems encountered in the prototype

However, after building the prototype, there are some problems:

1. The code of difference UML diagrams are not in same structure. Difference diagram, say class diagram and deployment diagram, have completely difference architecture. So writing code for one diagram is not able to apply to any other diagram
2. The prototype has not consider the effect of GEF. As all the diagram display to screen through GEF, the prototype expect that if the model of ArgoUML change, the model of GEF also changes, so as the display will updated. However, this is not the case, I need to work for it manually.
3. The model of GEF is very difficult to update parallel with ArgoUML manually, because there is not much document discuss about the structure of GEF, and the data structure of GEF is very complex, it contain a groups of models that act as backend data structure, and a groups of figures that act as frontend data structure. Keeping ArgoUML and GEF data structure update parallel is very difficult.
4. The model itself is not serializable, so moving the all the objects from one side to server is very hard. The prototype just implement a little part of objects in ArgoUML.

5.2.3 Solutions of the problems

Actually this is a general problem of try to merge 2 difference model, and we can solve this by tracking the event flow.

Generally when the user need to make the needed modification, he need to issue some event/action to the system, we can track these event and save the related information. Then use these information to do our algorithm.

The system need to capture the mouse/key event from the user. The system need to filter the event so that only event issue to create/delete node/edge are captured. Then save the events in a pool, and then apply the MVCC algorithm on these event.

6. Event sharing approach on software collaboration

6.1 Overview

Even sharing have been use in distributed computing for a very long time. Some distributed computing platform even provide a standard event passing mechanism, e.g.: Jini distributed event.

Remote event model in Jini is like the event model in AWT and Java Bean so that Java developer can easily adapt the remote event model without a steep learning curve. However, problems for network environment should also be taken into account in remote event model. Jini notices about that and define a set of interfaces and conventions for distribute event.

However, most real time collaboration drawing tools do not use this approach. The reasons are follow:

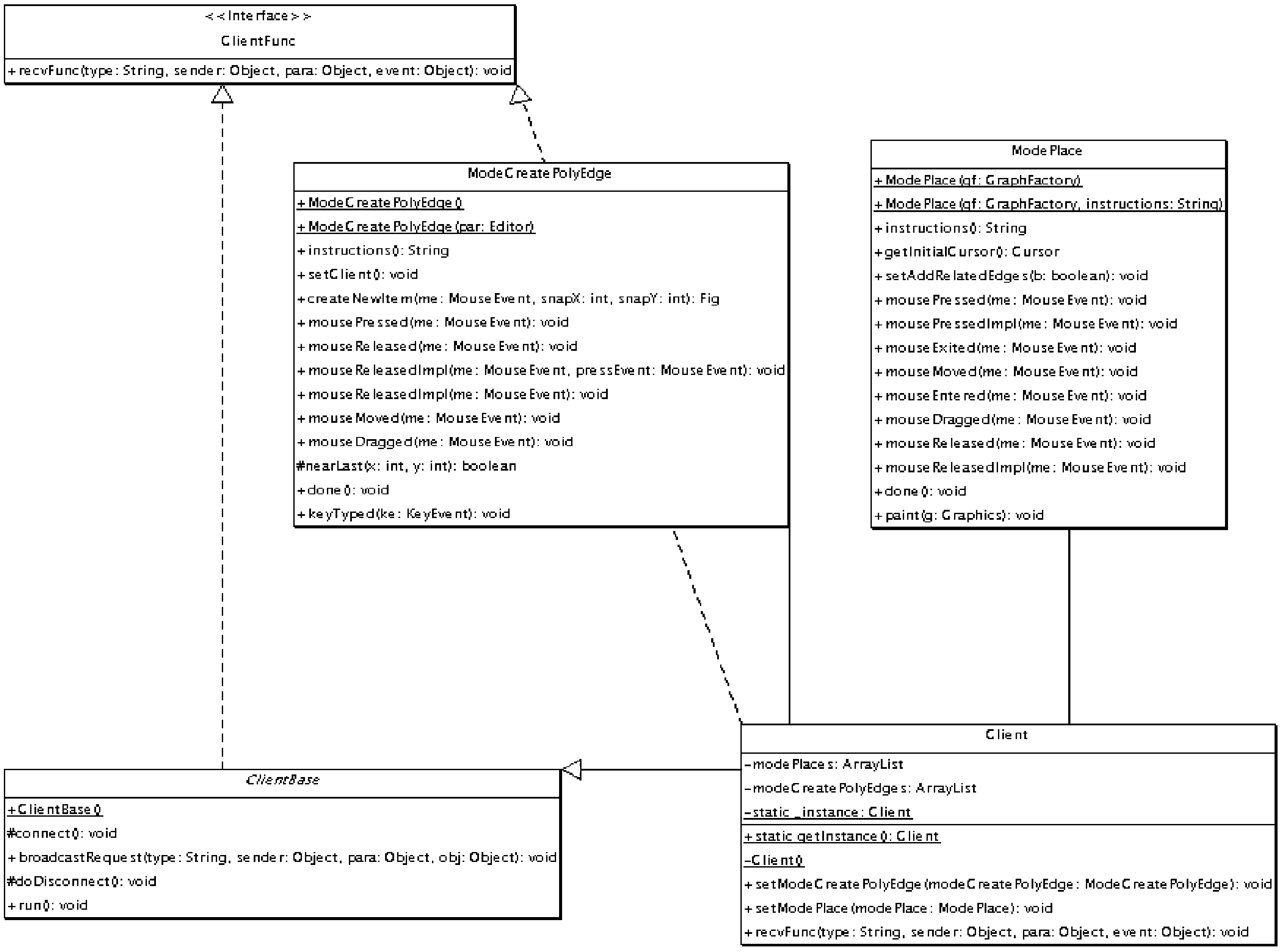
1) There are many events may need to manage, e.g.: Every mouse movement generate a mouse event. Every mouse movement may be useful. So there are lot of cases may need to consider.
2) Because there are so many events. There may need many network bandwidth to transfer all these events
3) In events sharing, it is harder to have a centralize model of the diagram, because it is hard to generate the model from events.

Because the problem encountered in sharing data structure seen not possible to solve easily, I decide to try this approach. At the time of implement this, the jini part is not really, so it just send the normal Java AWT event through socket network to simulate this.

6.2 Implementation details

1. Once the user click on the node/edge from the item menu, the mouse press and mouse release event will be record and broadcast to another clients.
2. We need to create some object (i.e.: Cmd* objects) if the clients have not create it.
3. Then the user click on the drawing pad and create node and edge, the mouse events also broadcast to other clients so the other client will also do the creating action.

6.3 The Class diagram of event sharing


<<Interface>>
ClientFunc
+recvFunc(type: String, sender: Object, para: Object, event: Object): void
ModeCreatePolyEdge
+ModeCreatePolyEdge()
+ModeCreatePolyEdge(par: Editor)
+instructions(): String
+setClient(): void
+createNewItem(me: MouseEvent, snapX: int, snapY: int): Fig
+mousePressed(me: MouseEvent): void
+mouseReleased(me: MouseEvent): void
+mouseReleasedImpl(me: MouseEvent, pressEvent: MouseEvent): void
+mouseReleasedImpl(me: MouseEvent): void
+mouseMoved(me: MouseEvent): void
+mouseDragged(me: MouseEvent): void
#nearLast(x: int, y: int): boolean
+done(): void
+keyTyped(ke: KeyEvent): void
ModePlace
+ModePlace(gf: GraphFactory)
+ModePlace(gf: GraphFactory, instructions: String)
+instructions(): String
+getInitialCursor(): Cursor
+setAddRelatedEdges(b: boolean): void
+mousePressed(me: MouseEvent): void
+mousePressedImpl(me: MouseEvent): void
+mouseExited(me: MouseEvent): void
+mouseMoved(me: MouseEvent): void
+mouseEntered(me: MouseEvent): void
+mouseDragged(me: MouseEvent): void
+mouseReleased(me: MouseEvent): void
+mouseReleasedImpl(me: MouseEvent): void
+done(): void
+paint(g: Graphics): void
ClientBase
+ClientBase()
#connect(): void
+broadcastRequest(type: String, sender: Object, para: Object, obj: Object): void
#doDisconnect(): void
+run(): void
Client
-modePlaces: ArrayList
-modeCreatePolyEdges: ArrayList
-static _instance: Client
+static getInstance(): Client
-Client()
+setModeCreatePolyEdge(modeCreatePolyEdge: ModeCreatePolyEdge): void
+setModePlace(modePlace: ModePlace): void
+recvFunc(type: String, sender: Object, para: Object, event: Object): void

6.4 Conclusion

The similar work implemented for all edge and node creation of ArgoUML, so that every kind of UML diagram in ArgoUML can share the model/node creation/deletion.
The advantage of using this approach are:

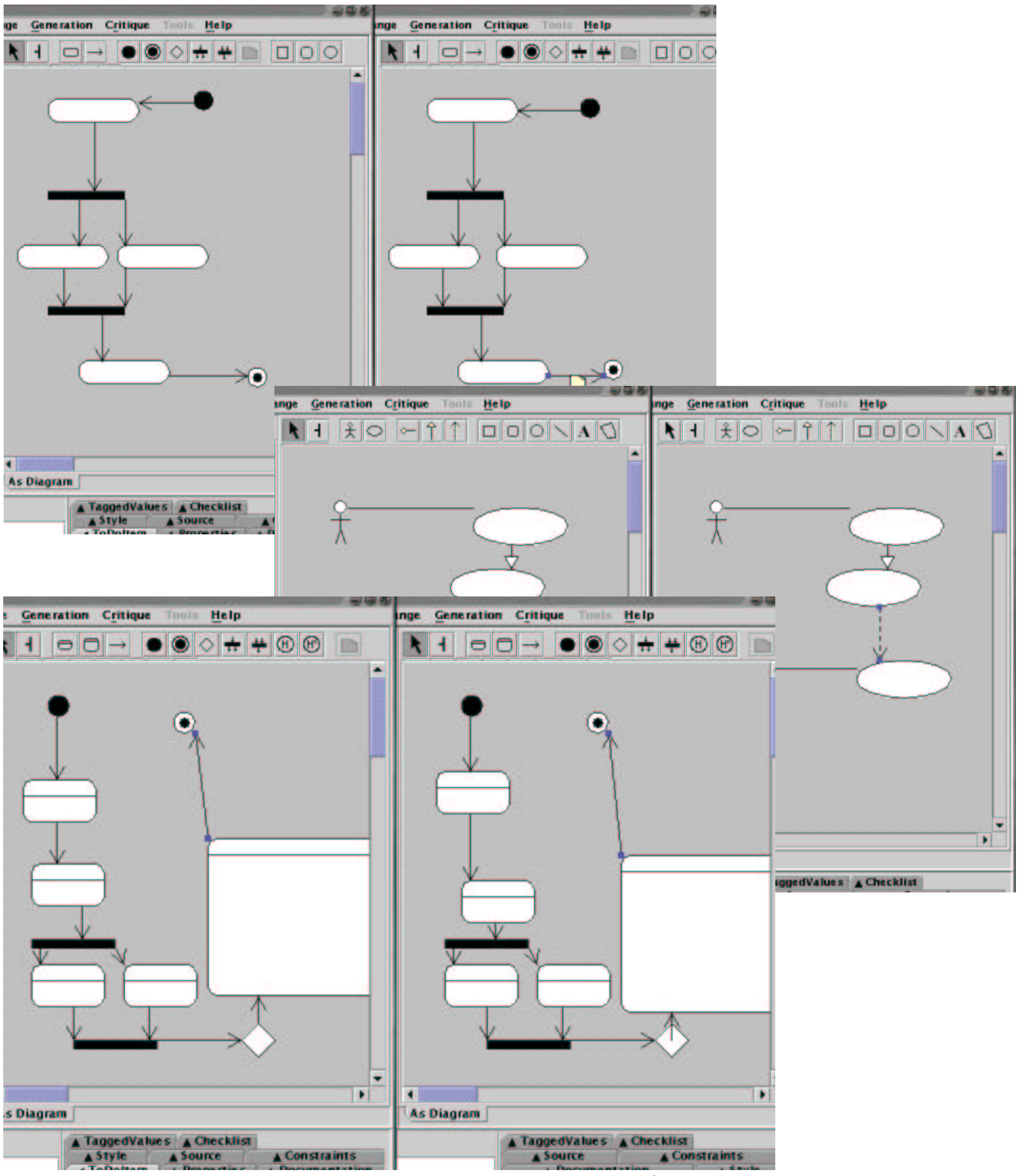
1. Network bandwidth friendly: it is much light weight for transfer an event than the model.
2. Instance response: The other client should add the model at the same time of the first creation of the model.
3. Apply to all diagram: As these codes patch the graphic framework package. So it able to share every kind of diagram if the modeling tools unify the node/edge creation/deletion into one graphic framework.
4. Relatively easy to do: The modeling tools may compose of many package, for examples, ArgoUML consist NSUML and GEF. The approach that share the model itself need to analysis all these package, then transfer and manage the object of individual package. Which need many time to do. Event sharing can let the develop encapsulate all these but handle the outer most interface of the modeling tools.
5. Even if the other client are busy, the event can be handle, because the client run in a separation thread.

However, there are also some constraints of this approach, which should common to all implementation of event sharing of modeling tools:

1. The current diagram will consume the mouse event, so all user must work on same diagram in order to process the event correctly.
2. It does not support for the client to join after some user have started. Although it is possible to save all the event and apply them for the new join client to create the models, it need to handle many unexpected exception of doing this, e.g.: If one event for some reason cannot be handle, is probably mean that all the rest event cannot handle. This project just ignore this at this moment
3. If the client can't handle the event for some reasons, which are rare cases because TCP connection will resend. The other sides will lost that model. This can be solved by saving the event send and receive, and compare them periodically. If there is a client find the event received are less than other, he can request the other resend that event.
4. The edge creation event is depend on the node location. For example, if user 1 create generalization relation between 2 model, the create edge event will send to the other side, with the location of edge to be created. However, if user 2 move one of the related model

to the others place, the system can create the edge because it cannot find end nodes. We can send the ending nodes identifier because they are AWT events.

5. This approach is very not flexible, because other than handle the event, the system may do may need to do some pre-processing and post-processing. There are no hint in the event handler of how to do pre-processing and post-processing, the developer need to figure out how to do this in a try and error way. Other than this, doing pre-processing and post-processing may need to break the encapsulation of some package use. Which may not permitted.



The above diagrams are the screen shots of event sharing

7. UML data structure sharing at ArgoUML

Difference approach are used to share the operation and attribute in the model. It is extract the primitive of the model and transfer them to the server in a fix period. The reasons of doing this is follow:

7.1 Reasons for using a other approach in models sharing

1. The UML modeling tools need to update the backend model for any modification, e.g.: if the user set the name of a model to "model" there are total 5 modification of the backend model, if the system try to catch all the event and send to other, the network must overload and argouml will become very slow.
2. There are many places of code to capture the code, unlike creating and deleting the model. There is no way to unify them. We need to patch many difference class to capture all the event. However, the number of the event handler will increase or decrease for further developments. If the system still share the elements of the model by event sharing, Every new release of the software will bring new bugs. It is very like to have inconsistent operation/attribute sharing and program bugs.
3. Quality UML modeling tools usually provide UML validation when modify the nodes elements. If the system send the event to the other side directly, it is very easy to add some invalid modification step to the other sides, which make the model inconsistent. The worst case is the whole model is not displayable.

7.2 Detailed steps of model sharing

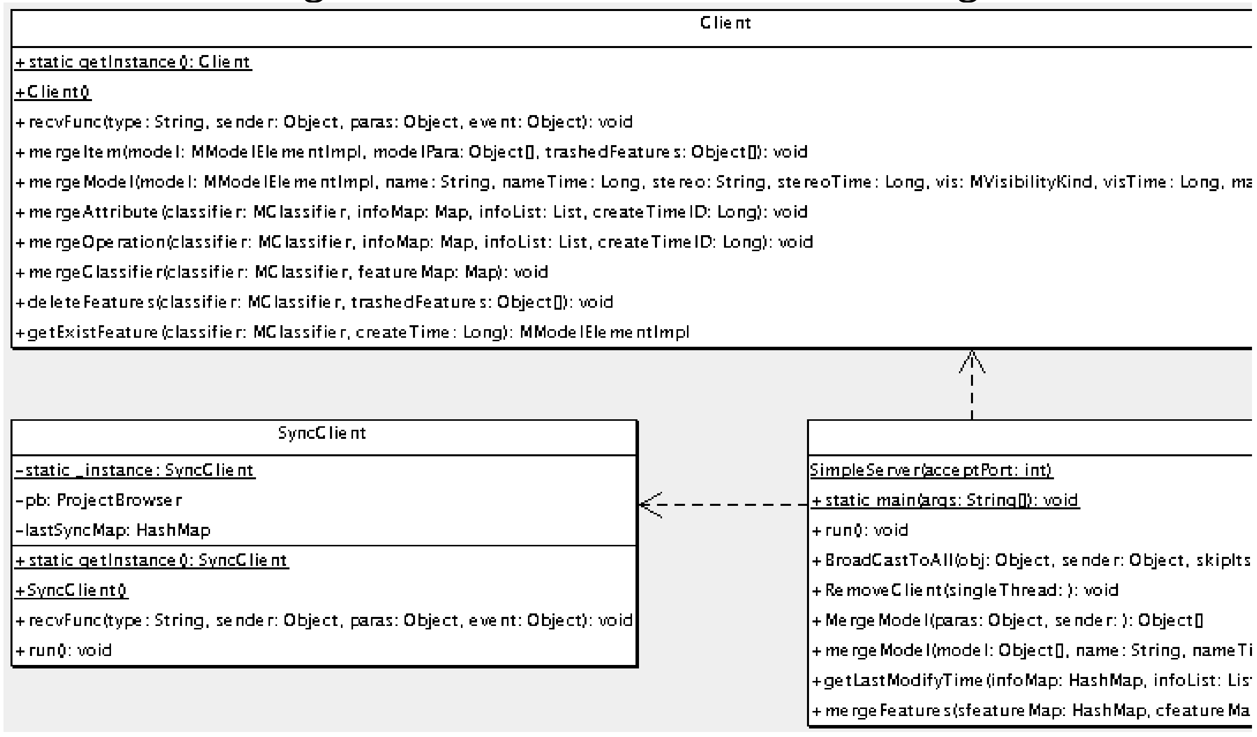
1. There is a thread running of every instance of UML modeling tools, at the client side, every several seconds it will analysis the model attributes and features of nodes/edges, extract the primitives items that are serializable.
2. Pack the primitives items into some defined format and broadcast to the other clients. The data structure of the sending formats is a tree structure with following elements:

1. ItemID	// For indentify difference item.
2. Timestamp	// The modification time of that Item
3. content objects	// The content of that items
4. childs	// items of other sub-elements

3. The other clients make/update the local model attributes and features accounting to the modification time and ID.

4. Timestamp needed to be added for every primitives, because the system need this to check which one is newer. If 2 items with some ID in a model, the system will keep the latest item.
5. The system assume the timestamp of different machines do not have large different, it is reasonable because we can synchronize the system time of different machines with NTP.
6. The system need to prevent the transfer items which are currently editing by somebody to the other sides, because it will cause IO exceptions. So the system need to find the modifying items and skip them.
7. Finally, reset the connection for every possible problems
8. Because of the time limitations, Only implement the model sharing for class diagram.

7.3 Class diagram of UML data structure sharing


Client
+static getInstance(): Client
+Client()
+recvFunc(type: String, sender: Object, paras: Object, event: Object): void
+mergeItem(model: MModelElementImpl, modelPara: Object[], trashedFeatures: Object[]): void
+mergeModel(model: MModelElementImpl, name: String, nameTime: Long, stereo: String, stereoTime: Long, vis: MVisibilityKind, visTime: Long, ma
+mergeAttribute(classifier: MClassifier, infoMap: Map, infoList: List, createTimeID: Long): void
+mergeOperation(classifier: MClassifier, infoMap: Map, infoList: List, createTimeID: Long): void
+mergeClassifier(classifier: MClassifier, featureMap: Map): void
+deleteFeatures(classifier: MClassifier, trashedFeatures: Object[]): void
+getExistFeature(classifier: MClassifier, createTime: Long): MModelElementImpl
SyncClient
-static _instance: SyncClient
-pb: ProjectBrowser
-lastSyncMap: HashMap
+static getInstance(): SyncClient
+SyncClient()
+recvFunc(type: String, sender: Object, paras: Object, event: Object): void
+run(): void
SimpleServer(acceptPort: int)
+static main(args: String[]): void
+run(): void
+BroadCastToAll(obj: Object, sender: Object, skipIts
+RemoveClient(singleThread:): void
+MergeModel(paras: Object, sender:): Object[]
+mergeModel(model: Object[], name: String, nameTi
+getLastModifyTime(infoMap: HashMap, infoList: Lis
+mergeFeatures(sfeatureMap: HashMap, cfeatureMa

7.4 Comparison of event sharing and data structure sharing

1. From the software design point of view, there is no clear winner. In some situation, using event sharing can have more clean code, more generic function and make the logic more simple to read. However, in some case data structure sharing is better because it is easier to have a backend model. In this project, both technique being used.
2. Generally speaking, the advantage of event sharing is real time response. This is very important in node and edge deletion in UML modeling. Because the otherside should know a node or edge be deleted by other side as soon as possible.
3. On the other hand, data structure sharing can let the sharing process as a background process. It is easier for collaboration when 2 or more user edit 2 or more diagrams.
4. Besides, there is no dependence between actions is another good thing of sharing data structure. In event sharing, it is very hard to resolve the problem occur if one action for any client fail to handle an action.

8. New versioning algorithm

After successful modify the single user UML modeling editor to support multiple users. It is the time to implement the collaboration algorithm proposed. However, there are some problems of the existing algorithm that have not consider before.

8.1 The problems of proposed approach

1. It has not prevent all conflict that will happen. For examples 2 clients modify same element in same time. The proposed algorithm has not address this situation. It simple try to keep the UML diagram is a valid state.
2. As it has not prevent all conflict occur, it also has not provide a solid conflict resolve method. In the implementation of model sharing, when conflict really happen, one of the user work will overwrite by the other user. It is not the suitable method. But if you apply the proposed algorithm in other implementation, the result may have more problems.
3. It will lost some work for the user, because the system will reject some changes that violate the dependence. The algorithm make the assumption the lost is not much. However, due to the network latency, it is not the case in any time.

Actually the proposed algorithm is not no value at all. First of all, it is a practical approach to implement the multiuser support of the system, it really work. And it help to user prevent the conflict occur in most case. However, as a research projects, we would like to adopt some more complex algorithm that address more problems in multiple user work.

In response to above weakness, Research have done on this topic. There is a tools call CVS address similar problems in text base documentation sharing. We can learn the mechanism of this tool to solve the above problems.

8.2 How CVS resolves conflict

8.2.1 Overview

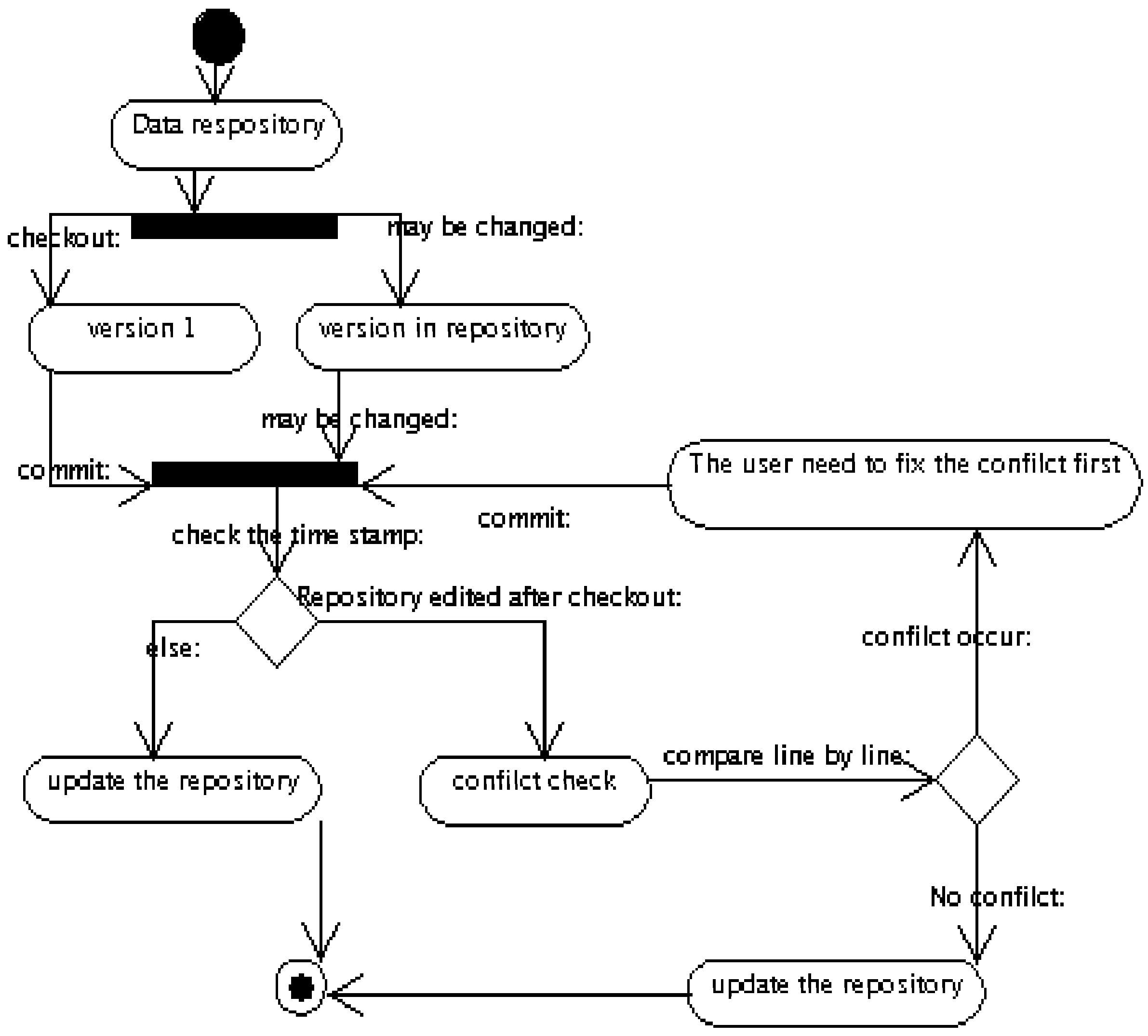
Below is the detail description of the conflict resolve of CVS, please refer to the appendices for the operation detail of CVS:

1. When there is a client checkout a document, CVS save the timestamp of checkout at the client.
2. Then the client do editing on that document, once finish the editing, the client commit the change to the central repository.
3. During committing document, CVS server check the checkout timestamp at the client. If the checkout timestamp is later than the timestamp of latest version of the document, CVS server do the following tasks:
    1. It update the server document.
    2. Assign a new version of the document.
    3. Save the timestamp of that version.
4. Otherwise, it check the difference of the document at the server and the client.

```
UInt32
   CountStringsInList (
       Ptr     inData)
   {
   <<<<<<< Conflict.c
       /* Alice: added the assertion */
       AssertIf_ (Ptr == nil);
   =======
       /* Bob: ignore nil input */
       if (Ptr == nil) return;
   >>>>>>> 1.12
       return *(UInt32*)inData;
 1.    }
```

5. The reason of doing this is simple. If the checkout timestamp is earlier than the latest version timestamp, than mean some other clients commit the document after checkout of that client. CVS cannot determine keeping the change from which client, so it keep both version and left it to the client to edit it.
6. After merging both changes, CVS send the merged document to the client without saving the document to reposition. But it will update the checkout time at the client.
7. After that client make suitable change of that document, then CVS server repeat step 3).

8.2.3 The action diagram of the flow



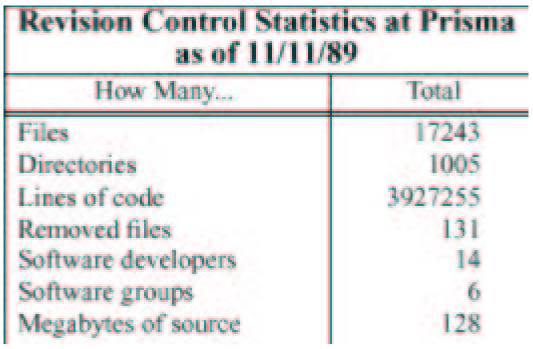
8.2.4 CVS Experience with SunOS 4[26]

8.2.4.1 Overview

CVS is widely used in software development and document management, and it have successfully help difference organization to mange their code or document. [26] is a paper analysis the theory and usage of CVS very detail, using SunOS 4 as an example

8.2.4.2 Scalablity of CVS

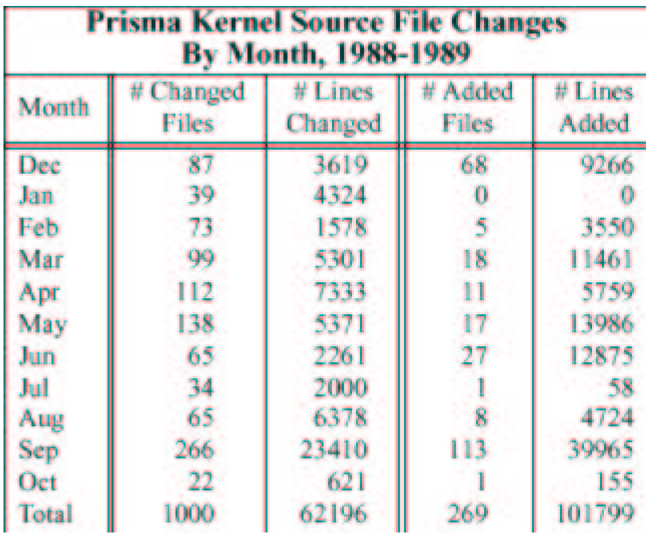
Table 1 show in SunOS 4, how many file have been managed with CVS. Table 2 shows the history of files changed or added and the number of source lines. Only changes made to the kernel sources are included. The large number of source file changes made in September are the result of merging the SunOS 4.0.3 sources into the kernel.



Revision Control Statistics at Prisma as of 11/11/89

How Many...	Total
Files	17243
Directories	1005
Lines of code	3927255
Removed files	131
Software developers	14
Software groups	6
Megabytes of source	128

Table 1: cvs statistics



Prisma Kernel Source File Changes By Month, 1988-1989

Month	# Changed Files	# Lines Changed	# Added Files	# Lines Added
Dec	87	3619	68	9266
Jan	39	4324	0	0
Feb	73	1578	5	3550
Mar	99	5301	18	11461
Apr	112	7333	11	5759
May	138	5371	17	13986
Jun	65	2261	27	12875
Jul	34	2000	1	58
Aug	65	6378	8	4724
Sep	266	23410	113	39965
Oct	22	621	1	155
Total	1000	62196	269	101799

Table 2: cvs usage history for the kernel

8.2.4.3 Conclusion – The algorithm of CVS is suitable for us

The performance of cvs is currently quite reasonable. In this paper, there are not specific tuning cvs done. Checking out the entire kernel source tree (1223 files/59 directories) currently takes 16 wall clock minutes on a Sun-4/280Mhz. However, bringing the tree up-to-date with the current kernel sources, once it has been checked out, takes only 1.5 wall clock minutes. Updating the complete 128 MByte source tree under cvs control (17243 files/1005 directories) takes roughly 28 wall clock minutes and utilizes one-third of the machine. For now this is entirely acceptable; improvements on these numbers will possibly be made in the future.

This result show that the algorithm is suitable for use in real time UML modeling sharing. Even if the performance is poor, there are many ways to tune. Thus this proof the algorithm of CVS is suitable to adopt in our system.

8.3 The new versioning algorithm

8.3.1 Overview

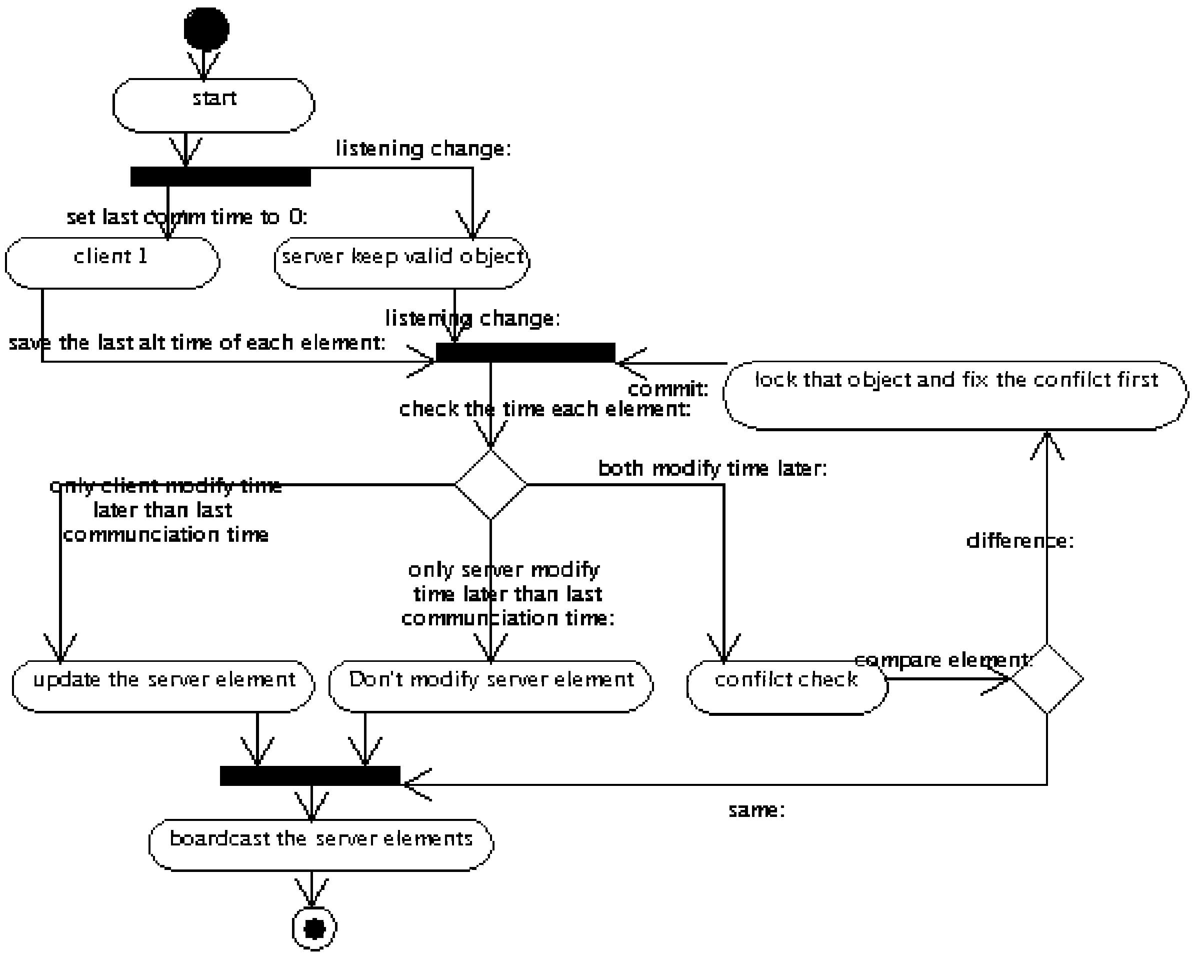
For the UML modeling sharing, a similar mechanism to CVS can be used to prevent implicitly locking of a part of model. Here is the brief description of the flow:

1. Normally the client does not need to lock the nodes when editing.
2. Every client has a thread to get the local model for a fixed period. It will send the client model to a center repository. Every element to be transfered will carry a last modification timestamp. The client will also send the last communication time with the center repository.
3. Like CVS, the server will keep the latest version of every element as well as the timestamp. Then the server compare the last modification timestamp with the client's timestamp of every element. If the timestamp of server element is later than last communication time, and the timestamp of client element is also later than the last communication time, and two element are difference, conflict arises.
4. If only server element is later than last communication time, the client needs to update the elements from the server.
5. If only client element is later than last communication time, the server needs to update the elements from the client.
6. If the two models are the same, there is no need to change for whatever timestamp.
7. Concerning to the conflict resolve, once the server detects the conflict, it will broadcast a message to all clients to lock that model. When the other clients receive the locking message, they will lock that element and prevent further editing, and send the latest change of that model to the server. Once the server receives all models from all clients, it will merge a model with conflict like CVS, who will then commit the model.
8. That client will then resolve the conflict at that model and send it back to the server. The server will suppose that model has no conflict and update the central repository and all clients. Then it unlocks that model of that client.
9. Then the client updates the last communication time with server for whatever case.

The algorithm addresses all the problems mentioned before.

1. It provides solid conflict definition. The definition of conflict based on timestamp is very clear and simple for the system to handle. Other than only maintain a valid UML diagram, this algorithm actually addresses all possible situations that conflict will occur in a very simple ways.
2. It provides conflict resolve solution. This also provides a unified way to solve the conflict, although it is very simple that just left the user to merge the conflict manually. It can be extended to a more intelligence resolve solution later.
3. The user will not lose any work. As the system neither rejects any changes of the user, nor deletes any work of the user. All the work of the user can be considered safe.

8.3.2 The action diagram of the flow


start
listening change:
set last comm time to 0:
client 1
server keep valid object
listening change:
save the last alt time of each element:
commit:
lock that object and fix the confilct first
check the time each element:
only client modify time later than last communciation time
both modify time later:
difference:
only server modify time later than last communciation time:
update the server element
Don't modify server element
confilct check
compare element:
same:
boardcast the server elements

8.4 Problems encountered

The main difference of implementation between this algorithm and CVS, is the nature of the document. The basic element of CVS is line of character, and the document is a list of lines.

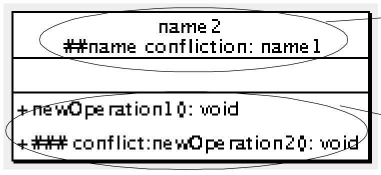
In contrast, the basic element of a UML document is more complex. A projects consist of many documents. A document consist of many node and edges, in class diagram there are class, interface and generalization relation; in use case diagram there are use case and include, extend relation.
Under all nodes and edge, they have differences elements, like operation, attribute, name, stereo types....
In order to simplify the case, only class diagrams have implement this algorithm to proof the concept. And assume that the elements of all kinds of nodes and edges are:

- name
- stereo type
- visibility

  For classifier like class and interface, there will be addition elements operation and attributes.

Other than this, as the UML modeling tools is not written by myself, lock the model to forbid user edit are forbid to do.

8.5 Screen shot of a model with conflict items

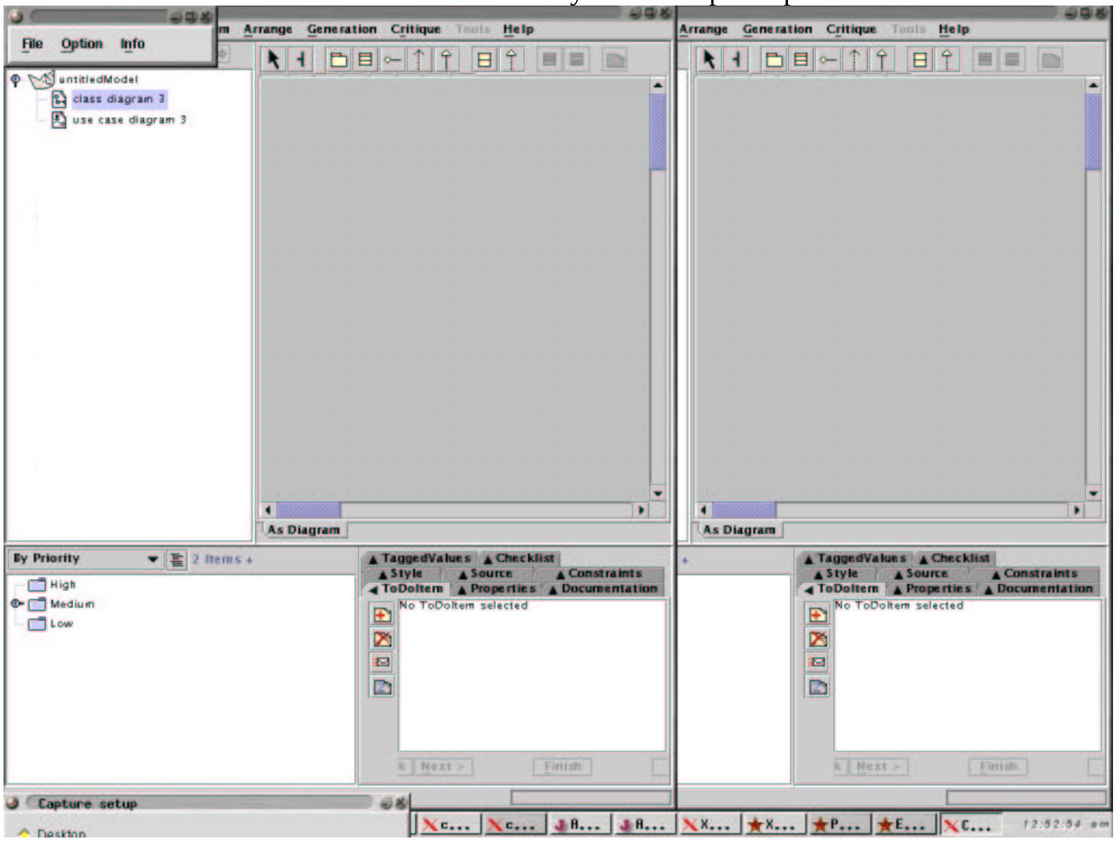

name 2
##name confliction: name 1
+newOperation1(): void
+### conflict:newOperation2(): void

One client call that model as **name1**, but other call it as **name2**

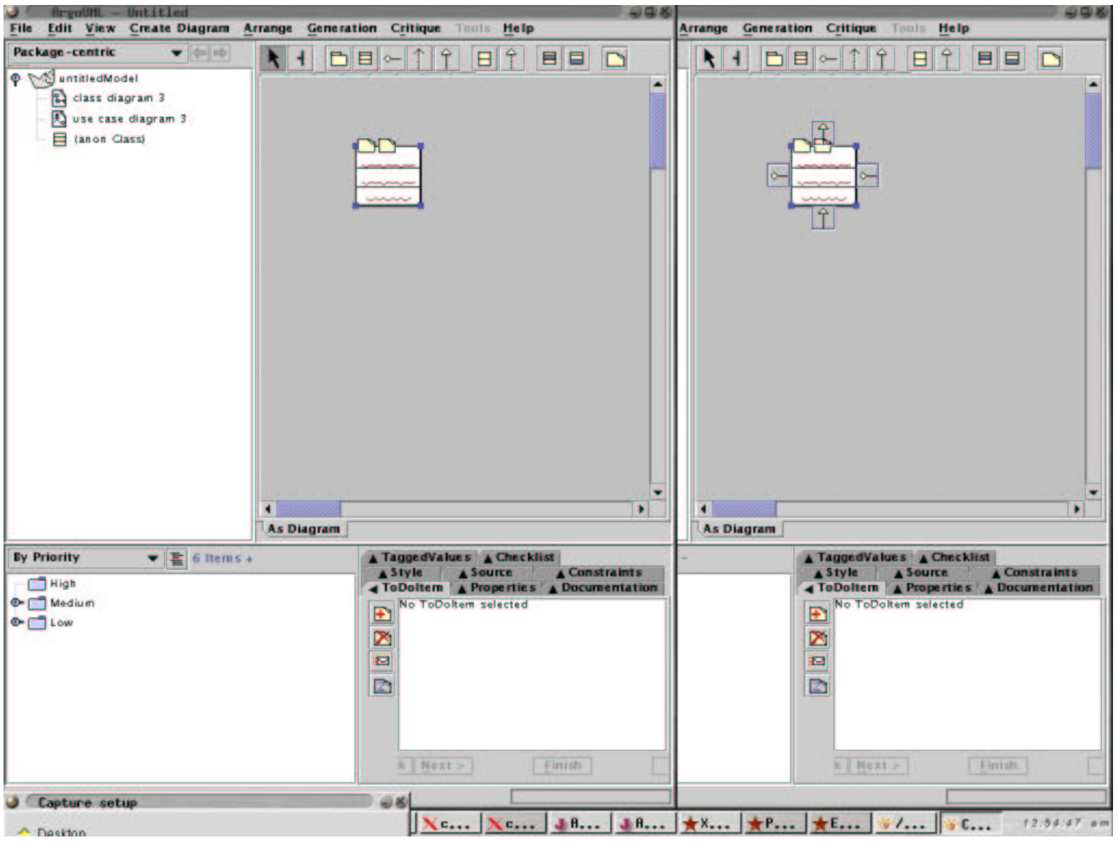
One client call that operation as **newOperation1**, but other call it as **newOperation2**

The flow of the system run:

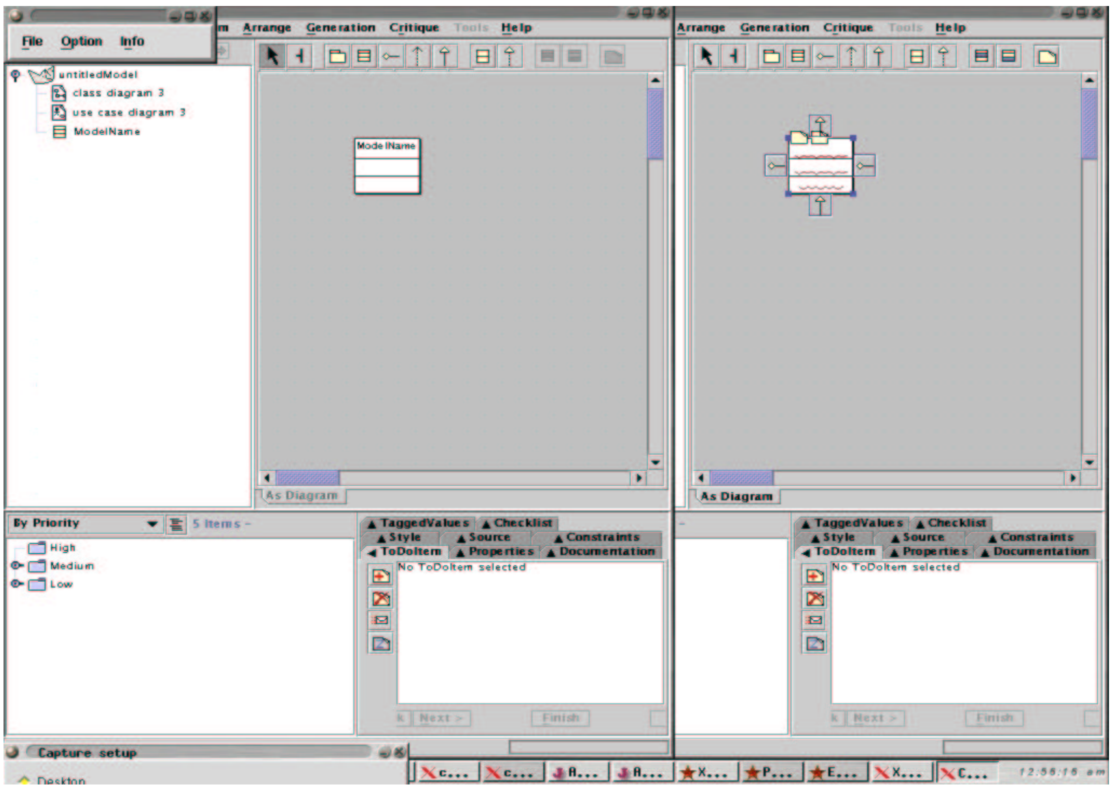
1. Initially, two instances of ArgoUML opened
2. Then user1 add a model, user2 add immediately, via ***event sharing***
3. Then user1 edit the name, user2 don't change immediately
4. After a while, the name of the model of user2 also change, via ***UML data structure sharing.***
5. Then 2 user continue to edit the diagram, adding and deleting models.
6. If they edit the some element, say name, in same time; the system will prompt for conflict arises
7. The users able to work collaborative in this system to improve performance



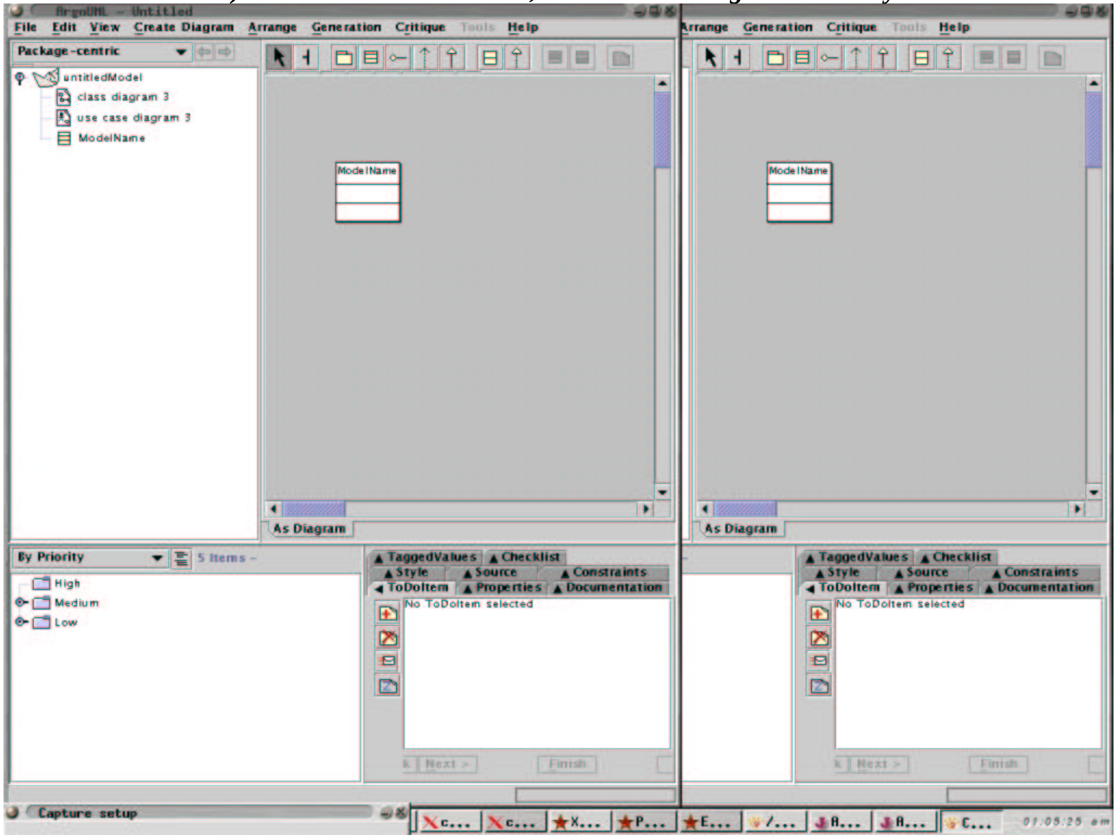
1) Initially, two instance of ArgoUML opened



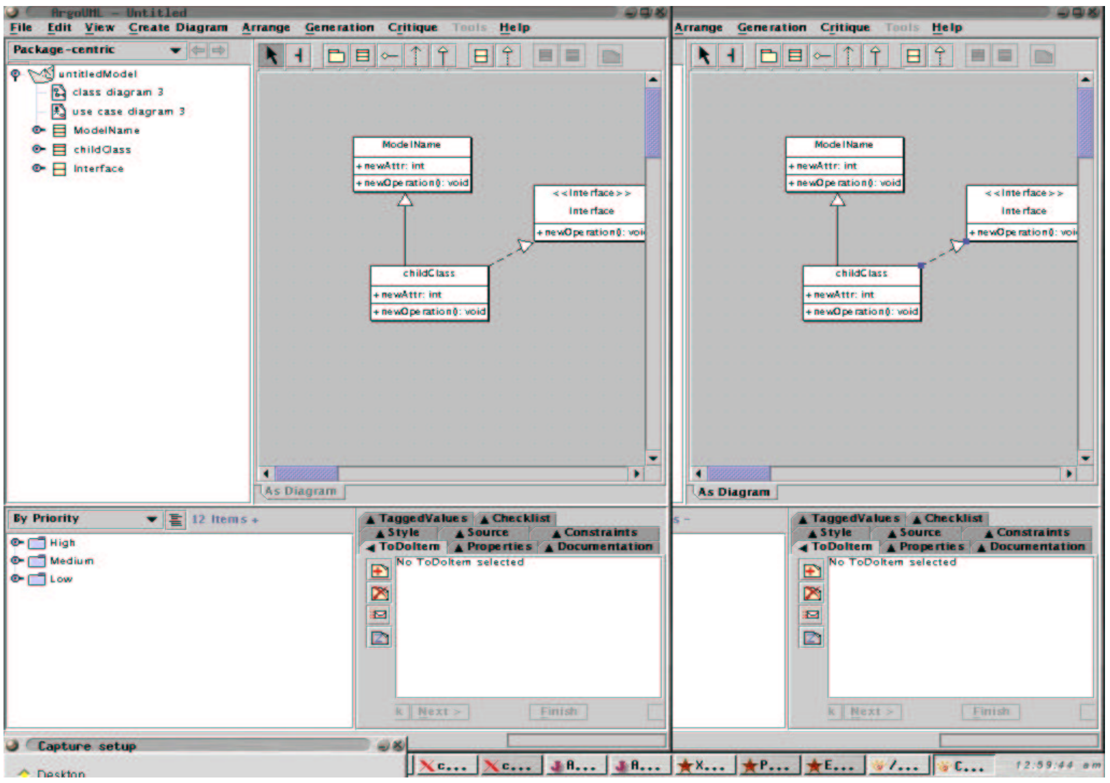
2) Then user1 add a model, user2 add **immediately,** ***via event sharing***



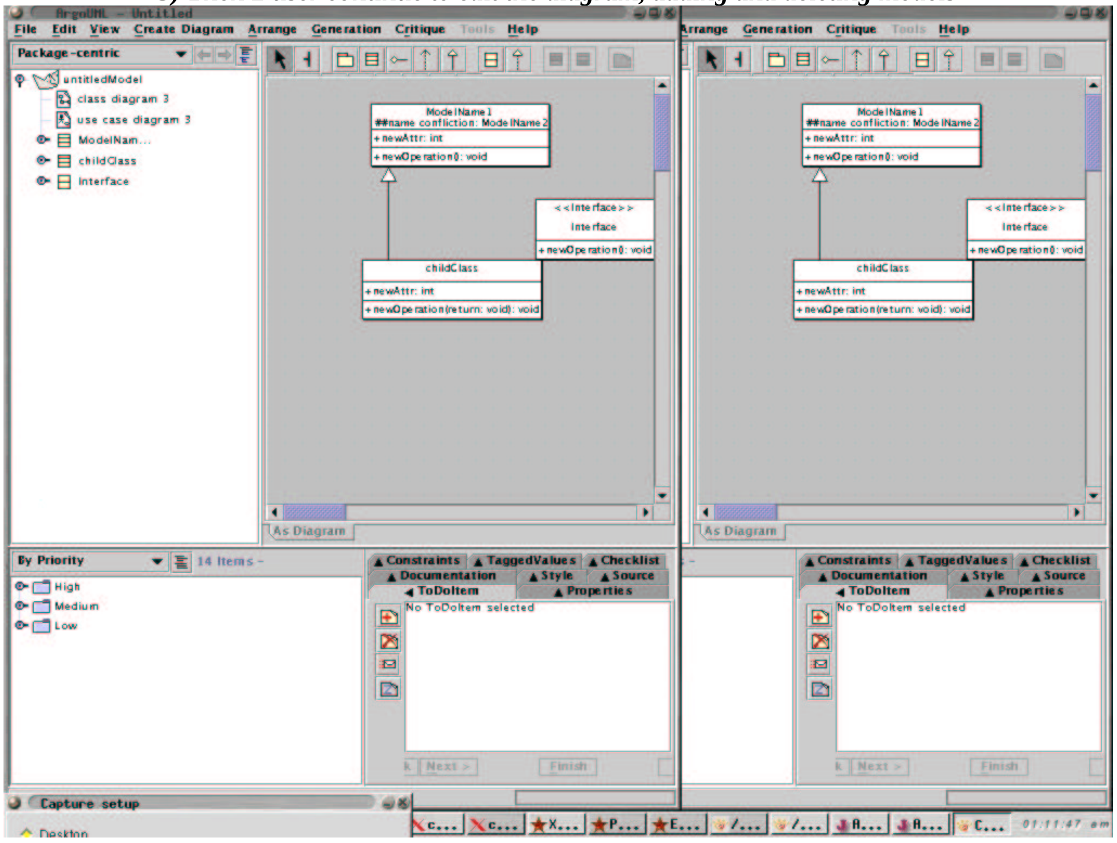
3) Then user1 edit the name, user2 don't change immediately



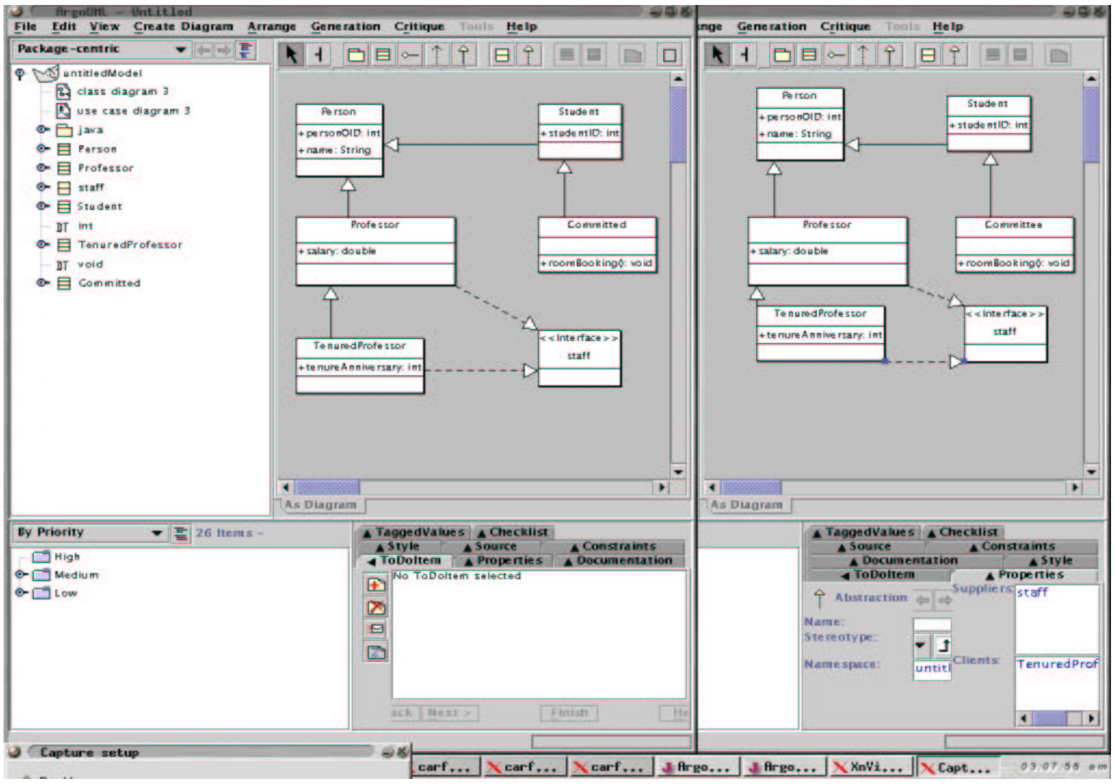
4) After a while, the name of the model of user2 also change, via UML data structure sharing.



5) Then 2 user continue to edit the diagram, adding and deleting models



6) If they edit the some element, say name, in same time; the system will prompt for conflict arises



7) The users able to work collaborative in this system to improve performance

8.6 Conclusion

1. The versioning engine is able to find the conflict when 2 user make modification of one unit in same time.
2. However, the implementation in this projects is not completed, because locking of the model of the UML modeling tools is not done. The modeling tools used is not allow the developer to lock the model from editing. Currently, the project only add the note to notify the other user this object have conflict and should not edit. However, after the conflict being resolve, there is no way to notify the other users currently.

9. Experience of enhanced public domain software

Here are the experiences of picking up the public domain/opensource software, and enhance it for the under graduate projects here. The following discussion is only relate to adding code on an existing project, not using opensource library.

9.1 Advantages of enhancing existing software

1. The software will provide much features related to the propose system. Thus it can save a lot of time from implementing many infrastructure items. Using this projects as an example, ArgoUML has already provided all the feature an UML modeling tools, we only focus on the collaboration support. Ideally, the project can build a collaborative UML editor that able to allow users to have collaboration work for all nine kinds of UML diagrams. Which is impossible to implement all these in a final year projects.
2. The skill of picking up a project and enhance it can be learnt. According to one famous developer: “Good programmer write program, best programmer re-write program”. It is important for a programmer to have the skill to pick up old project and enhance it. Especially in working environment, existing code are fully tested and documented. Re-implementing similar code need extract time on testing and documentation. Others than this, the programmer can learn good design from existing program.
3. Communication with the interest group of that project will help the project much. The interested group may provide good support and nice idea of the project.

9.2 Disadvantagse of enhancing existing software

1. The time spending in reading and understanding the project may be very long. Although good design can be learnt from this process, there may be not enough time available for this. Once final year begin, the class work are very heavy. In my own experience there are only 3-4 month for the project.
2. It is highly possible to broken the structure and style of the program when patching the system. There are some methodologies like refactoring to solve this problem, but it is possible of not having time or skill to apply these methodologies.
3. There are many unknown in working on an existing software. Using our project as an example, at first I suppose only need to know the code of the UML modeling tools. However, turn out I need to know the code of drawing framework (GEF) for event sharing, and the code of UML meta-model (NSUML) for versioning support. Other than reading extra code, there are possible that you cannot get the code of the libraries.

9.3 The consideration of choosing tools

1. Do not pick too large projects to work with. This depend on the programming experience, but for a final year projects, a small to median project that not excess 400 classes is suitable. Project with size larger is dangerous because there may be many un-expected problems in working on large projects. Other than that, too large project probably mean that the tools that you use consist of many function/code that are not needed.
2. The software design of the project should be checked before, to ensure that it is good use.
3. Identify all primitive are needed, do not assume anything without provide. In this project, I assume the data-structures are serializable and able to transfer in the network. But in fact there are extract work to extract the serializable data structure from UML data-structure. Here are some assumption that you need to proof generally.
   1. What are the objects that need to be transfer on the network? Are they serializable?
   2. Identify which functions belong to the tools, and which function belong to the libraries, make sure there is no need to edit the code of the libraries?

10. Conclusions of the project

10.1 Evaluation of extendeding ArgoUML

After the implementation, I find that there are some problems of using ArgoUML

1. The backend UML meta-model that are not serializable
2. The code need to learn are too large, just ArgoUML consist of 739 class, other than this, I need to patch the graphic framework GEF and UML meta-model NSUML, all consist of about 250 class.
3. The size of the projects is too large, so during the development, I need to make change of some code that not really relate to my part.
4. Because ArgoUML written by many people without pay, the coding style is not that consistence. Make it more difficult to learn.

ArgoUML is a nice tool to work with, it is a good UML editor, and have a highly modularized UML editing tools that allow user to add new components, like new code parsing module. However, what we intended to do need to make many changes of basic component, like UML metamodel and graphic framework. Extending ArgoUML to have versioning support is too difficult in a undergraduate project. However I will highly recommend someone like to develop an UML editor review ArgoUML first, because the design is good.

Other than ArgoUML, some other opensource UML editor are reviewed during implementation[30][31][32][33][34]. Some of them are likely designed, but none of them able to modify to a network available, collaborative UML editor without heavy work. Because most are heavily couple with the UML meta-model package and drawing framework, but both are not really network ready. Thus, in conclusion, we need to build from zero for a real time collaborative UML modeling tools for versioning support.

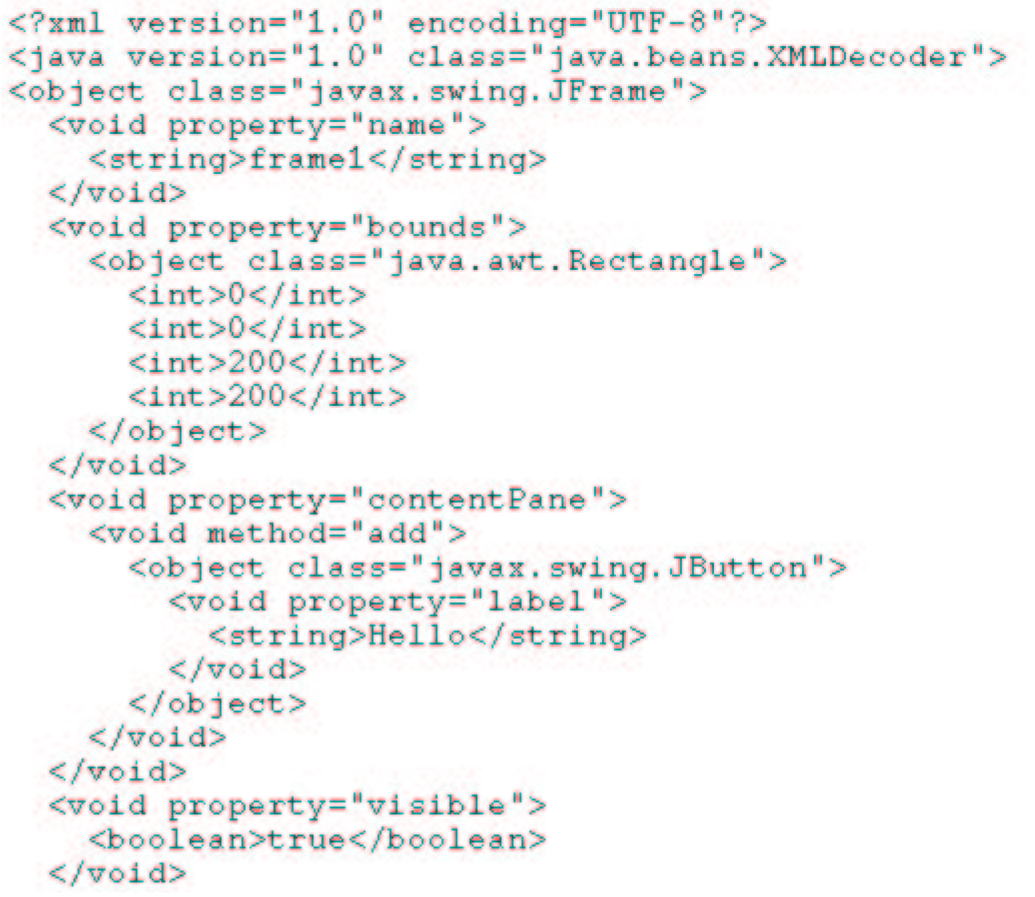
Personally speaking, I will think that it is worth to build a highly modularized UML editor from scratch, taking reference from the past projects[9].

10.2 Evaluation of XMI file format

The XMI standard is extensive and very big. But in fact a undergraduate project will not take the benefit from XMI. Our project neither need to communicate the model with other commerce tools like Rational Rose, nor need a file format that implement the complete specification of UML. All we really need is a XML structure for further processing. So in most case, the we spend a lot of time on XMI without real benefit.

Thus I will recommend using some UML meta data package that only implement partly of UML 1.3 up specification, but not the whole XMI standard. Dingouml[27] is a good choice, it implement the complete UML 1.3 specification, but it is much more simple than XMI package like NSUML. Then we can use XMLEncoder[28] to encode an XML document for us for further processing.

XMLEncoder is the new API from Java 1.4, which provide a standard way to encoding JavaBean to XML. The XML basically construct from the primitives of that JavaBean. Which are very handle to use.



```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.0" class="java.beans.XMLDecoder">
<object class="javax.swing.JFrame">
  <void property="name">
    <string>frame1</string>
  </void>
  <void property="bounds">
    <object class="java.awt.Rectangle">
      <int>0</int>
      <int>0</int>
      <int>200</int>
      <int>200</int>
    </object>
  </void>
  <void property="contentPane">
    <void method="add">
      <object class="javax.swing.JButton">
        <void property="label">
          <string>Hello</string>
        </void>
      </object>
    </void>
  </void>
  <void property="visible">
    <boolean>true</boolean>
  </void>
```

An example output of XMLEncoder

10.3 Evaluation of versioning collaboration methodology

In compare with the collaborative UML modeling tools using locking, versioning have the following advantage:

1. The user don't need to lock any model before editing. This make the editing more smooth, as repeating locking models and unlock models are tedious, especially for a large diagram.
2. The user don't need to wait for the other unlock the model. Because lock and unlock are tedious, so it is very easy for the designer forget unlock the model after editing. So if the other designer want to edit that model, he need to remind that designer unlock the model. This will become a big problem if that situation repeat many time. A even worst case is the developer forget to unlock the model before leaving his desk for a long time, e.g.: Go to lunch.
3. If the system crash, locking base collaborative UML editor may fail to unlock the model. Versioning system don't have this problem, as no one lock the model.
4. In the implementation point of view, versioning system is easier to program to survive if the server crash, because versioning system don't need to talk to server at any time. The communication time is tunable, so if the client know the server can't be reached, it can just try next time until it can merge with the reposition. In locking system, if the server crash, how can you request lock that model? If you just let the client lock the model if server crash, then if two client lock and edit same model, how can the server merge them later?

Other than implement a not difficult algorithm, there is no disadvantage of CVS at all, the CPU usage is low, the performance is good, the user just able to edit the diagram just like single user editor in most cases. In fact, the implementation of the algorithm is not too difficult. I will highly recommend the future using versioning algorithm rather than locking.

10.4 Evaluation of using Jini

This part is harder to have a conclusion, in this project, jini proofed to help us prevent many problems of traditional network environment, the lost of connection of server are easy to solve, the architecture help us prevent the possible problems brought by synchronization, like deadlock.

However, programming jini is much more time consumption than traditional network programming. It is easy for the developer to model the system as client and server, rather than a set of services. It is also more easy for the developer to find suitable network component, like a messaging server.

If the developer can easily identify the benefit that jini given match the project, then jini is the solution. For example, if you need to program a hospital embedded system, that need to frequency join and remove from the network, the embedded system need to able to re-config automatically according to the network joined, then jini is very suitable. In normal network application, like a collaborative UML editor, jini provide both advantage and disadvantage. If the deadline allowed, it should good to implement using jini, but it is not all the cases.

11. Furthur Enchancement

11.1 New Design of the collaborative UML editor

After doing the final year project. I have some idea of how to model a collaborative UML editor

1. The backend process like UML validation should be moved to the server. Some work like UML meta-data management, archiving work and the project management should move to the server to do. The clients should only consist of UI and drawing framework.
2. We can make our custom wrapper of UML meta-model and drawing framework. Thus we can have change the UML meta-model and drawing framework.
3. Provide a command line UML processor before the editor. If the design of a UML editor is good, the component of UML metamodel processing should automatic, we can verify this via implement a command line tool, like an UML validator.

11.2 Further improvement of the collaboration architecture

1. This project only implement the engine of versioning control. But the versioning system still need other component, like versioning history management[29]
2. Integrates the versioning engine with WebDav, so that we can take the advantage of HTTP connection, like SSL security connection.[29]
3. The versioning engine does not know any information about the client in this release. So some work like the user like to see the modification only belong to him is not possible now.
4. User cannot control the synchronization of the ArgoUML and versioning engine at all. The next release should add some control for the user, like the user should able to prevent his work merge with versioning engine if he want.

11.3 The Improvement of versioning algorithm

1. The versioning algorithm can be improved to no need to lock the whole model. Currently the system just assume 2 items are difference and report conflict if both server version and clients version and later than the last sync time. That mean 2 user editing the same basic element, e.g. The name of model. However, the locking make 3rd user is not able to edit other elements in that model. e.g.: he cannot model the stereo type of that class.
2. The locking can be refine to a multiple level locking. We make refine the locking to edit lock, delete lock and adding lock. For some event, we just need to lock the model being delete, but able to add smaller elements. For example, if there are conflict of the name of an operation, it should be ok for other user to add the stereo type of that operation.
3. For the situation that 2 user edit the same diagram in start. But, after some time, for some reason, they need to edit 2 diagram individual. We can use the versioning algorithm to do the merging. But it is better to apply the algorithm in the XML save file rather than the object.

12. Reference:

1	jGuru.com. *Why should I consider optimistic versus pessimistic approaches to database updates.* available online: http://www.jguru.com/faq/view.jsp?EID=479243
2	The O'Reilly Network. *Postgresql's Multi-Version Concurrency Control.* available online http://www.onlamp.com/lpt/a//onlamp/2001/05/25/postgresql_mvcc.html
3	Jungkyu Rho and Chisu Wu. *An Efficient Version Model of Software Diagrams*. IEEE. Proceedings of APSEC '98, available online: http://selab.snu.ac.kr/~jkrho/apsec98.html
4	Jon Munson and Prasun Dewan. *A Flexible Object Merging Framework*. ACM CSCW Proceedings, Oct 1994, available online: http://www.cs.unc.edu/~dewan/abstracts/merge.html
5	Sun microsystems. *Jini network technology overview*, available online: http://www.sun.com/jini/overview
6	Source of netbeans, available online: http://www.netbeans.org/devhome/download.html
7	Source of jedit, available online: http://jedit.sourceforge.net/index.php?page=download
8	Netborne, available online: http://www.digitalschemes.com/
9	Paul Lee, Web base real time collaborative UML editor, Available online: http://ils.comp.polyu.edu.hk/fypd/2000/6110/96258637d.pdf
10	Infoworld, ArgoUML offers unique decision support, Available online: http://www.infoworld.com/articles/ec/xml/00/04/17/000417ecargo.xml
11	Object Management Group, CORBA, XML and XMI Resource Page, Available online: http://www.omg.org/technology/xml
12	Junichi Suzuki, UML exchange format and Pattern markup language, available online: http://www.yy.ics.keio.ac.jp/~suzuki/project/uxf
13	Normand Rivard, UML-Xchange, available online: http://sourceforge.net/projects/uml-xchange
14	Object by Design, Transforming XML to HTML, available online: http://www.objectsbydesign.com/projects/xmi_to_html.html
15	Dr Perdita Stevens, XMI Hackers' Homepage, available online: http://www.dcs.ed.ac.uk/home/pxs/XMI
16	Jim Huggins, Abstract State Machines: UML State Machines, available online: http://www.eecs.umich.edu/gasm/papers/umlverif.html

17	Ho, Wai Ming; Je'ze'quel, Jean-Marc; Le Guennec, Alain; Pennaneac'h, Franc,ois, An Extendible UML Transformation Framework, available online: http://www.inria.fr/rrrt/rr-3775.html
18	Alan Dix, Modelling Versions in Collaborative Work, available online: http://www.comp.lancs.ac.uk/computing/users/dixa/papers/version-PSE97
19	Alan Dix, Version Control for Asynchronous Group Work, available online: http://www.comp.lancs.ac.uk/computing/users/dixa/papers/version92/version92.html
20	An UML validation framework, vUML, available online: http://www.abo.fi/~iporres/vUML/vUML.html
21	GEF: Java Library for Connected Graph Editors, available online: http://gef.tigris.org
22	ArgoUML: A modelling tool for design using UML, available online: http://argouml.tigris.org
23	Novosoft, NSUML: Novosoft UML Library for Java, available online: http://nsuml.sf.net
24	Java.sun.com, Java metadata Interface, available online: http://java.sun.com/products/jmi
25	Miro Jurisic, Understanding CVS: A brief introduction to the concepts of CVS, available online:http://web.mit.edu/macdev/Development/Documentation/www/CVS%20Documentation/Understanding%20CVS.html
26	Brian Berliner, CVS II: Parallelizing Software Development, available online: http://www.fnal.gov/docs/products/cvs
27	Dingo, The free, open-source UML modeler, available online: http://www.dingouml.org
28	API document for XMLEncoder of JDK 1.4, available online: http://java.sun.com/j2se/1.4/docs/api/java/beans/XMLEncoder.html
29	IETF, Versioning Extensions to WebDAV, available online: http://www.ietf.org/rfc/rfc3253.txt
30	JUG - Java UML Generator, available online: http://jug.sourceforge.net
31	UML object modeller for Linux, available online: http://uml.sourceforge.net
32	Dia, a drawing program, available online: http://www.lysator.liu.se/~alla/dia
33	Quick UML for java, available online: http://sourceforge.net/projects/quj
34	UML Sculptor, available online: http://umlsculptor.sourceforge.net

35	Christian Heide Damm, Klaus Marius Hansen, Michael Thomsen, Michael Tyrsted, Tool Integration: Experiences and Issues in Using XMI and Component Technology, available online: http://www.ideogramic.com/download/resources/toolsEurope2000.pdf
36	IBM.com, XMI Opens Application Interchange, available online: http://www-4.ibm.com/software/ad/standards/xmiwhite0399.pdf
37	CoCons.org, Enhancing a UML Modelling Tool with Context-Based Constraints for Components, available online: http://www.cocons.org/publications/CCL_plugin_for_ArgoUML.pdf
38	Gary Lam, A Framework for an Agent-based Development Environment with Jini / JavaSpace -Internet Integrated Development Environment Framework(Internet-IDEF)

13. Appendices

13.1 Code snippets of core component

13.1.1 Code snippet of event sharing

Here is some code snippet of how to share the event:

13.1.1.1 Client side event sending example

The origin source are here:

http://gef.tigris.org/source/browse/gef/src/org/tigris/gef/base/ModeCreatePolyEdge.java?rev=1.4&content-type=text/x-cvsweb-markup

The following code is written by my for event sharing, the java file is org.tigris.gef.base.ModeCreatePolyEdge

```
  // Change the method that processing mouse release event.
  public void mouseReleased(MouseEvent me) {
    MouseEvent pressEvent;

    // Because  in many case we need to pass 2 event for single
task, so we need to cache or make some mouse event ourselves.
    if(_pastMousePressEvent != null)
      pressEvent = new
MouseEvent(_pastMousePressEvent.getComponent(),
          _pastMousePressEvent.getID(),
          System.currentTimeMillis(),
          _pastMousePressEvent.getModifiers(),
          _pastMousePressEvent.getX(),
          _pastMousePressEvent.getY(),
          _pastMousePressEvent.getClickCount(),
          _pastMousePressEvent.isPopupTrigger()
         );
    else
      pressEvent = new MouseEvent(me.getComponent(),
          me.getID(),
          System.currentTimeMillis(),
          me.getModifiers(),
          me.getX(),
          me.getY(),
          me.getClickCount(),
          me.isPopupTrigger()
         );
    Object[] para = {pressEvent, getArg("edgeClass")};
    // Then broadcast the events to the other sides
    _client.broadcastRequest("mouseReleasedImpl",
"ModeCreatePolyEdge", para, me);
    // And then actually run the event handler.
    mouseReleasedImpl(me);
  }
```

The server is only a thin server, it receive the clients object, then passed the object to other clients. Then the clients handle the events receive. There may be some work before handle the event. Like initalize the eventListener factory.

13.1.1.2 Client side receive event example

The java file is org.tigris.gef.base.Client

```
     // Check if the type match
     if(sender.equals("ModeCreatePolyEdge") &&
type.equals("mouseReleasedImpl")) {
            Object[] paras = (Object[])para;
            MouseEvent args1 = (MouseEvent)event;
            MouseEvent args2 = (MouseEvent)paras[0];
            Class edgeClass = (Class)paras[1];
            for(int i=0; i<modeCreatePolyEdges.size(); i++) {
    if(/* find the correct object */) {
        // handle the event.
        (/* The object */).mouseReleasedImpl(args1, args2);
        flag = true;
    }
}
// If the correct object not find, we need to initalize it as
preprocessing
            if(!flag) {
  CmdSetMode csm = Globals.getCmdSetMode(edgeClass);
  csm.doIt();
  ((ModeCreatePolyEdge)Globals.mode()).mouseReleasedImpl(args1,
args2);
}
```

There are other possible events need to patch are:

1. Mouse actions in org.tigris.gef.base.ModeCreatePolyEdge
2. Deletion actions in org.argouml.uml.ui.ActionRemoveFromModel

13.1.2 Code snippet of features sharing

13.1.2.1 The thread periodic run to send model

The java file is org.argouml.kernel.SyncClient

```
 // Code that get all model in the system
    MutableGraphSupport gModel =
(MutableGraphSupport)Globals.curEditor().getGraphModel();
    HashMap modelMap = gModel.itemIndex;
    Client client = Client.getInstance();
    List modelList = new ArrayList(modelMap.keySet());
    HashMap sendMap = new HashMap();

 // Loop through every elements
    for(int i=0;i<modelList.size(); i++) {
        Object thisModel = modelList.get(i);

// skip working model to prevent IOException
        if( thisModel.equals(pb.getDetailsTarget()) ||
            client.skipModel.contains(thisModel) ) {
            System.out.println("skip "+thisModel+" at "+getClass());
            continue;
        }
        HashMap featuresMap = new HashMap();
        if(modelList.get(i) instanceof MClassifier) {
// Code to extract the model element of Classifier
            List features =
((MClassifier)modelList.get(i)).getFeatures();
            for(int j=0;j<features.size(); j++) {
                HashMap infoMap = new HashMap();
                ArrayList infoList = new ArrayList();
// Code to extract the model element of Attribute, child of
classifier
// Extract individual itemID, timestamp, of each element
                if(features.get(j) instanceof MAttribute) {
                    MAttribute attr = (MAttribute)features.get(j);
                    MModelElementImpl model =
(MModelElementImpl)attr;
                    MStructuralFeatureImpl sfeature =
(MStructuralFeatureImpl)attr;
                    infoMap.put("MAttribute", "");
                    infoList.add("MAttribute");
                    infoMap.put(attr.getName(), new
Long(model.setNameTime));
```

```
                    infoList.add(attr.getName());
                    infoMap.put(attr.getVisibility(), new 
Long(model.setVisibilityTime));
                    infoList.add(attr.getVisibility());
                    infoMap.put(attr.getType().getName(), new 
Long(sfeature.setTypeTime));
                    infoList.add(attr.getType().getName());
                }
// Code to extract the model element of operation, child of 
classifier
// Extract individual itemID, timestamp of each element
                else if(features.get(j) instanceof MOperation) {
                    MOperation oper = (MOperation)features.get(j);
                    MModelElementImpl model = 
(MModelElementImpl)oper;
                    MBehavioralFeatureImpl bfeature = 
(MBehavioralFeatureImpl)oper;
                    infoMap.put("MOperation", "");
                    infoList.add("MOperation");
                    infoMap.put(oper.getName(), new 
Long(model.setNameTime));
                    infoList.add(oper.getName());
                    infoMap.put(oper.getVisibility(), new 
Long(model.setVisibilityTime));
                    infoList.add(oper.getVisibility());
                    List para1 = oper.getParameters();
                    ArrayList para2 = new ArrayList();
// Process the childs of operation, parameter 
                    for(int k=0;k<para1.size();k++) {
                        MParameter mpara = (MParameter)para1.get(k);
                        ArrayList paraInfo = new ArrayList();
                        paraInfo.add(mpara.getKind());
                        paraInfo.add(mpara.getType().getName());
                        paraInfo.add(mpara.getName());
                        para2.add(paraInfo);
                    }
                    infoMap.put(para2,  new 
Long(bfeature.setParameterTime));
                    infoList.add(para2);
                }
                MModelElementImpl model = 
(MModelElementImpl)features.get(j);
                Object[] infos= {infoMap, infoList};
                featuresMap.put(new Long(model.createTime), infos);
```

```
            }
        }
        else if(modelList.get(i) instanceof MRelationship) {
        }
// Process the remaining element of Classifier
        MModelElementImpl model =
(MModelElementImpl)modelList.get(i);
        MStereotype stereotype = model.getStereotype();
        String stereoName = stereotype != null?
stereotype.getName(): null;
        Long lastSyncTime = (Long)lastSyncMap.get(model);
        if(lastSyncTime == null) lastSyncTime = new Long(0);
        Object[] para = {featuresMap, model.getName(), new
Long(model.setNameTime), stereoName, new Long(model.setSte
    model.getVisibility(), new Long(model.setVisibilityTime),
lastSyncTime};
        sendMap.put(modelMap.get(modelList.get(i)), para);
        lastSyncMap.put(model, new
Long(System.currentTimeMillis()));
    }
    if(pb.getProject() == null) continue;
    Object[] trashList =
pb.getProject().getTrashedFeature().toArray();
    Object[] para = {sendMap, trashList};
    broadcastRequest ("MergeFeature", "SyncClient", para, null);
// Wait for 10000 milli-second.
                do {
                    Thread.sleep(10000);
                } while(client.RUN_FLAG);
            }
// Reset the connection for every un-expected exception
        } catch (Exception e) {
          if(e.getClass().getPackage().toString().indexOf("java.io")
< 0)
              e.printStackTrace();
          doDisconnect();
          connect();
          run();
        }
```

13.1.2.2 The merge of the details of the objects

The java file is org.argouml.uml.ui.Client

// Get the local models

```
        ArrayList slist = new ArrayList(gModel.itemIndex.keySet());
        Object[] para = (Object[])paras;
        HashMap cMap = (HashMap)para[0];
        Object[] trashedFeatures = (Object[])para[1];

        for(int i=0; i<slist.size(); i++) {
            MModelElementImpl model =
(MModelElementImpl)slist.get(i);
            Object id = gModel.itemIndex.get(model);
            Object[] modelPara = (Object[])cMap.get(id);
// skip working model to prevent IOException
            if(modelPara == null ||
model.equals(pb.getDetailsTarget())) {
                continue;
            }
// Merge the model one by one.
            mergeItem(model, modelPara, trashedFeatures);
        }

    public void mergeItem(MModelElementImpl model, Object[]
modelPara, Object[] trashedFeatures) {
        RUN_FLAG = true;
        HashMap featureMap = (HashMap)modelPara[0];
        String name = (String)modelPara[1];
        Long nameTime = (Long)modelPara[2];
        String stereo = (String)modelPara[3];
        Long stereoTime = (Long)modelPara[4];
        MVisibilityKind vis = (MVisibilityKind)modelPara[5];
        Long visTime = (Long)modelPara[6];
        mergeModel(model, name, nameTime, stereo, stereoTime, vis,
visTime, false);

        //Check if feature changed of a node
        if(model instanceof MClassifier) {
            MClassifier classifier = (MClassifier)model;
            deleteFeatures(classifier, trashedFeatures);
            mergeClassifier(classifier, featureMap);
        }
        if(model instanceof MRelationship){}
        RUN_FLAG = false;
    }
// Merge items other than classifer, relations, attribute and
operation
```

```
    public void mergeModel(MModelElementImpl model, String name,
Long nameTime, String stereo, Long stereoTime, MVisibilityKind vis,
Long visTime, boolean makeFlag) {
        if(model.setNameTime < nameTime.longValue() || makeFlag)
            model.setName(name);
        if(model.setVisibilityTime < visTime.longValue() ||
makeFlag)
            model.setVisibility(vis);
        if(model.setStereotypeTime < stereoTime.longValue() &&
stereo != null) {
            MStereotype stereotype = new MStereotypeImpl();
            stereotype.setName(stereo);
            stereotype.setNamespace(model.getNamespace());
            model.setStereotype(stereotype);
        }
    }

    public void mergeAttribute(MClassifier classifier, Map infoMap,
List infoList, Long createTimeID) {
        MModelElementImpl model = getExistFeature(classifier,
createTimeID);
        boolean makeFlag = false;
        MAttribute attr = null;
// Make that element if has not create before, otherwise modify it
        if(model == null) {
            attr = classifier.getFactory().createAttribute();
            model = (MModelElementImpl)attr;
            makeFlag = true;
        }
        else {
            attr = (MAttribute)model;
        }
        MStructuralFeatureImpl sfeature =
(MStructuralFeatureImpl)attr;
        mergeModel(model, (String)infoList.get(1),
(Long)infoMap.get(infoList.get(1)), null, new Long(0),
(MVisibilityKind)infoList.get(2),
(Long)infoMap.get(infoList.get(2)), makeFlag);

        if(sfeature.setTypeTime < (
(Long)infoMap.get(infoList.get(3)) ).longValue() || makeFlag) {
            String typeName = (String)infoList.get(3);
            MClassifier mtype = p.findType(typeName);
            attr.setType(mtype);
        }
        if(makeFlag) {
            ((MModelElementImpl)attr).createTime =
createTimeID.longValue();
            classifier.addFeature(attr);
        }
    }

    public void mergeOperation(MClassifier classifier, Map infoMap,
List infoList, Long createTimeID) {
        MModelElementImpl model = getExistFeature(classifier,
createTimeID);
        boolean makeFlag = false;
        MOperation oper = null;

// Make that element if has not create before, otherwise modify it
        if(model == null) {
            oper = new MOperationImpl();
            model = (MModelElementImpl)oper;
            makeFlag = true;
        }
        else {
            oper = (MOperation)model;
        }
        MBehavioralFeatureImpl bfeature =
(MBehavioralFeatureImpl)oper;
        mergeModel(model, (String)infoList.get(1),
(Long)infoMap.get(infoList.get(1)), null, new Long(0),
                    (MVisibilityKind)infoList.get(2),
(Long)infoMap.get(infoList.get(2)), makeFlag);

        if(bfeature.setParameterTime < (
(Long)infoMap.get(infoList.get(3)) ).longValue() || makeFlag) {
            List curParas = oper.getParameters();
            for(int k=0;k<curParas.size();k++) {
                oper.removeParameter((MParameter)curParas.get(k));
            }
            ArrayList newParas = (ArrayList)infoList.get(3);
            for(int k=0;k<newParas.size(); k++) {
                ArrayList paraInfo = (ArrayList)newParas.get(k);

                String typeName = (String)paraInfo.get(1);
                MClassifier mtype = p.findType(typeName);
                MParameter mpara = new MParameterImpl();
                mpara.setType(mtype);
                mpara.setKind((MParameterDirectionKind)paraInfo.get(
0));
                mpara.setName((String)paraInfo.get(2));
                oper.addParameter(mpara);
            }
        }
        if(makeFlag) {
            ((MModelElementImpl)oper).createTime =
createTimeID.longValue();
            classifier.addFeature(oper);
        }
    }

    public void mergeClassifier(MClassifier classifier, Map
featureMap) {
        ArrayList timeList = new ArrayList(featureMap.keySet());
        for(int j=0; j<timeList.size(); j++){
            Long createTime = (Long)timeList.get(j);
            Object[] infos = (Object[])featureMap.get(createTime);
            if(infos == null) {
                System.out.print(classifier);
                System.out.print(featureMap);
            }
            Map infoMap = (HashMap)infos[0];
            List infoList = (ArrayList)infos[1];   //We need this to
keep getting the needed information in sequence

            if(infoList.get(0).equals("MAttribute")) {
                mergeAttribute(classifier, infoMap, infoList,
createTime);
            }
            else if(infoList.get(0).equals("MOperation")) {
                mergeOperation(classifier, infoMap, infoList,
createTime);
            }
        }
    }

// Delete the features that have delete at other sides
```

```
    public void deleteFeatures(MClassifier classifier, Object[]
trashedFeatures) {
        if(trashedFeatures == null) return;
        List sfeatures = classifier.getFeatures();

        for(int j=0; j<sfeatures.size(); j++){
            MModelElementImpl model =
(MModelElementImpl)sfeatures.get(j);
            for(int i=0; i<trashedFeatures.length; i++){
                if( ( (Long)trashedFeatures[i] ).longValue() ==
model.createTime ) {
                    classifier.removeFeature((MFeature)model);
                }
            }
        }
    }
    public MModelElementImpl getExistFeature(MClassifier classifier,
Long createTime) {
        List sfeatures = classifier.getFeatures();
        MModelElementImpl model = null;
        for(int i=0; i<sfeatures.size(); i++) {
            if(createTime.longValue() == (
(MModelElementImpl)sfeatures.get(i) ).createTime) {
                model = (MModelElementImpl)sfeatures.get(i);
                break;
            }
        }
        return model;
    }
}
```

13.1.3 Code snippet of new versioning algorithm

Here is the code snip of the core logic:

```
// The method to merge the model
    public synchronized Object[] MergeModel (Object paras,
ServerThread sender) {

        Object[] para = (Object[])paras;
        HashMap cMap = (HashMap)para[0];
        ArrayList clist = new ArrayList(cMap.keySet());

// put the new added model in the repository
          if(modelMap == null) {
            modelMap = new HashMap();
            modelMap.putAll(cMap);
          }
          else {
              modelList = new ArrayList(modelMap.keySet());
            for(int i=0; i<clist.size(); i++) {
                modelID = (Integer)clist.get(i);
                if( !modelList.contains(modelID) ) {
                    modelMap.put(modelID, cMap.get(modelID));
                }
            }
        }

        modelList = new ArrayList(modelMap.keySet());
        for(int i=0; i<modelList.size(); i++) {
            modelID = (Integer)modelList.get(i);
            Object[] modelPara = (Object[])cMap.get(modelID);
            Object[] model = (Object[])modelMap.get(modelID);
            conflictFind = false;
            conflictObj = (Object[])model.clone();

// Delete the removed model from the repository
            if(modelPara == null) {
                modelMap.remove(modelID);
                      continue;
            }

            HashMap featureMap = (HashMap)modelPara[0];
            String name = (String)modelPara[1];
            Long nameTime = (Long)modelPara[2];
            String stereo = (String)modelPara[3];
            Long stereoTime = (Long)modelPara[4];
            MVisibilityKind vis = (MVisibilityKind)modelPara[5];
            Long visTime = (Long)modelPara[6];
            lastSyncTime = ((Long)modelPara[7]).longValue();

// Merge the model element changed of a node
            model = mergeModel(model, name, nameTime, stereo,
stereoTime, vis, visTime, false);

// Merge the features changed of a node
            if(featureMap.size() > 0){
                model[0] = mergeFeatures((HashMap)model[0],
featureMap);
            }

// Tell all the other client to lock that model
            if(conflictFind){
                ArrayList list = new ArrayList();
                list.add("Conflict");
                list.add("SimpleServer");
                list.add(conflictObj);
                list.add(modelID);
                sender.BroadCastToClient(list);
                System.out.println( list+" sent");
            }
            modelMap.put(modelID, model);
        }
        Object[] returnObject = {modelMap, para[1]};
        return returnObject;
    }

// Method to merge the basic model of a model
    public Object[] mergeModel(Object[] model, String name, Long
nameTime, String stereo, Long stereoTime, MVisibilityKind vis, Long
visTime, boolean makeFlag) {
        String sname = (String)model[1];
        Long snameTime = (Long)model[2];
        String sstereo = (String)model[3];
        Long sstereoTime = (Long)model[4];
        MVisibilityKind svis = (MVisibilityKind)model[5];
        Long svisTime = (Long)model[6];
```

```
// conflict case, both latest modify time of repository and client
and later than last communication time
        if(lastSyncTime < nameTime.longValue() && lastSyncTime <
snameTime.longValue() && sname != null && !sname.equals(name) &&
!sname.equals("") && !name.equals("") ) {
            conflictObj[1] = name + "\n ##name confliction: " +
model[1];
            conflictObj[2] = new Long(System.currentTimeMillis());
            conflictFind = true;
        }
// If only client time is later than last communication time, update
repository version
        else if(lastSyncTime < nameTime.longValue() || sname ==null)
{
            if( name != null && !name.equals("")) {
                model[1] = name;
                model[2] = new Long(System.currentTimeMillis());
            }
        }
//Merge stereo type
        if(lastSyncTime < stereoTime.longValue() && lastSyncTime <
sstereoTime.longValue() && sstereo != null &&
!sstereo.equals(stereo)) {
            conflictObj[1] = model[1] + "\n ##stereotype
confliction: " + stereo;
            conflictObj[2] = new Long(System.currentTimeMillis());
            conflictFind = true;
        }
        else if(lastSyncTime < stereoTime.longValue() ||
sstereo==null) {
            model[3] = stereo;
            model[4] = new Long(System.currentTimeMillis());
        }

//Merge visiability
        if(lastSyncTime < visTime.longValue() && lastSyncTime <
svisTime.longValue() && svis != null && !svis.equals(vis)) {
            conflictObj[1] = model[1] + "\n ##visibile confliction:
" + vis;
            conflictObj[2] = new Long(System.currentTimeMillis());
            conflictFind = true;
        }
        else if(lastSyncTime < visTime.longValue() || svis==null) {
            model[5] = vis;
            model[6] = new Long(System.currentTimeMillis());
        }
        return model;
    }

// Get the last modification time of a feature, choose the most last
modification time of an element belong to this feature
    public long getLastModifyTime(HashMap infoMap, List infoList) {
        long max = 0;
        for(int i=1;i<=3;i++){
            long value =
((Long)infoMap.get(infoList.get(i))).longValue();
            if(value > max) {
                max = value;
            }
        }
        return max;
    }

    public HashMap mergeFeatures(HashMap sfeatureMap, HashMap
cfeatureMap) {
        ArrayList ctimeList = new ArrayList(cfeatureMap.keySet());
        ArrayList stimeList = new ArrayList(sfeatureMap.keySet());

        for(int j=0; j<stimeList.size(); j++){
            Long createTime = (Long)stimeList.get(j);
            Object[] infos = (Object[])cfeatureMap.get(createTime);

// remove feature that removed in the client
            if(infos == null);
                sfeatureMap.remove(createTime);
        }

        for(int j=0; j<ctimeList.size(); j++){
            Long createTime = (Long)ctimeList.get(j);
            Object[] cinfos = (Object[])cfeatureMap.get(createTime);
            Object[] sinfos = (Object[])sfeatureMap.get(createTime);

//If not find at server items, mean that that feature need to add
            if(sinfos == null) {
                sfeatureMap.put(createTime, cinfos);
                continue;
            }

// if both have that item, we need to see if conflict arise
            HashMap cinfoMap = (HashMap)cinfos[0];
            List cinfoList = (ArrayList)cinfos[1];
            HashMap sinfoMap = (HashMap)sinfos[0];
            List sinfoList = (ArrayList)sinfos[1];

            long cLastModifyTime = getLastModifyTime(cinfoMap,
cinfoList);
            long sLastModifyTime = getLastModifyTime(sinfoMap,
sinfoList);

// Merge features, similar to basic element of UML model
            if(lastSyncTime < cLastModifyTime && lastSyncTime <
sLastModifyTime && lastSyncTime > 0) {
                HashMap conflictMap = (HashMap)conflictObj[0];
                ArrayList conflictList = new
ArrayList(conflictMap.keySet());
                Object[] conflictInfos =
(Object[])conflictMap.get(createTime);
                HashMap conflictInfoMap = (HashMap)conflictInfos[0];
                  ArrayList conflictInfoList =
(ArrayList)conflictInfos[1];

                String oldName = (String)sinfoList.get(1);
                Object oldValue = sinfoMap.get(oldName);
                String newName = "### conflict: "+oldName;

                conflictInfoList.remove(1);
                conflictInfoMap.remove(oldName);
                conflictInfoList.add(1, newName);
                conflictInfoMap.put(newName, oldValue);
                conflictMap.put(createTime, cinfos);
                conflictMap.put(new
Long(System.currentTimeMillis()), sinfos);

                conflictFind = true;
            }
            else if(lastSyncTime < sLastModifyTime||sinfoMap.size()
== 0) {
                sfeatureMap.put(createTime, cinfos);
            }
```

```
        }
        return cfeatureMap;
    }
}
```

13.2 Introduction to CVS

13.2.1 What is CVS?[25]

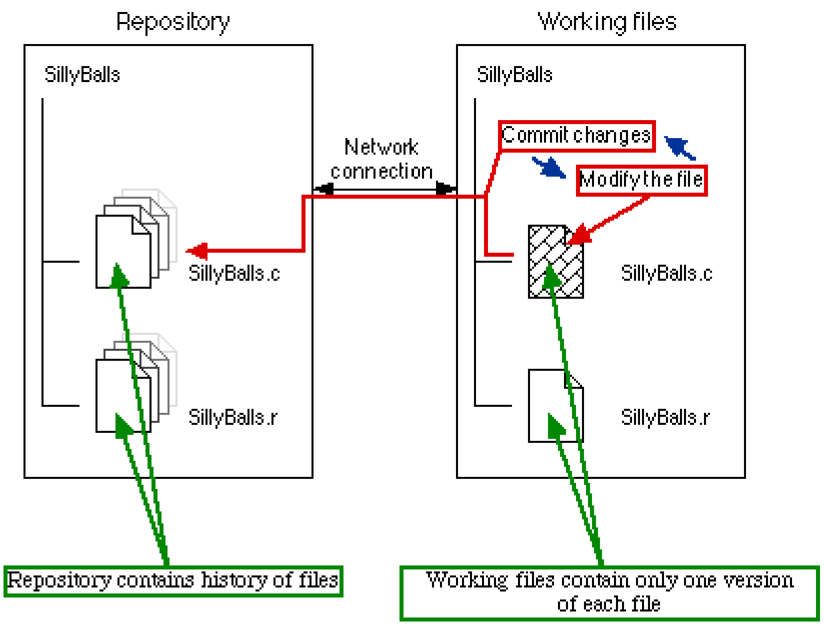
This part is quote from the reference, Concurrent Versions System, or CVS, is a revision control system that

1. Allows multiple developers to collaborate on software projects, while providing much help in keeping the projects in a consistent state although they may be manipulated by any number of developers at any given time
2. Allows a developer or developers to maintain version history of a software project and track changes made to the project over time
3. Allows developers to maintain several concurrent versions of a project, while providing help in moving changes among those versions, and preserving consistency of individual versions.

13.2.2 Basic ideas behind CVS

Two principal parts of a CVS system are the repository and working files. The repository resides on a CVS server, and contains history and version information about all files in the repository. Working files reside on developers' machines and only represent a particular revision of each file.

Suppose a single developer, Alice, is working on a project on her development machine. Then the picture looks like this:



The relation of repository and working files

Whenever she makes changes to some file in the project, Alice wants to record those changes in the repository so that there is a record of her progress later. The act of sending the changes made to working files to be incorporated into the repository is called 'committing the changes'. Alice commits her changes every day before she leaves work. After a while, the project history will grow, and the repository will contain information about past versions of all the files that are part of the project:

Thus we see the simplest form of CVS interaction: modify - commit - repeat. We will see later how this model will have to be somewhat extended later to cover for more complex scenarios.

As the project progresses, Alice may need to add or remove some files. She can do so using the CVS commands 'add' and 'remove', which schedule the files for addition or removal. However, the changes will not be propagated to the repository until Alice commits the files.

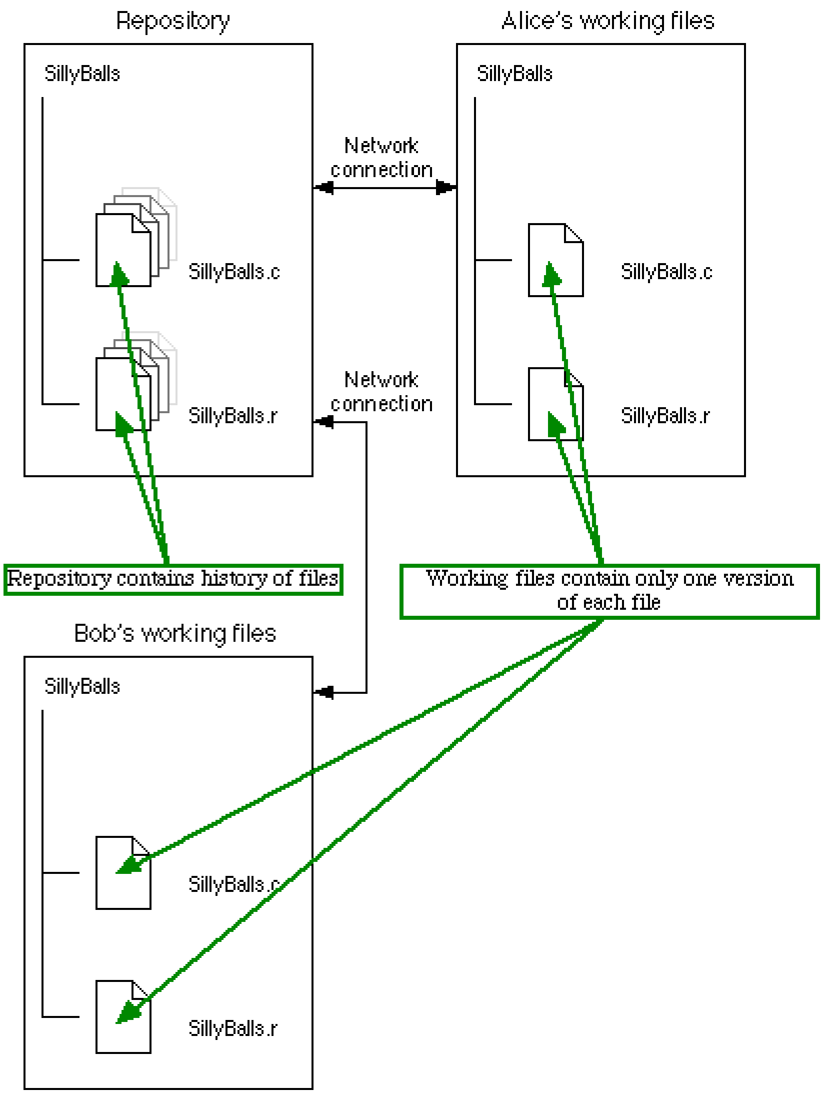
Thus we see the first important rule of CVS:

The repository is ALMOST never modified until you commit your changes

Exceptions to this rule are rare, and will be pointed out as they arise.

13.2.3 Getting slightly more complex: multiple developers

Some day, another developer, Bob, is assigned to work on Alice's project. They now both use CVS to work the project. The picture is now like this:



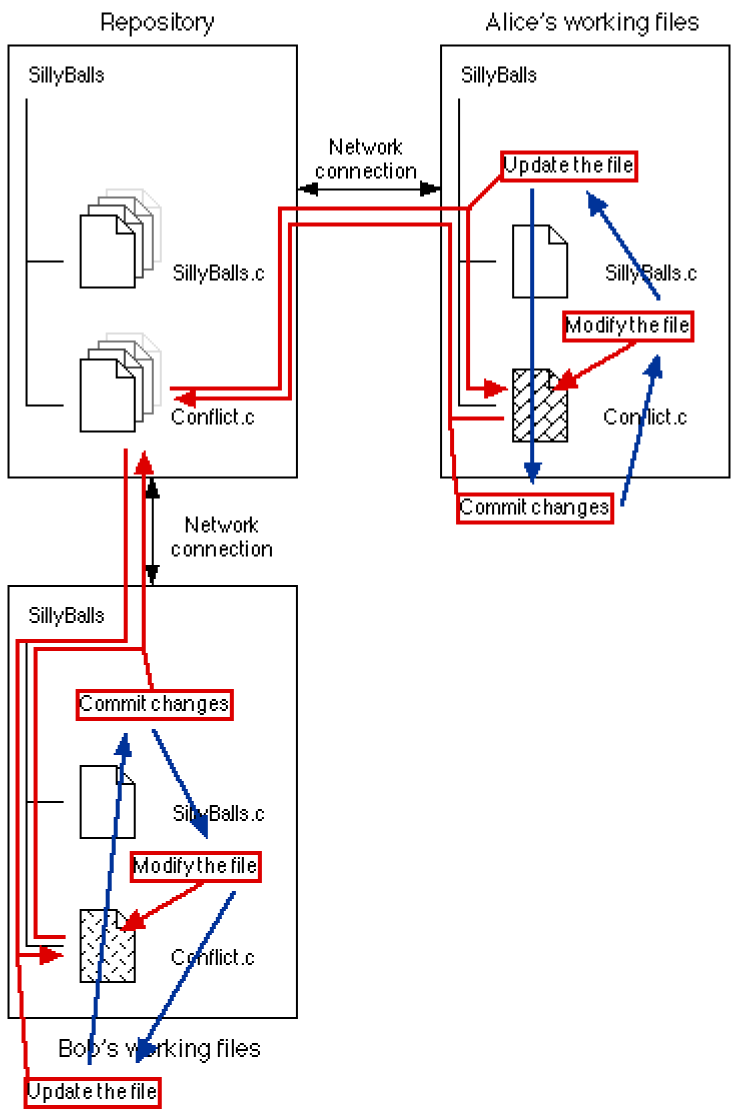
Multiple user editing one document with CVS

They happily work on two separate parts of the project, until some day Alice decides to modify the file called Conflict.c. Unbeknownst to her, Bob has already modified the same file, and committed his changes. Alice makes her modifications to the file, and attempts to commit her changes. However, she gets the following message from CVS:

```
cvs server: Up-to-date check failed for `Conflict.c'
```

cvs [server aborted]: correct above errors first!

Alice realizes from this cryptic message that the problem is that her working version of Conflict.c was not up-to-date before she had modified it. She needs to update her working file to the most recent version of Conflict.c from the repository, before she can commit her changes. She uses the CVS 'update' command to update her working files, and after that she happily commits her changes.



The "Copy-Modify-Merge" flow diagram of CVS

Clearly, Bob and Alice need to modify their working pattern to accommodate for the fact that they might be working on the same files from time to time. They change their use of CVS to:

modify - update - commit - repeat. That way they are almost completely sure that, whenever they attempt to commit changes, they will be successful.

13.2.4 Fighting for control: merge conflicts

Soon, Alice and Bob discover that they were too hopeful the last time they revised their habits. One day, Alice diligently attempted to update her local files, only to get an error:

```
Merging differences between 1.11 and 1.12 into Conflict.c
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in Conflict.c
C Conflict.c
```

After briefly consulting with Bob, Alice realizes that she modified the same portion of the file as Bob, and that CVS decided that their sets of changes were incompatible. However, looking inside her local version of Conflict.c, Alice finds the following:

```
UInt32
CountStringsInList (
    Ptr     inData)
{
<<<<<<< Conflict.c
    /* Alice: added the assertion */
    AssertIf_ (Ptr == nil);
=======
    /* Bob: ignore nil input */
    if (Ptr == nil) return;
>>>>>>> 1.12
    return *(UInt32*)inData;
}
```

After a brief discussion with Bob (during which he is sent to read some books about writing solid code), they agree that Alice used the correct approach. Alice modifies the file to read:

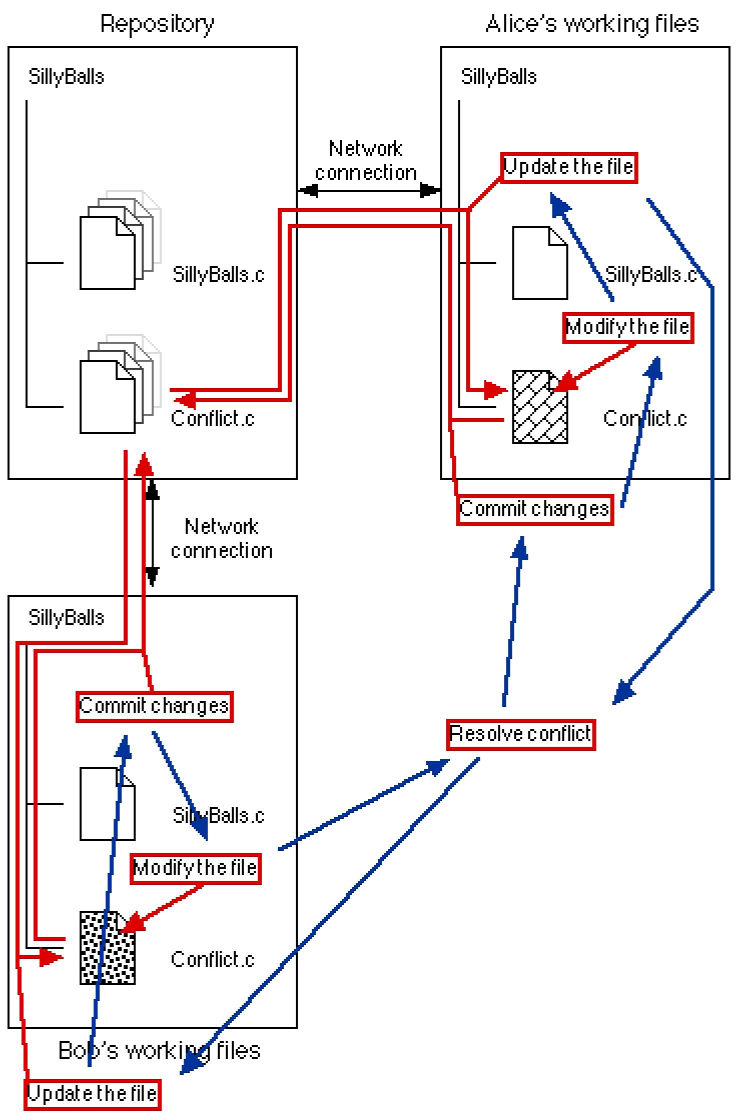
```
UInt32
CountStringsInList (
    Ptr     inData)
{
    /* Alice: added the assertion and educated Bob*/
    AssertIf_ (Ptr == nil);
    return *(UInt32*)inData;
}
```

after which she successfully commits her changes.

Having been through this, Alice and Bob learn some important facts about CVS update:

- if CVS update encounters a conflict and is therefore unable to update the file, it will mark the conflict in the file with 'conflict markers' (like in the above example). Fortunately, conflict markers do not compile in any known language, so they are hard to miss.
- if a CVS update fails, CVS will backup the working file before update - so that you can easily revert to it if you decide that you want to abandon the update. The backup file will be placed in the same folder as the file being updated, and it will inevitably have a bad, confusing filename, such as ".#Conflict.c.1.11".

As a result, they modify they work flow to be: modify - update - resolve conflicts - commit - repeat.



The "Copy-Modify-Merge" flow diagram with conflict resolve of CVS

13.2.5 Checkout: the missing link

One thing that this explanation swept under the rug was the very important question of how Alice created the initial copy of her working files. The initial act of acquiring a fresh copy of the files from the repository is called checkout, and is used only to create a complete new copy of working files on a developer's machine. Even if Alice or Bob remove some (but not all) the working files and folders on their development machines, they need to perform an update (and not checkout) to get new copies from the repository.

13.2.6 Conclusion

The model that has been developer above (modify - update - resolve conflicts - commit - repeat) does not require further modifications to be usable in practice. Some people might prefer to modify it slightly, but the basic ideas contained in this model always remain.

For example, Alice might choose to update her working files each morning as she comes to work, work on them until the afternoon, and then commit all her changes; of course, committing the changes will sometimes cause an up-to-date check to fail, so she will have to update her files again in order to commit, and possibly resolve some merge conflicts at that time. However, since up-to-date failures are not very frequent, and merge conflicts are even less frequent, this way of using CVS is perfectly reasonable, and does not deviate fundamentally from the modify - update - resolve conflicts - commit - repeat model.