Consistency Maintenance Framework For Collaborative Software Modelling Tools

Marta Lozano

A dissertation submitted to the University of Dublin,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

September 15, 2003

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: ________________

Date: September 15, 2003

Acknowledgements

I would like to thank my supervisor Alexis Donelly for his faith, trust, support and enthusiasm. Thanks to the M.Sc.NDS class, who made this year an interesting and unforgettable experience. Thanks to my family for their support on the most difficult moments, and thank to those who made financially possible this experience.

Abstract

The globalization of companies and business, and the improvements in communication and computing have lead to the need of new models of collaborative work. Real-time collaborative editing systems are included in the field of Computer Supported Collaborative Work (CSCW) systems, which allow users to view and design the same document simultaneously from geographically dispersed sites connected by networks.

Distributed Software Engineering (DSE) requires technical knowledge that spans geographical and organizational boundaries. In a distributed environment, developers are dispersed across different sites and even countries. Even thought major contributions have been lately introduced to enable CSCW applications on the Internet to support global collaboration, the area of DSE requires further research.

There are three inconsistency problems that arise in Collaborative Editing Systems: divergence, causal ordering violation and user intentions violation. Divergence can be solved serializing the operations at all sites, causality violation can be solved with a causal ordering communication protocol. However user intention violation solution is dependent on application semantics.

There are few group support framework specialized in DSE, and distributed software modeling. However, we are not aware of any Collaborative Software Modeling Framework using Consistency Maintenance mechanisms where the user intentions are preserved. Current distributed software modeling frameworks address concurrency with traditional methods as locking, turn taking, serialization, etc.

The algorithms and schemas presented in this work have been implemented in the **DArgoUML** prototype system. DArgoUML is a **D**istributed version of ArgoUML, which includes a Flexible Consistency Maintenance Framework based on Software Modeling Knowledge. The Framework can be considered Flexible as it allows the system to maintain temporal inconsistencies, as the shared document will merge to a consistent version. Some algorithms have been devised to detect different types of conflicts based on the different level of inconsistency they generate. Techniques have been presented to address each specific conflict or level of Inconsistency. Besides a mechanism for conflict group awareness is proposed, where users are aware of other user intentions when concurrent operations do conflict.

Table of Contents

Chapter 1
Introduction

1.1 Background

The globalization of companies and business, and the improvements in communication and computing have lead to the need of new models of collaborative work. Real-time collaborative editing systems are included in the field of Computer Supported Collaborative Work (CSCW) systems, which allow users to view and design the same document simultaneously from geographically dispersed sites connected by networks.

Distributed Software Engineering (DSE) requires technical knowledge that spans geographical and organizational boundaries. It has become a need for many organizations. New business models often result in distributed organizations that cannot be physically centralized in one location requiring developers to be dispersed across different sites and even countries.

If software development is viewed as a special case of collaborative editing systems then synchronous collaborative tools are needed to support synchronous work of different developers on the same artifact.

Even thought major contributions have been lately introduced to enable CSCW applications on the Internet to support global collaboration, the area of DSE requires further research.

Distributed software modeling, distributed software development, distributed requirements engineering, distributed project planning, distributed document management, distributed change management, distributed workflow management, software agents for software development… etc, can be included in the field of DSE.

There are three inconsistency problems that arise in Collaborative Editing Systems: divergence, causal ordering violation and user intentions violation. Divergence can be solved serializing the operations at all sites, causality violation can be solved with a causal ordering communication protocol. However user intention violation solution is dependent on application semantics.

There are few group support framework specialized in DSE, and distributed software modeling. There are very few Collaborative Editing Systems supporting Consistency Maintenance (convergence, causal ordering and user intentions). We are not aware of any collaborative software modeling framework employing Consistency Maintenance mechanisms where the user intentions are preserved. Current distributed software modeling frameworks address concurrency with traditional methods not very appropriate for collaborative work as locking, turn taking, serialization, etc.

The REDUCE (REal-time Distributed Unconstrained Cooperative Editing) project aims to research, develop, and apply innovative technologies for *consistency maintenance* and *concurrency control* in real-time Collaborative Editing systems. Under the REDUCE project Collaborative Text, Graphics and Programming systems have been researched. REDUCE project is actually running in Griffith University, Australia. This has been the main research and ideas source for my thesis. The Consistency Maintenance Framework devised for the Collaborative Graphic Editing System have been modified and extended for supporting Collaborative Software Modeling.

The algorithms and schemas presented in this work have been implemented in the **DArgoUML** prototype system. DArgoUML is a **D**istributed version of ArgoUML (a open source Software Modeling tool). A Flexible Consistency Maintenance Framework based on Software Modeling Knowledge has been included in DArgoUML. The Framework can be considered Flexible as it allows the system to maintain temporal inconsistencies, as the shared document versions will merge to a consistent version. Some algorithms have been devised to detect different types of conflicts based on the different levels of inconsistencies they generate. Techniques have been presented to address each specific conflict or level of Inconsistency. Besides a mechanism for conflict group awareness is proposed, where users are aware of other user intentions when concurrent operations do conflict.

1.2 Motivations

As introduced in the previous section, the area of Distributed Software Engineering requires more research. Distributed Software Development is becoming a fact in many organizations however there are few tools that support this collaborative work.

Collaborative editing systems are special distributed systems because of the human interaction. Traditional concurrency control mechanisms as locking, turn taking or serialization are not very appropriate for collaborative editing system. There are some new techniques for concurrency control more appropriate as operational transformation or multiple object creation. These techniques have been applied mainly to text and graphics environment. One of the main motivations has been the analysis of how these new concurrency control techniques could be applied to applications with richer semantic information, as is software modelling tools. Just to see how the semantics are richer: in a UML model there are two types of information: graphical information and UML information. For the UML information the Abstract syntax and the Well-Formedness rules restrict the behaviour of the possible operations that can be applied to the model.

There are very few collaborative software modelling tools and research projects. The existent ones use traditional methods for concurrency control. DArgoUML extends open source ArgoUML for supporting distribution.

1.3 Roadmap

This section describes briefly each of the remaining chapters contained in this dissertation.

1.3.1 State of the Art

Computer Supported Collaborative Work, CSCW State of the Art is addressed in this Chapter:

In the first section the CSCW concept is explained giving some popular definitions and then the CSCW applications are categorized based on two dimensions, time and place. Some CSCW concepts as Group Management, Concurrency Control mechanisms, etc are introduced.

In the next section some Collaborative Editing Systems that address consistency maintenance are presented.

Then a non traditional Concurrency Control system is presented: A Consistency Maintenance Framework where all user Intentions are preserved. Many interesting ideas for my work have been generated through the inspection of this set of techniques.

Finally the State of the art in Software Modelling tools is presented: The standards they use, mechanisms for modelling, the sharing of the UML models and a recent tool and few research projects for real-time cooperation!

1.3.2 Architecture

In this chapter the whole process of extending the single-user open source software modelling tool into a version supporting distribution is presenting. Then the architecture of the Consistency Maintenance component is presented. The next section describes the two channels of communication among collaborating peers: The Sharing channel for transferring the complete application state and the collaboration channel for real-time operations. In the next section the component that translates the operations from/to the network is explained. Latecomer support and the support of JavaGroups for CSCW systems are finally presented.

1.3.3 Concurrency in Collaborative Editing Systems

This chapter has two main sections, the first one is a classification of operations based on their concurrent behaviour, and the second section addresses the issue of concurrency detection.

1.3.4 Consistency Maintenance Framework for Collaborative Software Modelling tools.

This chapter presents a Flexible Consistency Maintenance framework for Collaborative Software Modelling tools.

First the three inconsistency problems and their solutions are again presented. User Intention violation can not be solved with a generic solution as the other two: divergence and causal ordering violation. For preserving user intentions a solution based on application semantics is devised. In the environment of Software Modelling tools, application semantics includes: UML information, Graphical Information and Restriction Rules based on UML Specification and Time-Line dependencies.

Finally a method for detecting different types of conflicts based on the level of inconsistency produced and the category of the operations that generated it, is presented. The Matrix for Conflict detection is the core of the framework. Then types of conflicts and their possible resolutions are addressed.

1.3.5 Implementation

This chapter includes the description of the implementation process of the elements described in the architecture chapter.

1.3.6 Conclusions

In the Conclusions chapter first I summarize what has be done in this thesis, followed for what has been achieved and finally some future work suggestions.

Chapter 2

State of the Art

2.1 CSCW: Computer Supported Collaborative Work

2.1.1 Introduction.

The globalization of companies and business and the improvements in communication and computing have lead to need of new models of collaborative work (software). In a distributed environment, developers are dispersed across different sites and even countries. CSCW systems enable geographically dispersed participants work together.

CSCW are special Distributed Systems because of the Human Computer Interaction. As result of this Interaction some factors acquire special relevance as Group Awareness, Multi-User interfaces, Concurrency Control and Group Communication and Coordination.

There are many definitions for CSCW and Groupware, some of the most popular are:

> Ellis [1]defined Groupware as "*Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment.*"
>
> According to Brinck "CSCW is the study of how people work together using computer technology. Typical topics include use of email, hypertext that includes awareness of the activities of other users, videoconferencing, chat systems, and real-time shared applications, such as collaborative writing or drawing." [2]

Groupware refers to the technology that people use to work together, the real computer-based systems, the hardware and software which supports group work while *CSCW* refers to the field that studies the use of that technology as well as their psychological, social and organizational effects. CSCW is concerned with the study theory of how people work together and how groupware affects the group behavior.

2.1.2 Classification of CSCW in Time and Place

There are four situations in which groups may work together. This distinction was made first by Johansen [3]. The different types of cooperation arise from the combination of two dimensions: time and place. Regarding with time dimension the communication can be synchronous or asynchronous:

Synchronous Communication or Collaboration: Real- Time systems that allow participants see each other changes immediately. Group of users can cooperate at the *"same time"*.

Asynchronous Communication or Sharing: Participants cooperate over *"different periods of time"*. For example a user could edit a document and other user could make some annotations over the document afterwards.

For instance in the field of Software Modelling tools a user could create a UML model. It could be transmitted to another user in XMI format. Second user could update the UML model and send it again to the first user. This would constitute *Asynchronous Communication.* While in a *Synchronous Collaborative Software Modelling tool,* several users could model a software system together, and the modifications on the UML elements would be seen immediately by all the collaborators.

2.1.3 Common CSCW Applications

2.1.3.1 Message Systems

Message Systems are email-enabled software applications. These systems use textual messages as interchange format for communicating with group of users. There are several types of message systems:

- ***Email:*** Allows the transmission and reception of electronic messages, which consists on various fields including the recipient, sender, subject matter and body of the message. Email systems provide a framework with functionalities for the creation, view, and management (storage, deletion) of messages, including additional functionalities as attach/insert file to a message. Email is the most well known message system today.

- ***Newsgroups:*** are similar in concept to email systems except that they are intended for messages among large groups of people instead of one-to-one communication. For communicating, messages are left in a newsgroup. Any reader belonging to that group can access the messages.

- ***Chat systems:*** A system that allows any number of logged-in users to have a typed, real-time, on-line conversation, with other users logged via a network.

2.1.3.2 Computer Conferencing capabilities

Possible, thanks to the increase of communication bandwidth and video compression techniques to allow real-time and sound links between remote sites. Meetings can be arranged among geographically dispersed sites in which participants are able to see and hear each other and thus work together.

2.1.3.3 Shared diaries and Calendars

Provide support for the arrangement and organization of meetings. Group members record their individual appointments and schedules in the electronic diary/calendar. If somebody wants to arrange a meeting all available dates may then be discerned.

2.1.3.4 Bulletin boards

Capabilities for storing messages and files. Discussions may be arranged around topics of interest and anyone with access may read all messages on a particular topic left by others or add messages to the topic. Forums and discussion groups allow users to post messages but don't have the capacity for interactive messaging.

2.1.3.5 Application Sharing Systems

Allow participants to share an ordinary single-user application. (Word processor or spreadsheet). The application run on a workstation.

2.1.3.6 Collaborative Editing Systems

Or CES are multi-user editors; allowing members of a team (that can be dispersed geographically), work on the same document concurrently (real-time). There are synchronous and asynchronous cooperation models. Examples of this type of systems are collaborative word processors, graphic system (white boards), programming system (kind of text processor), software modeling system. This thesis is about synchronous software modeling system, that allows dispersed users work together in a UML model when designing a software system.

2.1.4 CSCW Keywords

2.1.4.1 Group Awareness

Being aware of other users' locations, activities, and intentions relative to the task enables people to work together more effectively. There are different types of awareness, for many kinds of systems is very useful to know who is collaborating at some stage in the session, who has done what, what are the future intentions of the users, when there are concurrency conflicts, who are the participants in a conflict and why the conflict has been generated. A shortcoming of an over awareness information may be the violation of users privacy.

2.1.4.2 Multi-User interfaces

Multi-user interfaces are different than single-user interfaces. However that difference should not be very high. In [9] the gap in terms of usability between single-user editor and multi-user editor is presented: Users are forced to learn new Interfaces and new ways of working. The ICT "collaboration Transparency" project aim to transform single-user applications into multi-user applications without changing too much the application code. In a multi-user interface information has to be added in the shape of group awareness so users can cooperate effectively.

In this thesis a single-user application for software modeling has been transformed in a multi-user application. The user interface has hardly changed so the level of usability is high. Besides conflict awareness information has been added to reflect all *user intentions*.

2.1.4.3 Concurrency Control

In a collaborative system participants are generating operations concurrently. Some of these operations can conflict with each other. A concurrency control mechanism is needed to resolve inconsistency problems. Because of the *"human interaction"*, CSCW systems are special distributed systems. Therefore, many concurrency control techniques which worked well in traditional systems are not appropriate for CSCW systems. Thus, new concurrency control techniques need to be developed.

2.2 Collaborative Editing Systems

As it has been introduced in 2.1.3.6 Collaborative Editing Systems CES are included in the field of CSCW. CES allow members of a team (that can be dispersed geographically), work on the same document concurrently (real-time).

This section describes an overview of some CES and the mechanisms employed to address concurrency problems

2.2.1 REDUCE PROJECTS (Real Time Distributed Unconstrained Cooperative Editing)

Variety of collaborative projects, initially REDUCE was focused on Collaborative Text Editing. [17]

A brief overview of some REDUCE projects is presented as for these systems a Consistency Maintenance Framework has been devised, where the three inconsistency problems in a CES are addressed: divergence, causality preservation and user intention violation. Intention preservation property can only be achieved by application dependent mechanisms. So a different mechanism is proposed in text editing and graphics editing.

The work and research done for REDUCE projects have been deeply examined and used as basis for important ideas for this thesis.

For the ***Text Editing environment*** an optimistic approach to concurrency control called operational transformation is proposed. The novelty of this schema is that it allows independent operations to be executed in any order but ensures that their final effects are identical and the intentions are preserved. [4]. They show how Intention-Preservation achieved by operational transformation is not achievable by any traditional serialization protocol. ***User Intention Preservation depends on Application Semantics.***

REDUCE collaborative technologies and systems have been applied to other areas as Collaborative Graphic Editing, GRACE, or collaborative programming RECIPE.

Consistency maintenance mechanisms applied to REDUCE and GRACE are deeply addressed in this chapter in section 2.3.

2.2.1.1 GRACE (Graphics Editing System)

GRACE is a prototype of a collaborative graphics editing system [18]. In GRACE, a novel mechanism for preserving user intentions has been presented: [5] A Multiple Object Version Scheme. In this scheme when a conflict is detected different versions of the targeted object are created. This schema has the property of minimizing the number of versions created combining properly compatible versions in the same version. GRACE Consistency Maintenance mechanisms are addressed deeply in this chapter in section

2.2.1.2 RECIPE (A prototype for Internet-based real-time collaborative Programming)

RECIPE [19] is an Internet-based real-time collaborative programming system that allows physically dispersed programmers to concurrently and collaboratively design, code, test, debug and document the same program. [7].

In RECIPE a Hierarchical collaboration schema is presented: The prototype can share the compiling applications, debugging applications or even the entire Unix shell application for collaboration. There are four types of sharable sessions in the RECIPE prototype system, that is Unix shell session, Edition session, Compiling session and Debugging session. RECIPE prototypes uses REDUCE techniques for supporting real time cooperative editing.

2.3 Consistency Maintenance Framework

2.3.1 Achieving Convergence, Causality Preservation and Intention Preservation in Real-Time Cooperative Editing Systems

"Real-time cooperative editing systems allow multiple users to view and edit the same text/graphic/image/multimedia document at the same time from multiple sites connected by communication networks. Consistency Maintenance is one of the most significant challenges in designing and implementing real-time cooperative systems…" [4].

In [4] a consistency model, with properties of convergence, causality preservation, and intention preservation, is proposed as a framework for consistency maintenance in real-time cooperative editing systems.

The cooperative editing systems subject to their research have the following *properties*:

- ***Real-time***: *"Local response is quick (almost as in a single-user application) and independent from network latency, and the latency for reflecting remote actions should be low (dependent on network latency)"*

- ***Distributed***: *"Cooperating users may reside in different machines connected by different communication networks with non deterministic latency"*.

- ***Unconstrained***: *"Multiple users are allowed to concurrently and freely edit any part of the document at any time"*.

A *replicated architecture* is proposed as the only solution for accommodating the properties mentioned above especially for good responsiveness and unconstrained behaviour. In a replicated architecture there is no central server, the shared document is replicated at all sites. *"One of the most significant challenges in designing and implementing real-time cooperative editing systems with a replicated architecture is consistency maintenance of replicated documents"*

In [4] consistency maintenance model the three inconsistency problems that appear in real-time collaborative editing systems, as well as the properties they violate and their solutions is presented:

- ***Divergence:*** *"Operations may arrive and be executed at different sites in different orders, resulting in different final result. Unless operations are commutative final editing results would not be identical among cooperating sites"*. The Divergence problem can be solved by any *serialization protocol*.
- ***Causality Violation:*** *"Due to the non deterministic communication latency, operations may arrive and be executed out of their natural cause effect order"* The causality violation can be solved with a *Causal Order Communication Protocol (Causal Order Multicasting)*.
- ***User Intention:*** *"Due to concurrent generation of operations, the actual effect of an operation at the time of its execution may be different from the intended effect of this operation at the time of its generation"*. User intentions violation can be solved with *application dependant mechanisms*.

For a formal description of the consistency model some definitions are presented:

2.3.1.1 Definition 1.1. Causal ordering relation '$\rightarrow$'

Given two operations OP_1 and OP_2, generated at sites I and J, then $OP_1 \rightarrow OP_2$ if:

1. I = J and the generation of OP_1 happened before the generation of OP_2 or
2. I ≠ J and the execution of OP_1 happened before than the generation of OP_2.

2.3.1.2 Definition 1.2. Dependent and Independent operations

Given two operations OP_1 and OP_2:

1. OP_2 is said to be *causally dependent* on OP_1 if only if $OP_1 \rightarrow OP_2$.
2. OP_1 and OP_2 are said to be *causally independent* if and only if neither
3. $OP_1 \rightarrow OP_2$, nor $OP_2 \rightarrow OP_1$. It is expressed as $OP_1 \mid\mid OP_2$.

2.3.1.3 Definition 1.3. Intention of an Operation

The Intention of an operation OP_1 is the Execution effect, which can be achieved by applying OP_1 on the document state from which OP_1 was generated.

2.3.1.4 Definition 1.4. A Consistency Model

A collaborative editing system is said to be consistent if it always maintains the following properties:

a) ***Convergence:*** When the same set of operations have been executed at all sites.
b) ***Causality Preservation:*** For any pair of operations OP_1 and OP_2, if $OP_1 \rightarrow OP_2$ then OP_1 is executed before OP_2 at all sites.
c) ***Intention Preservation:*** For any operation OP_1, the effects of executing OP_1 at all sites are the same as the intention of OP_1, and the effect of executing OP_1 does not change the effects of independent operations.

As it was said before convergence can be achieved with total ordering, causality preservation with a causal ordering protocol, and intention preservation cannot be achieved with a generic solution, the solution to user intentions violations is application dependent which means that it will be solved differently depending on the application.

2.3.2 User Intentions Preservation in Collaborative Graphics Editing Systems

User Intention Preservation mechanisms for Collaborative Graphics Editing systems has been examined as these systems can be used for CAD and CASE tools to draw design diagrams, or to draw illustrative figures within documents collaboratively.

Chen and Sun ideas for maintaining consistency in real-time collaborative graphics editing system, [5], have been examined deeply, and considered the starting point for the designing of *Consistency Maintenance Framework for Collaborative Software Modelling tools.*

As it was stated in the previous section, *"the inconsistency problem which needs to be solved is intention violation caused by the execution of conflicting operations"*

Chen presents a new mechanism for preserve all users intentions in the Graphics Environment: A Multiple Object Version Scheme. In this scheme when a conflict is detected different versions of the targeted object are created. This schema has the property of minimizing the number of versions created combining properly compatible versions in the same version.

2.3.2.1 Notation

First a notation is introduced to give a precise definition of operation conflict, and following the formal definitions of conflict and compatible relation among operations: [5]

Target (OP_1): Object being targeted by the operation OP_1.
Property.Type (OP_1): Attribute of the object that is targeted by OP_1.
Property.Value (OP_1): The new value for the attribute to be updated.

2.3.2.2 Definition 2.1. Conflict Relation $\otimes$

Given two operations OP_1 and OP_2 they conflict with each other $OP_1 \otimes OP_2$ if:

1. $OP_1 \mid\mid OP_2$.
2. Target (OP_1) = Target (OP_2)
3. Property.Type (OP_1) = Property.Type (OP_2).
4. Property.Value (OP_1) $\neq$ Property.Value (OP_2).

2.3.2.3 Definition 2.2. Compatible relation ⊙

Given two Operations OP1 and OP2, if they do not conflict with each other, they are compatible, expressed as $OP_1 \odot OP_2$.

2.3.3 Accommodating all operation effects

When a conflict is detected how do actual systems accommodate all user intentions:

- ***Null effect:*** Neither of the conflictive operations has a final effect on the target object. This can be achieved by rejecting/undoing an operation when it is found to be conflicting with another operation. This Null effect does not preserve any user Intention. The consequence of this intention violation is that, whenever there is a conflict, the work concurrently done by involved users will be destroyed, which is highly undesirable in the collaborative working environment. When a conflict occurs, the involved users are provided with no explicit information about what the other users actions were or intentions might be.

- ***Single operation effect***: Retain the effect of only one operation. This can be achieved by enforcing a serialized effect among all operations. The final results at both users sites are identical. The user Intentions are not preserved and only one user work can be preserved.

- ***All operations effect***, based on multiple versions strategy [17, 102, 100] two versions of the object will be created. In this way the effects of both operations are accommodated in two separate versions. The side effect of this approach is that the single version object may be converted to multiple versions if a conflict occurs.

2.4 Collaboration and Sharing among UML Tools

2.4.1 Introduction

Collaborative Software Modeling tools are included in the category of Collaborative Editing Systems where participants in the collaborative group update a shared document. In a Synchronous system users can see immediately the changes introduced by other users. The shared document is a UML model, usually stored in XMI format.

The Unified Modeling Language (UML) [21] is a general-purpose notational language for designing software systems. UML has become the industry *standard language* for modeling software systems and *communication*.

Communication is the main benefit about using a standardized language. With UML precise definitions can be constructed, understood, and interchanged among Software Modeling tools and Professionals.

UML CASE tools provide a framework to help designers specify, construct and visualize complex systems using UML. There are many UML tools, to mention a few:

> *Rational* are market leaders in the UML CASE tools .UML was devised by Grady Booch, James Rumbaugh, and Ivar Jacobson within Rational. Rational offers as well a Unified Process (RUP) used throughout the software lifecycle.
>
> *TogetherSoft* developed one of the first UML CASE tool to be totally *integrated* with the code. The tool can do reverse engineering obtaining a model from the code.
>
> *Cittera* supports *real-time collaboration* so that multiple parties across various geographic locations are able to work together, simultaneously, on the same model.
>
> *ArgoUML* is an Open Source Development Project and a free UML modeling tool. It has a commercialized extension widely used called Poseidon for UML. (ArgoUML is the starting point for the construction of the Multi-User Synchronous Collaborative Software Modeling Tool, in other words Real-Time Group UML Tool).

2.4.2 XML Metadata Interchange: XMI

When UML first appeared, there was no standard format for interchange of UML models; most individual tools had their own proprietary format.

XMI, the OMG's XML Metadata Interchange format [20], is a vendor independent format for saving, loading UML models, as well as import/export information from/to other UML tools. *The interchange format enables heterogeneous and homogeneous UML tools to share model information*: So, a UML model created with Rational UML tool should be understood by other case tools as Together Soft tools.

However XMI is a relatively recent standard that is co-evolving and settling down with UML. Different combinations of UML versions and XMI versions exist: only an exact match will enable tool-to-tool interchange. (XMI 1.0, 1.1, UML 1.1, 1.3, 1.4).

A big advantage of XMI being based on XML is that the whole range of generic XML tools is available. In [14] a new perspective for managing UML models is presented using the fact that they are saved into and loaded from XMI format. The main idea is that some tasks are easier to carry over the XMI file instead of over the UML Case tool. For example with the use of an XML parser, the "visibility" of all the "public" attributes of the classes belonging to a package could be changed into "private". This task could also be performed using UML case tool functionality but it will be slower as it has to be done element by element.

XMI includes only UML information the diagram layout it is lost. In current specifications of UML 1.x, the metamodel definition does not include sufficient details to include graphical and diagram information necessary to represent and interchange the diagrammatic aspects UML models in an interoperable manner. This has resulted in a number of proprietary extensions to UML and by implication proprietary XML/XMI DTDs causing information loss when UML models are exchanged between tools.

2.4.3 Evolution of UML modeling tools: Collaboration and Sharing.

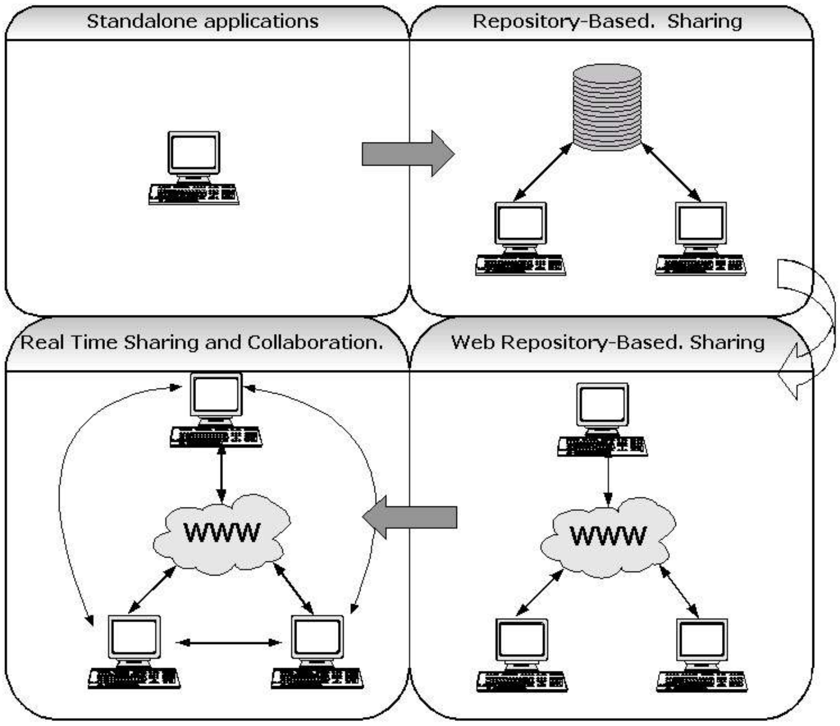
UML Case tool have evolved from Standalone applications where there were no exchange of information, to Real Time Sharing and Collaboration where tools can not only interchange UML information asynchronously but also they can interact in a Real Time manner.

Standalone tools: No Sharing Information.

***Repository-based Model Sharing*:** The Information can be shared among clients. If the repository is not proprietary the information could be shared among heterogeneous Clients. The Information on the repository is not Real-Time.

Web-based Model Sharing: The proprietary repository can be accessed on the Internet.

Real-Time Model Sharing & Collaboration: The different clients can access not only to the repository Information but they can see as well Real Time operations of the others participants.


Standalone applications
Repository-Based. Sharing
Real Time Sharing and Collaboration.
Web Repository-Based. Sharing
WWW
WWW

1. **Figure 2.1Collaboration and Sharing among UML Tools**

Few tools and projects for working in a Real-time Collaborative manner have appeared recently (Cittera [22], Dmeeting [12], The knight Project [13]) however consistency maintenance is achieved with traditional methods that reduce or even avoid concurrency (floor control mechanism, serialization). In these concurrency control mechanisms not more that one user can manipulate the same object at the same time, or with serialization only one-user intentions would be preserved and the operation to be preserved is decided by the system, usually this decision is taken without application dependant information.

A new stage on the evolution of UML CASE tools is needed with Consistency Maintenance mechanisms where all users intentions can be preserved:

Real-Time Collaborative tools will generate conflicts when different participants generate conflictive operations. A consistent application dependant mechanism should be used in order to detect, and manage the conflicts generated as result of concurrency.

2.4.4 ArgoUML: Starting point for a Synchronous Collaborative UML Tool Prototype

These are the reasons for Choosing ArgoUML[23] as starting point for constructing a multi-user UML Case tool:

2.4.4.1 ArgoUML supports standards extensively

Standards: UML 1.3 Specification, XMI, SVG, OCL and others, in this respect, ArgoUML is still ahead of many commercial tools [24]:

- ***Open Source*** Project originally developed by a small group of people as a research project
- ***UML is itself an open standard***. ArgoUML use open standards for all its interfaces: The key advantage of open standards is that it permits easy inter-working (in order to avoid fascist systems) between applications, and the ability to move from one application to another as necessary.
- Instead of using a self-implemented UML meta-model, ArgoUML uses a meta-model implementation provided by NovoSoft.
- ***XML Metadata Interchange (XMI)***: Is the standard for saving meta-data that make up a particular UML model. In theory this will allow you to take the model you have created in ArgoUML and import it into another tool.
- ***Graphic Vector Standards:*** ArgoUML saves diagram layouts using PGML (Portable Graphics Markup Language) an earlier proposed standard than SVG, however the tool has as well functionality for exporting the diagrams in this format. SVG format is to be included in the next versions as format for saving/loading diagram layouts.

2.4.4.2 Usability of the UML CASE Tool

Group tools do not always success in terms of usability. Users familiarized with their single-user editors are forced to learn new interfaces. As is presented in [9], there is a *gap in terms of usability between group editors and familiar single-user editors*. One solution to mitigate this problem is called application sharing or collaboration transparency, in which existing single-user applications are converted into groupware applications.

ArgoUML is a single-user UML CASE tool quite extended. It is a familiar tool, so participants in the new collaborative prototype do not need to employ effort learning new interfaces. The aim is to transform the well-known existing single-user tool into a group tool.

2.4.4.3 Unique Generation of ID for networked systems

The elements of the UML model have associated an identifier, which has to be unique all along the sites participants in the session. Two sites will generate a different set of Unique Ids. ArgoUML achieve this property generating Ids using the network address of the site.

Chapter 3

Architecture

3.1 Single-user Software Modelling Tool: ArgoUML

ArgoUML, a single-user software modeling tool is the starting point of the Collaborative Software Modeling prototype. In this section a basic explanation about the architecture of ArgoUML is presented before addressing in following sections the architecture of the collaborative prototype. Figure 3.1 is a basic description of ArgoUML components:

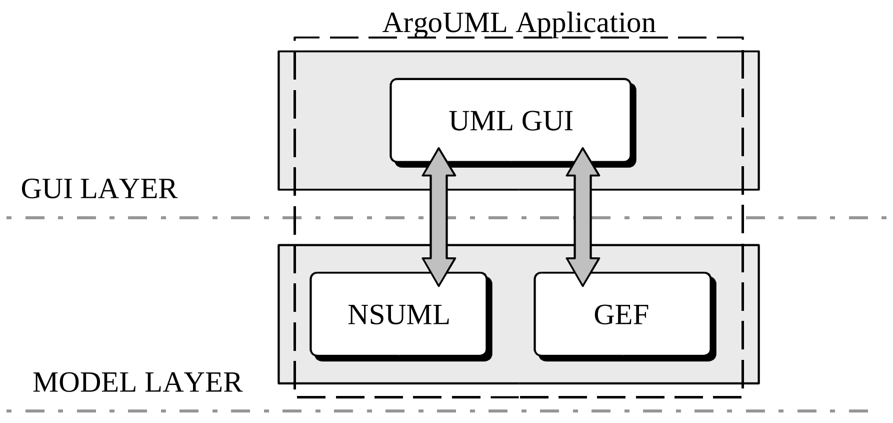


Figure 3.1ArgoUML Components

Until now UML Specification does not include graphical information to represent diagram layout. As result of this lack, every UML tool has specific (proprietary) methods for storing graphical Info.

UML tools manage two types of information regarding with UML diagrams: UML model and graphical model. The UML model includes all the data related with UML elements, as Class definitions (attributes, operations, modifiers), Interface definitions, Relations definitions (cardinality, aggregation mode). The graphical information necessary to represent those elements in the Graphical User Interface (localization, size..) is included into the graphical model.

ArgoUML uses for both UML model and graphical model external implementations:

> ***NSUML:*** A Complete Implementation of the UML Specification 1.3, NSUML, has been implemented by Novosoft. The implementation is being used by other projects as well. The implementation includes de definition of the *UML elements* (Classifiers, GeneralizableElements), and the *abstract syntax* and *Well-Formedness rules* of the UML Specification.
>
> ***GEF:*** Graphical model for representing the UML model. It constitutes the diagram layout. A UML Class Diagram is interpreted as a graph, where the classes and interfaces are *nodes*, and the relations (associations, generalizations, realizations and dependencies) are *edges*. It has been implemented by Tigris. OJO references

User interface events update both models:

> OP_1: CREATE CLASS AT (X, Y): Updates UML model and graph model.
> OP_2: MOVE CLASS TO (A, B): Updates only graph model.

ArgoUML Application*:* ArgoUML Graphical User Interface enable users manipulate both UML model and graphical model. ArgoUML stores the UML model in XMI format and graphical model in PGML. However ArgoUML can export the graphical model into SVG format. ArgoUML has functionality added to the User Interface as the Cognitive Support, a mechanism for advising designers about specific solutions based on patterns design.

3.2 Distributing ArgoUML

There is a gap in terms of usability between single-user editors and multiple-user ones. This gap comes from the fact that in multiple-user applications, users are forced to learn new user interfaces. Collaboration Transparency Project [9] aim, is to translate existing single user application into multiple user application without changing the application code.

The idea of the "gap in terms of usability" has been taken into account for the design of the collaborative prototype. Users that were familiarized with ArgoUML can use the prototype with the same easiness. The only change in the user interface is the addition of conflict awareness information.

Figure 3.2 shows the components added to transform the single-user application into the multiple-user application.

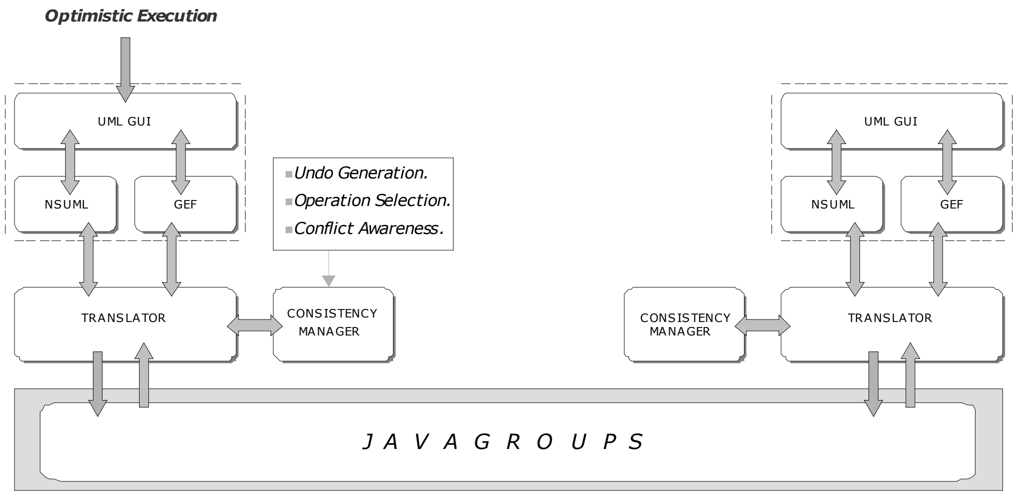


Figure 3.2Distributing ArgoUML

The architecture is fully distributed, *"Replicated Architecture"*, there is no central server. The shared data is replicated at all sites.

Replicated Architecture* + *Optimistic Execution* = *Fast Response Time

Combining a replicated architecture with an optimistic execution the application will have a fast response time independent of network latency. *Optimistic execution* means that the events generated locally are executed immediately in the UML and Graphic model, and afterwards the events are sent to the network. The response time is a very important factor in systems with Human Interaction, as the response time has a high influence on the users perception about the quality of the tool.

With a central server and a pessimistic execution the response time is not as fast, and it depends on the latency of the network: from the moment a user generates an event until it appears on the screen some operations occur involving transmission through the network.

First the event has to be sent to the server, the server takes a decision, and the server multicast the event to all the collaborators. The user could think the tool is not working very well instead of realizing all the communication process that lies behind.

The only *shortcoming of optimistic execution is that the Consistency Maintenance mechanisms are more complicated*, as some of the local operations executed immediately may conflict with other concurrent operations generated at other sites concurrently. In some cases some undo or transformation mechanisms will be applied.

The main components that have been added to obtain the distributed architecture are:

> *Translator:* Translate local events into a serializable format to be sent through the network and transforms the remote operations into changes in both local UML and Graphical models.
>
> *Consistency Maintenance component:* assures the UML model is consistent among all sites.
>
> *Communication Platform:* Responsible for transmitting all the Real-Time operations, as well as the whole application state for "Latecomer support".

3.3 The Consistency Maintenance Component

The Consistency Maintenance component assures that the UML model is *consistent among all sites these means the consistency maintenance properties are maintained (convergence, divergence and User Intentions)*. However the consistency model has been designed to be flexible and allow the system to maintain temporal inconsistencies that at some point in time will merge into a unique and consistent version. *Always final results will be Consistent.*

The Consistency Maintenance component has the following properties:

> *Uniqueness of the Conflict Resolution: Conflicts should have the same resolution at all sites.* In the face of a conflict, all sites should perform the same conflict resolution. With a central server, the events would be serialized, or the server would choose following some priority schema one of the possible solutions. *In a Peer-to-Peer architecture, conflicts should be resolved locally, and the result has to be Unique among all sites.*

Fairness of the Resolution: One solution in order to obtain Uniqueness on Conflict resolution could be giving priority to the users based on their network address. (For example: Highest network address → highest priority). The resolution is unique but is not fair, in this case the events launched by some user having the highest network address will always prevail over the other users actions. The fairness of the resolution is achieved with application semantics information. There are some solutions:

Rules Engine: deciding which operation/operations will prevail.

Randomly: (For example in some games)

App dependent Priority Levels: In the application the users have assigned a priority level. This priority mechanism is based on application semantics, not like prioritizing based on some non-application semantic information as the network address example mentioned before.

The Consistency Maintenance Component for the collaborative prototype has a rule engine to take resolutions in the face of a conflict. The resolution will be unique among all sites.

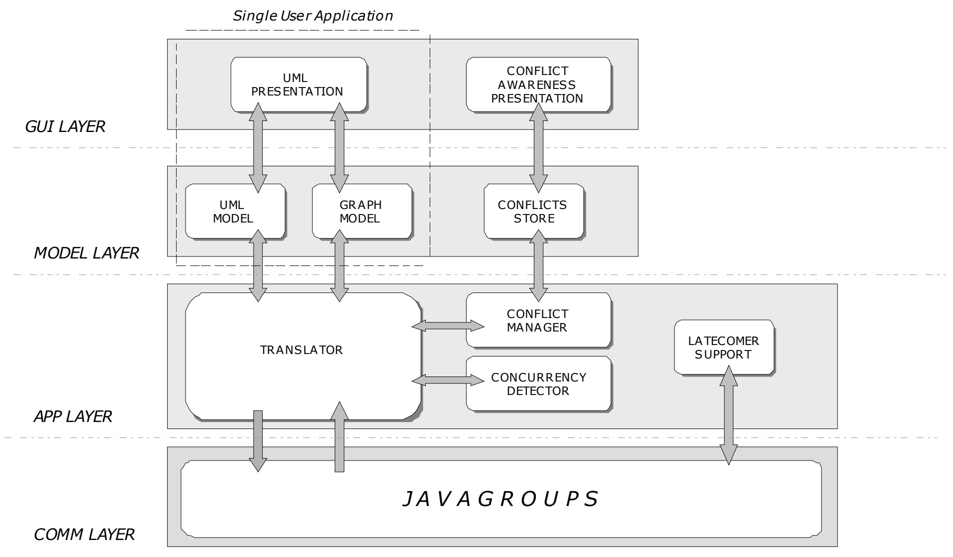


Figure 3.3Architecture of a Collaborative Peer

There are four main subcomponents:

- *Concurrency Detector.*
- *Conflict Manager.*
- *Conflict Store.*
- *Conflict Awareness Presentation.*

3.3.1 Concurrency Detector

Conflicts can occur only among concurrent operations 4.2.2 Independent operations.. For every operation received the component will detect the set of operations that have a concurrent relation, among the operations received and the local ones (optimistically executed). There are different mechanisms for detecting concurrency. In 4.3 , a concurrency detection mechanism when using total ordering or causal ordering protocols have been proposed.

The set of concurrent operations and possible conflicting ones is examined by the Conflict Manager.

3.3.2 Conflict Manager

The conflict manager consists on a rule engine that detects different types of conflicts and for each conflict type performs a specific resolution. The conflict Manager maintains the Conflict Store Information. Conflict manager rules and resolutions are deeply addressed in 5. Consistency Maintenance Framework.

3.3.3 Conflict Store

Data store for each element in the UML model with at least a conflict associated. For each conflictive element there is a list of properties that caused the conflict, and for each property the participants on the conflict are specified, as well as the values and/or operations the participants were trying to perform.

3.3.4 Conflict Awareness Presentation

There is conflict awareness information associated with the graphical model and with the UML model. When an element has at least a conflict, its appearance is modified so the collaborators are aware that a conflict has occurred over that element. (In the prototype conflictive elements are marked with a red shadow (the nodes and edges of the graph). If an element is conflictive (red shadow) the conflict information can be consulted (mouse right button). The information appears categorized by property.

3.4 Communication Platform

Peers in the collaborative prototype can communicate with each other over two channels:

3.4.1 The Sharing channel

Enables the transmission of the application state. The transmission of the whole current UML project occurs at the beginning of each collaborator session: When a newcomer joins the group, it sends a request to one of the members of the group that responds sending the complete UML Project. This newcomer support is known as ***Latecomer support.*** [11]

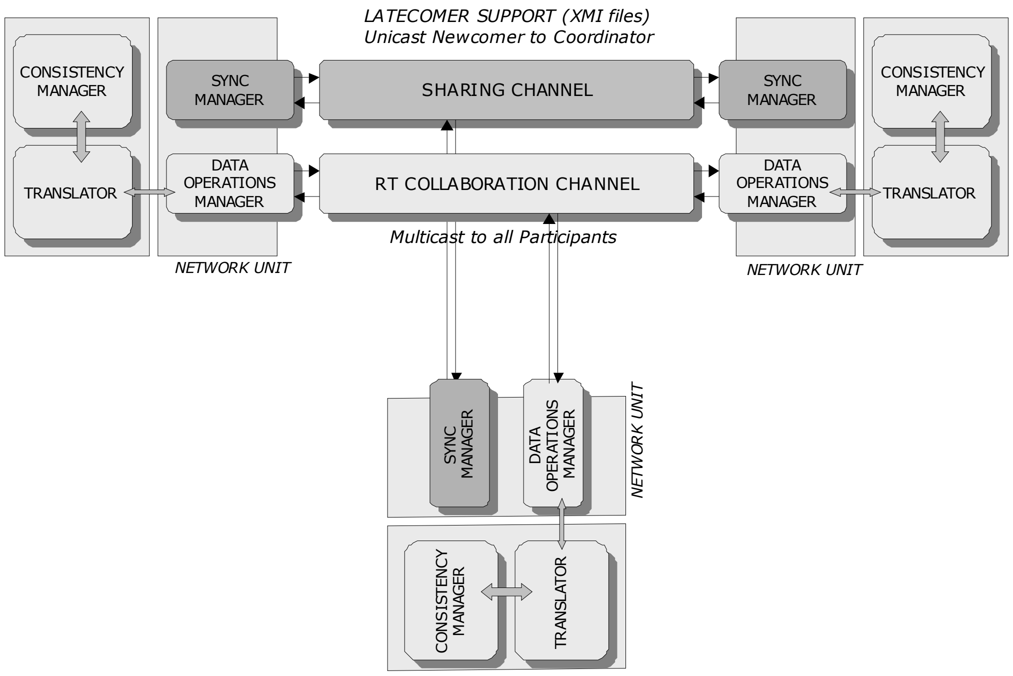


Figure 3.4Communication Architecture

The application state is represented in XMI format for the UML model information, and PGML for the diagram layout.

The communication on this channel is unicast and synchronous: unicast meaning that the transmission is between a pair of collaborators (The coordinator of the group, and the newcomer). And synchronous as the newcomer keeps waiting until the reception of the application state.

3.4.2 The Collaboration channel

Used for the transmission/reception of all the real time operations generated by the collaborators during the active Session. So the Translator on each collaborator peer receives the remote events from all other users and propagates them into the local model, and in the same fashion it distributes remotely the local operations.

Users can collaborate working over the same UML Document. The communication in this channel is multicast and asynchronous. All the peers will receive all the events.

3.5 Integrating Application Layer with Communication Platform

The application layer is integrated with the communication platform through networks units on each peer. Each network unit has a *DataOperationManager* component and a *SynchronizationManager* component.

The *DataOperationManager* is responsible for sending/receiving the Real-Time operations in a asynchronously fashion. The *SynchronizationManager* is responsible for sending/receiving the complete application state.

The *DataOperationManager* and the *SynchronizationManager* use JavaGroups utility classes. Some of the utility classes are *PullPushAdapter* and *MessageDispatcher,* these classes characterizes the communication as being synchronous and asynchronous, and implement network listener. This is addressed deeply in chapter 6   Implementation.

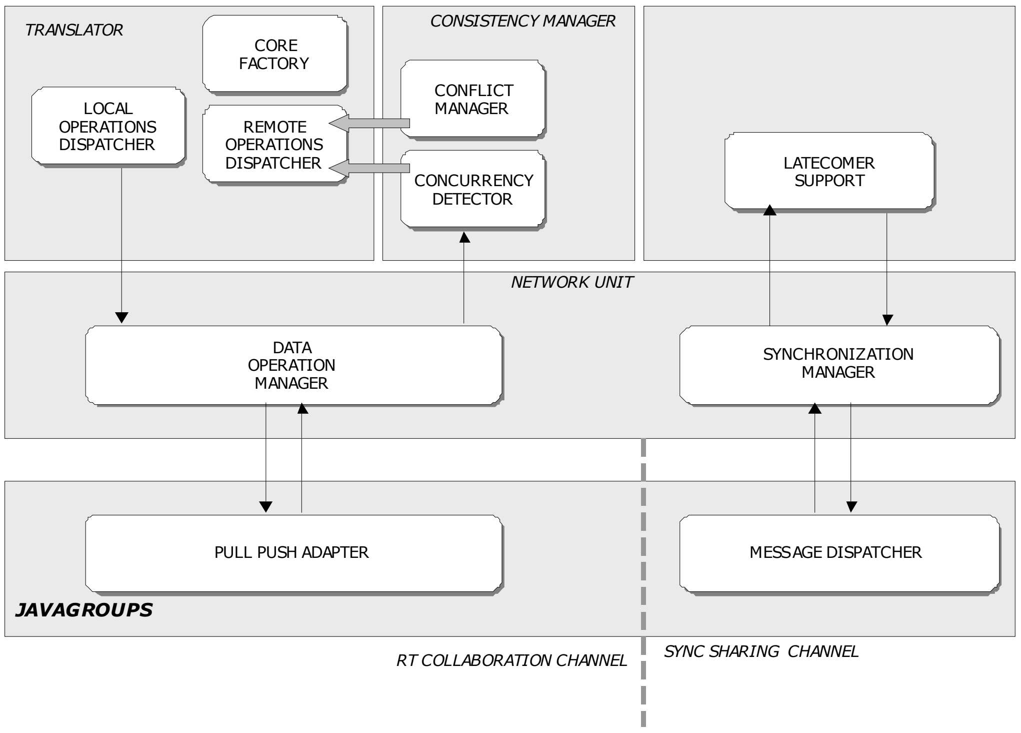


Figure 3.5 Integration of Application Layer and Communication Platform

3.6 The Translation Component

The Translation component consists on two main subcomponents: The *LocalOperationsDispatcher* and the *RemoteOperationsDispatcher*.

The *LocalOperationsDispatcher* translates all the local operations (after optimistic execution) into a serializable format in order to be sent through the network. The *DataOperationManager* receives the translated data from the *LocalOperationsDispatcher* and multicast it to the group.

When the *DataOperationManager* receives data from the network, the data is passed to the *ConcurrencyDetector*, if concurrence is not detected then the *RemoteOperationsDispatcher* translates the network data into operations to both models (UML and graphical).

3.7 The Latecomer Support

Latecomer support allows latecomers to join a collaboration session already in progress. When the newcomer joins the session the whole application state is transmitted to its site. Most existed prototypes in synchronous collaboration environment do not support latecomers. All the clients have to start the session at the same time. Otherwise, they may see different stages of the ongoing session. In the real world, the number of users in a collaboration session changes dynamically. This challenge is addressed in [11].

Existing latecomer support mechanisms can be divided in two categories:

> ***Transportation Protocols***: *"Such as Scalable Reliable Multicast Protocol. All the data packets from the beginning of the session are stored. The late coming application can reconstruct the current state according to the start state and these stored packets. However protocol level algorithms have some disadvantages. First, it is not efficient to transfer all the transport packets because most of the transmission information may be not relevant to the application. Second, some states cannot be reconstructed by using the transport packets".*
>
> ***Application Level***: Latecomer support is implemented with application dependent mechanisms. Most existing solutions are focusing on the application level approach.

Latecomer support for Software Modeling tools can be implemented easily simply transferring the application state in XMI format. In DArgoUML a decentralized approach has been implemented. The newcomer requests the application state to the coordinator of the group. The coordinator of the group is a simple peer.

However with the Consistency Maintenance Framework this simple operation has to be completed sending as well Consistency Maintenance related information as "conflict information" and "undo tables".

3.8 Integration with JavaGroups

JavaGroups has been used as platform for reliable group communication. It has been selected over other technologies like JXTA as communication platform.

3.8.1 JavaGroups Support for CSCW

JavaGroups offers high support for developing collaborative systems as:

Reliability: So that the operations/messages sent by all the users receive the users in a collaborative session. Reliability is necessary to maintain the consistency on the shared document.

Selection of Multicasting Protocol: JavaGroups provides different Multicasting Protocols implementations. So, FIFO, causal ordering, total ordering… can be selected. Thus, two of the three consistency problems (divergence and causal ordering violation) that arise in this types or systems can be easily solved.

The Consistency Maintenance problem left to solve is user intention violation that depends on application semantics and cannot be solved with a generic solution. For solving this problem a Flexible Consistency Framework has been devised based on Software Modeling Knowledge.

Simplicity: It is simple, easy to use and smart. With some basic knowledge about multicasting and sockets, the learning period is very small.

JavaGroups has made easier the process of developing the communication layer. With other technology, external mechanisms for reliability and multicasting ordering should have to be included, and the process likely would not have been so easy. Thanks to JavaGroups!

3.8.2 Architecture of JavaGroups

The participants can join the group, send messages to all members and receive messages from members in the group. The system keeps track of the members in every group, and notifies group members when a new member joins, or an existing member leaves or crashes. A group is identified by its name. Groups do not have to be created explicitly; when a participant joins a non-existing group, that group will be created automatically. The participants of a group can be located on the same host, within the same LAN, or across a WAN. A member can be part of multiple groups.

The architecture of JavaGroups consists on 3 parts:

- ***The Channel*** used by application programmers to build reliable group communication applications.
- ***The Building Blocks***, which are layered on top of the channel and provide a higher abstraction level.

- ***The Protocol Stack,*** which implements the properties specified for a given channel.

Channel: To join a group and send messages, a participant on the session has to create a channel and connect to it using the group name. The channel is the handle to the group. While connected, a member may send and receive messages to/from all other participants in the group.

The properties for a channel are specified in a colon-delimited string format. When creating a channel a protocol stack will be created according to these properties. All messages will pass through this stack, ensuring the quality of service specified by the properties.

> ***Building Blocks:*** Channels are simple and primitive. They provide asynchronous message sending/reception, somewhat similar to UDP. A message sent is essentially put on the network and the send method will return immediately. Conceptual requests, or responses to previous requests, are received in undefined order and the application has to take care of matching responses with requests. Besides the application actively retrieves messages from a channel (pull-style). Building Blocks provide more sophisticated mechanisms on top of a Channel. Applications communicate directly with the building block rather than the channel. The aim of Building Blocks is to save the application programmer from having to write tedious and recurring code, e.g. request-response correlation.

- *MessageDispatcher:* Provides synchronous (as well as asynchronous) message sending with request-response correlation, e.g. matching responses with the original request. It also offers push-style message reception (by internally using a Push Pull Adapter). The MessageDispatcher can be used in both client and server role: a client sends request and receives responses and a server receives requests and send responses. MessageDispatcher allows a application to be both at the same time.

- *PushPullAdapter:* This class is a converter between the pull-style of actively receiving messages from the channel and the push-style where clients register a callback, which is invoked whenever a message has been received. Clients of a channel do not have to allocate a separate thread for message reception.

- *Other Blocks: RpcDispatcher, DistributedHashTable, ReplicatedHashTable, DistributedTree, NotificationBus.*

The Protocol Stack: All messages sent and received over the channel have to pass through the protocol stack. Every layer may modify, reorder, pass or drop a message. The composition of the protocol stack for a channel is determined by the creator of the channel: a property string defines the layers to be used (and the parameters for each layer). When creating a channel, the properties of the underlying protocol stack can be specified as argument. A null argument means, use the default composition of layers in the protocol stack. A possible property specification may instruct JavaGroups to create an unreliable, UDP-based channel, another one may specify a loss-less, FIFO channel, and yet a third one may create a loss-less, FIFO, virtually synchronous, total order channel.

- The Sharing Channel has been implemented using a MessageDispatcher component. (See chapter 7. Implementation)

- The Collaboration Channel has been implemented using a PullPushAdapter component.

Chapter 4

Concurrency in Collaborative Editing Systems

4.1 Introduction

In this chapter a classification of operations based on causal dependencies is presented. Operations can be concurrent (independent) or causally dependent. Concurrent operations are examined to detect conflicts. Only concurrent operations can conflict.

In the second section a mechanism for detecting concurrent operations is presented.

4.2 Classification of operations based on concurrency

In Synchronous Collaborative Editing Systems several users can generate operations to manage the same shared document. Some of these operations are generated in response to the execution of previous operations; in this case a "Causal Ordering" relation exists among the operation executed previously and the operations generated in response. Some other operations are generated concurrently by users at different sites. Concurrent operations can generate conflicts if they try to modify the same attribute of the same object with different values. In 2.3.1.1Definition 1.1. Causal ordering relation a formal definition of "causal ordering" and "independent" 2.3.1.2Definition 1.2. Dependent and Independent operations relations is presented.

Conflict management mean accommodation of all user intentions. Conflict management and conflict definition are dependant on application semantics. Conflict definition, detection and management for software modeling systems are addressed deeply in 5.Consistency Maintenance Framework.

4.2.1 Causal ordering relation

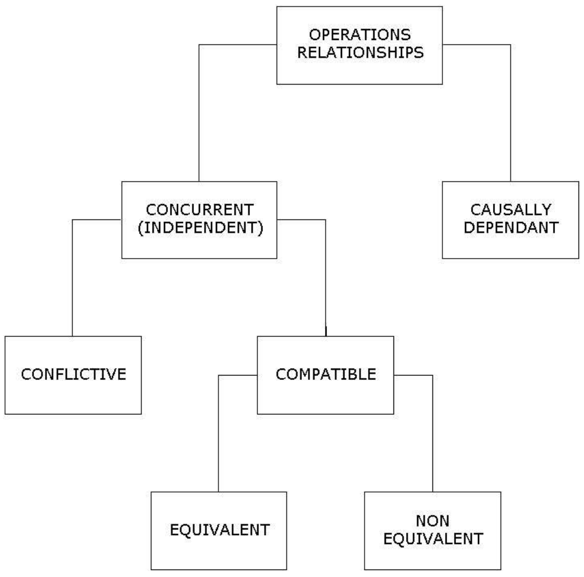
Given two operations OP_1 and OP_2 they are causally dependent, $OP_2 \rightarrow OP_1$ (OP_2 depends on OP_1), if OP_2 was generated with knowledge of OP_1. This dependency occurs among all the operations generated by one user at the same site, or if operations are generated at different sites, OP_2 was generated after the reception and execution of OP_1.

A Formal definition can be found on **2.3.1.1Definition 1.1. Causal ordering relation**

4.2.2 Independent operations.

Two operations are said to be *independent* or *concurrent* if they both were generated without knowledge of each other. Independent operations do no have "causal ordering relationships" so operations generated on the same site can never be concurrent.

A Formal definition can be found on **2.3.1.2Definition 1.2. Dependent and Independent operations**



4.2.3 Conflict Relation

Two concurrent operations can have a conflict relation. A conflict relation appears when all user intentions cannot be accommodated on the same object. For example several users trying to modify the same attribute of the same object with different values. The definition of conflict is dependant on application semantics. In [5] a formal definition was presented for the Collaborative Graphical Environment. This definition has been extended for the Collaborative

Software Modeling Environment. For a deeply understanding of conflicts examine 5.Consistency Maintenance Framework.

4.2.4 Definition A.1. UML Semantic Conflict Relation

Two operations have a conflict relation if both cannot be executed for one of the following reasons:

1. They try to modify the same attribute of the same object with different values. Only one user intention can be preserved. **2.3.2.2Definition 2.1. Conflict Relation** ┌
2. They both cannot succeed because of some restrictions imposed by application dependant rules. *UML Restriction Relations, and Time Line Dependency Relations* are addressed in 5.Consistency Maintenance Framework

4.2.5 Definition A.2. UML Semantic Compatible Relation

Two operations are compatible if they do not have a conflict relation regarding with Definition A.1. If two operations are compatible all user intentions can be accommodated into the same object.

Compatible operations can be classified into Equivalent and Non-Equivalent operations.

4.2.6 Definition A.3. Equivalent Relation

Two operations are said to be equivalent if their intended effects are the same. So all users intentions could be preserved executing only one of them.

4.2.7 Concurrent Cases

For instance if two users are trying to generate a generalization from class parent to class child, it will only be needed to generate a single generalization, to perform a single operation.

NOTE: Two equivalent operations have the same parameters:

OP_1: MODIFY ATTRIBUTE CLASS IS FINAL (Class Id = 1525, "true").
OP_2: MODIFY ATTRIBUTE CLASS IS FINAL (Class Id = 1525, "true")

But if the operations are CREATE operations, the Identifier of the object to be created will be different, as the Identifier is created locally in the site where the operation was generated:

OP_1: CREATE ASSOCIATION (Association Id = 15, Client Id = 16, Supplier Id = 17)
OP_2: CREATE ASSOCIATION (Association Id = 25, Client Id = 16, Supplier Id = 17)

In the first case any of the two operations can be selected and executed at any site. In the second case the same operation to be executed has to be selected at all sites, otherwise the new created generalization would have different identifiers in different sites and future operations would succeed in some sites whether it would fail in others leading to inconsistencies in the model.

4.3 Concurrency Detection

Several tasks should be performed in order to manage Conflictive User Intentions in a Collaborative System:

- *Detecting concurrency*: Among the set of operations received, detect which of them are causally dependent and which are concurrent.
- *Detecting conflict relations* among concurrent operations addressed in 5.Consistency Maintenance Framework.
- Mechanisms for *resolving/managing* each type of detected *conflict* addressed in 5.Consistency Maintenance Framework

As it has been said before in this chapter, two operations are concurrent if they were generated without the knowledge of each other. Operations generated by the same user are never concurrent. So, concurrent operations can only be generated in different machines.

Two operations are concurrent if they were generated on different machines, before the reception of each other:

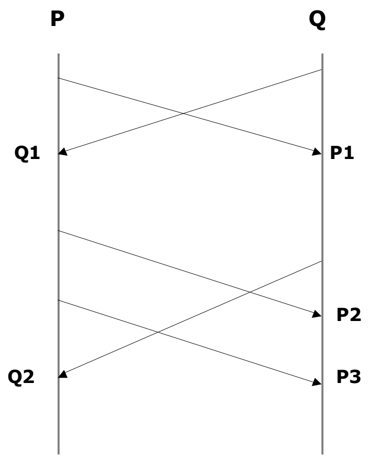


Figure 4.1Concurrent Operations

In the figure operations Q1 and P1 are concurrent; Q2 is concurrent with P2 and P3. Q2 depends on the execution of P1 and Q1. This can be expressed in a formal way according with 2.3.1.1Definition 1.1. Causal ordering relation and 4.2.2Independent operations.

a) $Q_1 \mid\mid P_1$
b) $Q_2 \mid\mid P_2, Q_2 \mid\mid P_3, P_3 \rightarrow P_2$
c) $Q_2 \rightarrow Q_1, Q_2 \rightarrow P_1$.
d) $P_2 \rightarrow Q_1, P_2 \rightarrow P_1$

The communication platform employed for constructing the systems is JavaGroups. With JavaGroups the protocol stack that will be used in the communications can be configured, in this way, the multicasting algorithm, reliability mode… etc can be selected.

For the Collaborative Editing system, at least a causal ordering multicasting algorithm is needed. JavaGroups provides even a total ordering multicast channel that includes causal ordering. Total ordering indicates that the same causal ordering will be received at all sites [16]. With total ordering concurrency could be detected:

ID (OP_1): Each operation has a Unique Identifier along all sites.
DependsOn (OP_1): Each operation knows which was the last operation received in the site where it was generated. (last operation received in the site or executed locally in the site).

With this information the state in which an operation was generated can be known. As there is a total ordering among all sites, the set of concurrent operations for OP_1 are those operations received at sites between the reception of DependsOn (OP_1), and the reception of $OP_{1.}$

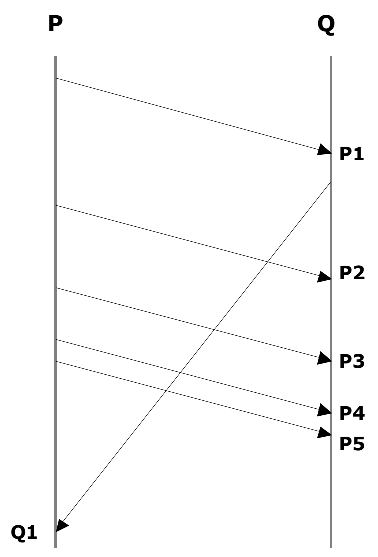


Figure 4.2Concurrent Operations

In the figure 4.2 a possible total ordering could be (P_1, P_2, P_3, P_4, P_5, Q_1):

DependsOn (Q_1) = P_1;

Concurrent operations with Q_1 are all those received between P_1 and Q_1 = (P_2, P_3, P_4, P_5).

A Concurrence detector at each site will detect the set of operations concurrent with each received operations. And the possible conflict relations would be studied among those concurrent operations.

If instead of total ordering the multicasting algorithm selected for the communication channel is causal ordering, the detecting procedure is similar, the value DependsOn instead of being an identifier for a unique operation a vector with the last operation received from each site should be maintained.

Once that the possible set of concurrent operations for each received operation is obtaining, conflicts relation among those operations are examined, and when a conflict is detected a specific management is performed. Chapter 4 is dedicated to Consistency Maintenance: Types of Conflicts, detection, management and resolution of conflicts.

Detecting conflict relations among concurrent operations addressed in 5.Consistency Maintenance Framework

Chapter 5

Consistency Maintenance Framework for Collaborative Software Modelling tools.

5.1 Introduction

In this chapter a framework for achieving Consistency Maintenance in a Collaborative Synchronous Software Modelling tool is presented:

First the *concurrency problems* regarding with consistency maintenance that arise in a Collaborative Editing System are presented, as well as the solution to each of them. There is one concurrency problem that has to be solved with application dependant mechanisms: *User Intention Violations*. This chapter and this Thesis are all about User Intention Preservation!

Second, an explanation of how *Conflict definition is dependant on application semantics* is presented. In other words in different environment conflicts are generated by different causes so the definition of conflict, the premises to detect a conflict, depend on the nature of the application.

Third, an extract of the OMG UML Specification 1.3 is presented, specifically the Abstract Syntax and Well-Formedness rules. The aim of this section is to show the fact that *UML Specification rules characterizes the behavior of the operations*. Some operations will have a restrictive behavior as they modify values affected by some UML restrictions. So, an understanding of UML Specification is required to understand the Definition of Conflict.

Based on the above, the formal definitions that characterize operations based on UML Restrictive behavior are presented in the fourth section.

In the fifth section *Time Line dependencies* among operations are presented as well as the formal definitions.

With the definitions in section fourth and fifth sections, the Conflict definition for Software Modeling Environments is finally completed.

In the next Section the different types of information and its importance for consistency maintenance that a UML Diagram consists on are presented, ***UML information and Graphical Information***.

From all this information - definitions, operation behavior, etc, a *hierarchy of operations classified by category is presented.* Each category of operations has a different behavior and associated conflict resolution mechanisms.

Finally the Conflicts: a *Matrix with the possible types of conflict* is presented. The different categories of operations are placed on each axis of the Matrix. This matrix shows the types of conflicts generated by the combination of two operations belonging to any category. With this conflict matrix the types of conflict can be detected based on the categories of the operations.

Then an explanation of *how each conflict can be managed*. Very few conflicts are resolved by the system and others will be solved by the participants.

There are some cases where the inconsistencies are not permitted: Strict Consistency. In some other cases conflicts will generate a temporary inconsistency: Flexible Consistency Model.

Finally the recording of user intentions is described. This information is preserved in a conflict object presented not only to the users participants in the conflict. The knowledge of user intentions improves group awareness in the application.

5.2 Inconsistency Problems in Collaborative Editing Systems

The three inconsistency problems that arise in a replicated architecture for a collaborative editing system have been presented in 2.3Consistency Maintenance Framework. The following table is an schema of the Consistency Maintenance problems, properties that are violated when the problems appears and the solutions to solve the problems.

Maintenance Problem	***Property Violated***	***Solution***
Divergence	*Convergence*	*Serialization*
Causality Violation	*Causality Preservation*	*Causal Ordering*
Intention Violation	*Intention Preservation.*	*App Dependent Mechanisms*

Divergence: *"Operations may arrive and be executed at different sites in different order resulting in different final results".* The effects of executing a set of operations OP_1, OP_2, OP_3 and OP_4 at different sites in different orders will cause divergent results. The consistency property that is not achieved is *Convergence*. Convergence ensures same final results on the copies of the shared document at all sites at the end of a session. The divergence problem can be solved with any *Serialization Protocol* that ensures the same total order at all sites.

Causality Violation: "Operations may arrive and be executed out of their natural cause-effect order". The property that is violated is *Causality Preservation.* It ensures the execution order of dependant operations to be the same as their natural cause-effect order during a session. Causality Preservation can be achieved with ordering on dependant operations (*Causal Ordering*).

Intention Violation: "As result of conflicting operations (for example operations to change the same attribute of the same object to different values), the effect of some operations is not preserved." Intention Preservation can only be achieved with application dependant mechanisms, as the conflicts of concurrent operations (independent operations) would be generated and solved differently depending on the application semantics.

The consistency model imposes execution order only on dependant operations, but not on independent operations as long as the convergence and intention preservation properties are always preserved.

5.3 Application Semantics dependency for Conflict Management

As it was presented in Table 5.1, *"Consistency Maintenance Problems and Solutions"*, user intention preservation is only possible with application dependant mechanisms.

The definition of Conflict and its management is also dependant on application semantics.
In Chapter 3, the definition for Conflictive operations in the Software Modeling environment is presented for the first time:

Definition A.1. Conflict Operations: Two operations have a conflict relation if both cannot be executed for one of the following reasons:

1. They try to modify the same attribute of the same object with different values. Only one user intention can be preserved. [Definition 2.1. Conflict Relation ⊗]

2. They both cannot succeed because of some restrictions imposed by Application dependant rules.

The first part of this definition has been obtained from the Consistency Framework presented in [5] for Graphical Collaborative Environments. However the Conflict definition has been extended in order to include other relationships among operations that occur in the UML Modeling Environment.

We now show how some of Chen affirmations appropriate for the Graphical Environment are no longer appropriate for other environments:

"Create Operations will always be compatible with each other because each create operation create a different object, Create operations do not conflict with other types of operations because operations targeting the same object as create operation must be causally after this create"

In the Graphical environment operations usually refer to one object (except grouping and ungrouping), like "DRAW A CIRCLE", "FILL CIRCLE", "MOVE CIRCLE". While in Software Modeling environments operations can refer to more than one item/object as for example:

> OP_1: CREATE GENERALIZATION (from parent-class to child-class): Involves three elements: generalization object to be created, parent-class item and child-class item.
>
> OP_2: CREATE ATTRIBUTE (on class): Involves two elements: The attribute to be created and the class to be created.

OP_1 is depends on the existence of the "parent-class" and "child-class". Operation CREATE GENERALIZATION (from parent-class to child-class) conflicts with operation DELETE CLASS (parent-class), as both effects cannot be maintained. These *"Time Line Dependency Relations"* originate conflicts.

There are some rules in the UML Specification that give rise to new conflict relations *"Conflict Restriction Relations"*. For example the following two operations will conflict:

a) OP_1: CREATE GENERALIZATION (from a-class to b-class);
b) OP_2: MODIFY CLASS ATTRIBUTE (a-class):
   b.1) ATTRIBUTE TYPE (IS_LEAF);
   b.2) ATTRIBUTE VALUE (true);

The effect of operation OP_2 is that a-class set its attribute "is-Leaf" to true. This means that a-class cannot participate in a generalization as parent class, while operation OP_1 is trying to such a generalization with a-class having a parent role.

In the UML Environment the ABSTRACT SYNTAX AND WELL FORMEDNESS RULES (OMG UML Specification 1.3) generates these two new categories of conflict relations:

1. UML Restriction Relation.
2. Time-Line Dependency Relation.

5.4 OMG UML Specification 1.3 Foundation package Core: Abstract Syntax and Well-Formedness Rules

The Abstract Syntax and Well-Formedness rules of the OMG UML Specification characterizes the behavior of some operations. The specification must be examined in order to know which operations have a restrictive behavior.

The Core package is the most fundamental of the sub-packages that compose the UML Foundation package. It defines the basic abstract and concrete metamodel constructs needed for the development of object models. Abstract constructs are not instantiable and are commonly used to reify key constructs, share structure, and organize the UML metamodel. Concrete metamodel constructs are instantiable and reflect the modeling constructs used by object modelers (cf. metamodelers). Abstract constructs defined in the Core include ModelElement, GeneralizableElement, and Classifier. Concrete constructs specified in the Core include Class, Attribute, Operation, and Association.

5.4.1Association:

Name: The name of the association that has in combination with its associated Classifiers must be unique within the enclosing namespace (usually a Package).

Operation affected: Modify Association Name.

5.4.2Aggregation

When placed on one end (the “target” end), specifies whether the class on the target end is an aggregation with respect to the class on the other end (the “source” end). Only one end can be an aggregation.

At most one Association End may be an aggregation or composition.
[Well-Formedness Rule. Association. 2]

Operations affected:
Modify Association Aggregation.
Modify Association Aggregation.

5.4.3Classifier

A classifier is an element that describes behavioral and structural features; it comes in several specific forms, including *class*, *data type, interface, component*, *artifact*, and others that are defined in other metamodel packages….

Name: It has a name, which is unique in the Namespace enclosing the Classifier.

Operations affected:
Modify Class Name.
Modify Interface Name.

5.4.4 Feature

A feature is a property, like operation or attribute, which is encapsulated within a Classifier.

Name: The name used to identify the Feature within the Classifier or Instance. It must be unique across inheritance of names from ancestors including names of outgoing AssociationEnd.

Operations Affected:
Modify Attribute Name.

5.4.5 GeneralizableElement

A GeneralizableElement is a model element that may participate in a generalization relationship.

[Well-Formedness Rule. GeneralizableElements.1]
A root cannot have any Generalizations.

Operation affected:
Modify Class is Root.

[Well-Formedness Rule. GeneralizableElements.2]
No GeneralizableElement can have a parent Generalization to an element that is a leaf.

Operation affected:
Modify Class is Leaf (final).

[Well-Formedness Rule .GeneralizableElements.3]
Circular inheritance is not allowed.

Operation affected:
Create Generalization.

5.4.6 Namespace

A namespace is a part of a model that contains a set of ModelElements each of whose names designates a unique element within the namespace. In the metamodel, a Namespace is a Model Element that can own other Model Elements, like Associations and Classifiers. The name of each owned Model Element must be unique within the Namespace.

Operation Affected:
Modify Class Name, Modify Generalization Name.
Modify Attribute Name, Modify Association Name.
Modify Interface Name, Modify Realization Name.
Modify Dependency Name.

5.5 UML Restriction Relation

As explained in the previous section "Abstract Syntax and Well-Formedness of core package UML" there is a set of Rules-Restrictions that must be taken into account. These restrictions make some operations to conflict among them. As explained earlier, for example a Generalization between two classes, parent-class and child-class, can only be created if parent-class is not "leaf" and child-class is not "root". A class has a set of modifiers (public, abstract, root, leaf) that describes the behaviour of the class. So, some operations are dependant on these values. On the other hand "is-leaf" modifier cannot be set to true if the generalization already exists for the parent class.

Thus,

5.5.1 Definition B.1 Restriction Relation: OP_1 ® OP_2

Given two Operations OP_1 and OP_2 they have a restriction relation OP_1 ® OP_2 if:

1. If executed sequentially in any order (OP_1, OP_2) or (OP_2, OP_1), the execution of the second operation can't be executed because of the restrictions imposed by the first one.

For Example: In this first case, OP_1 and OP_2 have a Restriction Relation.

> OP_1. CREATE GENERALIZATION (parent-class, child-class),
> OP_2. MODIFY CLASS ATTRIBUTE IS-LEAF ("true").

In the second case they don't:

> OP1. CREATE GENERALIZATION (parent-class, child-class),
> OP2. MODIFY CLASS ATTRIBUTE IS-LEAF ("false").

5.5.2 Definition B.2 Restrictive Behavior of an operation OP_1.

An operation OP_1 is said to have Restrictive Behavior if it is affected by the rules defined in the Abstract Syntax or Well-formed ness rules of UML Specification. The operations with restrictive behavior can cause future operations to fail due to the restrictions they imposed when they were executed.

For example OP_1: MODIFY CLASS ATTRIBUTE IS-LEAF ("true") could cause future OP: CREATE GENERALIZATION operations to fail.

5.6 Time-Line Dependency Relation

5.6.1 Introduction

As has been said before in the Software Modeling environment operations can refer to more than one item/object as for example:

OP_1: CREATE GENERALIZATION (from parent-class to child-class): Involves three items: generalization object to be created, parent-class item and child-class item.

OP_1 is dependant on the existence of the "parent-class" and "child-class". Operation CREATE GENERALIZATION (from parent-class to child-class) conflicts with operation DELETE CLASS (parent-class) as both effects cannot be maintained.

In paper [8] a hierarchy of temporal dependencies among operations is presented. The Temporal set represents all types of conflicts that cause inconsistencies across a timeline.

Temporal Roles:

1) *Depends-On*: Indicates that the action depends on some other action being earlier in the timeline.

2) *Dependable:* Indicates that the action may be used as a target of a Depends-On relation.

3) *Server-Depends-On:* Indicate that the action may break some depends-On relations.

For example, the following UML operations have the following temporal relations:

CREATE CLASS:

Dependable: Other operations depend on this one.

CREATE GENERALIZATION (parent-class, child-class):

Depends-On: CREATE CLASS.

Dependable: Yes. (Modify and Delete Generalization).

DELETE CLASS:

Depends-On: CREATE CLASS.

Server-Depends-On: Yes. The execution of this operation could create conflicts with other operations that Depends-On CREATE CLASS as MODIFY CLASS, CREATE GENERALIZATION…. etc.

From the Idea of time-line dependencies the following definitions would be needed in the software modeling environment for classifying the operations.

5.6.2 Definition C.1. Independent-Behavior Operation

An operation OP_1 is said to have *Independent-Behavior* if it doesn't depends on the execution of some other action being executed earlier in time.

Examples:

CREATE CLASS

CREATE INTERFACE

5.6.3 Definition C.2. Dependent-Behavior Operation.

An Operation OP_1 is said to have *Dependent-Behavior* if in order to be executed it needs some other operation to have been executed earlier in time.

Examples:

CREATE ASSOCIATION

CREATE ATTRIBUTE

CREATE OPERATION

NOTE: The Causally Dependency explained in The Definition 1.2 explains when two Operations have a Causally Dependant or Causally independent relationships. This dependency comes from the Causal Ordering Property. While In Definition C.1 and C.2 Dependency refers to the nature of the Operations. Definition 1.2 is only about Concurrency.

5.6.4Definition C.3. Break Dependencies Behavior.

An Operation OP_1 is said to have *Break Dependencies Behavior* if its execution could cause some dependant operations to fail.

Examples:

DELETE CLASS

DELETE INTERFACE

DELETE ATTRIBUTE

5.7 Types of Information in a UML Diagram

A UML Diagram consists of UML Information and Graphical Information for the diagram Layout.

As explained in 2.4.2XML Metadata Interchange: XMI in the UML and XMI Specifications no graphical information is included. When a UML model is exported in XMI format only the UML information is included. UML tools save graphical Information in graphical vector formats such as SVG or PGML. Software modeling tools and the user are responsible for managing the graphical appearance.

It can be said that the following two diagrams contain the same UML Information and can be said to be UML Consistent.

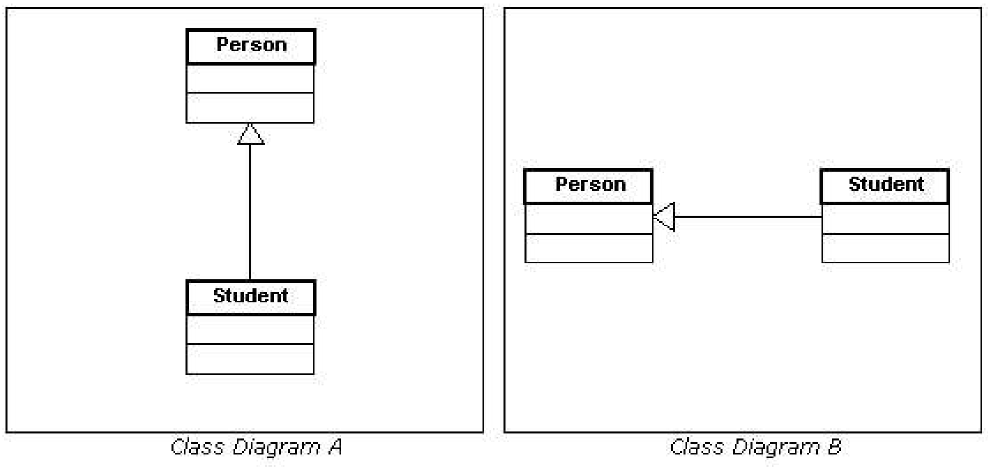


Figure 5.1Equivalency of UML Diagrams

Graphical Information Consistency Maintenance is considered of less importance, as it does not affect to the UML Information Consistency. In the face of a conflict the graphical operations are considered the least important.

Graphical Operations have been classified in two categories:

> *Graphical Distributed*: These operations are distributed to all sites. They give information about the current localization of the UML elements. There are very few Graphical operations that are distributed, only the modification of the localization of the edges (classes and interfaces) of the graph:
>
> > MOVE CLASS TO (x, y).
> > MOVE INTERFACE TO (x, y)
>
> *Graphical Local*: These operations are not distributed among all sites, as they will only confuse the other participants and will waste network bandwidth:
>
> > RESIZE CLASS (x,y, height, width)
> > TRANSLATE CLASS (dx, dy): drag event with the mouse.
> > GRAPHICAL OPERATIONS of the Edges (Associations, Generalizations, Dependencies, Realizations)

While the first set of operations "Graphical Distributed" can be considered to clarify the UML model, the second set of operations will only confuse the other users.
If one user drags a class on the screen, the other users should not see how the class is being moved during the movement, only the final position is needed and revealed. Otherwise the participants would see all the movements, clicks, drags and drops of their collaborators.

Even if a few operations are transmitted in the face of a conflict they are considered the least important and are ignored as they do not contribute to the UML consistency maintenance.

5.8 Classification of Conflict Operations based on Application Dependant relations (UML relations)

In the UML Collaborative Prototype a subset of the possible operations managing UML elements have been implemented. This subset consists of operations for managing Class Diagrams.

The Operations Implemented are CREATE, MODIFY and DELETE, applied to Classes, Attributes, Interfaces, Associations, Generalizations, Realizations and Dependencies.

CREATE: Creates the Objects.
MODIFY: Modifies the Properties of the Objects.
DELETE: Erases the object.

Based on the operations' behavior (restrictive non restrictive, dependent non dependent, break dependency behavior, graphical information operations) , the types of conflicts that can be generated, and the management of the generated conflicts a hierarchy of operations has been constructed classified by category.

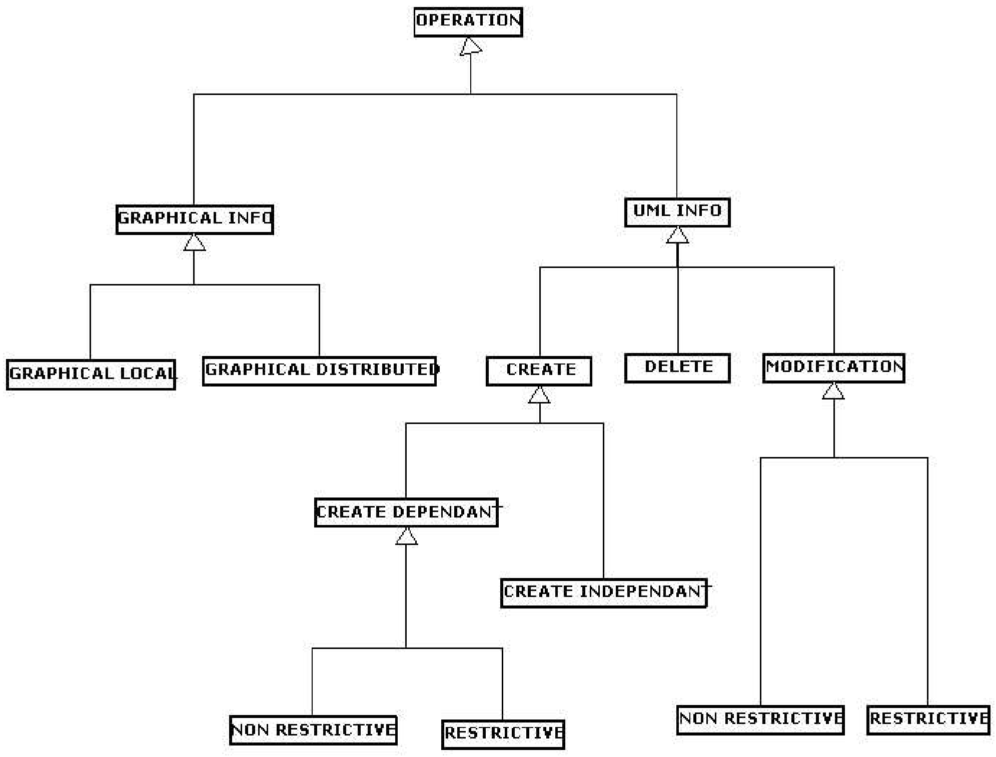


Figure 5.2Operation Categories for Conflict Management

5.9Description of Operations Categories

5.9.1Create Independent

Create Independent: Operations included in this category are those with an "Independent-Behaviour" as per ***Definition C.1.*** These create operations are always compatible with other create Independent operations as each create operation creates a different object, and they cannot conflict with other create dependant, modify or delete operations, as operations targeting the same object as the create operation must be causally after this create.

Examples are the creation of the nodes in a UML Diagram:

CREATE CLASS.
CREATE INTERFACE.

The creation of classes or Interfaces does not depend on the existence of other elements. These operations will never conflict with any other operation: MODIFY CLASS ATTRIBUTE operations will never be concurrent with CREATE CLASS, as it is generated after the execution of CREATE CLASS.

5.9.2 Create Dependent Non Restrictive:

Operations included in this category are those CREATE operations with a *"Dependent Behaviour"* according with definition C.2 and at the same time do not have a *"Restrictive Behaviour"* according with definition B.2.

Create Operations that do depend on the execution of other operations earlier in the time line. The Objects created are not under restrictions that could make the Creation operation fail. Operations of this type are: (for example)

CREATE ATTRIBUTE on Class X
CREATE OPERATION on Class X
CREATE ASSOCIATION from Class X to Class Y
CREATE REALIZATION from Class X to Interface Y
CREATE DEPENDENCY from Class X to Class Y

5.9.3 Create Dependent Restrictive:

Operations included in this category are those CREATE operations with a *"Restrictive Behavior"* according to Definition B.2 and with a *"Dependent Behavior"* according to definition C.2.

Explanation by Example:

CREATE GENERALIZATION from Child Class to Parent Class.

In order to be successfully executed, Child Class and Parent Class must have been created earlier and Child Class must have its "is-root" attribute set to false, and Parent Class must have its "is-leaf" attribute set to false, otherwise the operation will fail.

Another example would be, trying to execute the following two concurrent operations:

CREATE GENERALIZATION from Child Class to Parent Class.
CREATE GENERALIZATION from Parent Class to Child Class.

The UML model specifies that a Class can be the Parent of the Child in a relationship, but it does not allow it to be the parent of one class and the chill of the same class.

5.9.4 Modify Non Restrictive:

Modify operations always have "Dependent Behaviour" since in order to modify a value it has to be created previously. To this category belong Modify operations that do not restrict operations to fail in the future. In other words operations that do not modify any value that could restrict other operations in the future. Some examples are:

MOD_ATTRIBUTE_INITIAL_VALUE, MOD_ASSOCIATION_NAVEGABILITY.
MOD_ASSOCIATION_MULTIPLICITY, MOD_ATTRIBUTE_VISIBILITY.
MOD_ATTRIBUTE_IS_STATIC, MOD_ATTRIBUTE_IS_TRANSIENT.
MOD_ATTRIBUTE_IS_VOLATILE.

5.9.5 Modify Restrictive:

Modify operations that have "Restrictive Behavior". Examples:

MOD_CLASS_NAME. MOD_DEPENDENCY_NAME.
MOD_ASSOCIATION_NAME. MOD_GENERALIZATION_NAME.
MOD_ATTRIBUTE_NAME. MOD_CLASS_IS_FINAL.
MOD_CLASS_IS_ROOT. MOD_ATTRIBUTE_TYPE.
MOD_ASSOCIATION_AGGREGATION. MOD_INTERFACE_NAME.
MOD_REALIZATION_NAME.

5.9.6 Delete:

Delete operations have a *"Break dependencies"* behaviour so its execution could cause conflicts with operations that depend on the object that is targeted to be deleted.

DELETE_CLASS, DELETE_GENERALIZATION.
DELETE_ATTRIBUTE, DELETE_OPERATION.
DELETE_ASSOCIATION.

Delete operation can perform deletes in cascade. Deleting a class means deleting all its attributes, associations, generalizations…

5.9.7 Graphical Local

As explained before in this chapter a UML Diagram has some Graphical Information (Diagram Layout) and UML Information. Graphical Inconsistency does not lead to UML Consistency. In a UML Diagram the important information is the UML Information not the graphical one.

However two levels of Importance have been given to Graphical Information.

To the Category of GRAPHICAL_LOCAL belong all the graphical operations that have been considered of no importance and not only does not give any information but also confuses or distracts the user. For example:

- ELEMENTS_TRANSLATION: Dragging elements on the screen. It would be very confusing to see in the screen how all the users are moving the elements of fixing the layout.

- ELEMENTS_RESIZING.

- EDGES LOCATION: Edges are Associations, Generalizations, Dependency and Realizations.

- EDGES SHAPE MODIFICATION.

In this case is the tool responsible for assigning some appropriated values.

5.9.8Graphical Distributed:

Even If Graphical Information does not give any UML information the Localization of the Nodes (Classes and Interfaces) is distributed. An analyst/designer could arrange the layout of the diagram so it is more clear and understandable. For this purpose it it is only necessary to distribute the localization of the Nodes (Classes and Interfaces Only the localization will be distributed not the translation. If a user is moving a class from position A until position B, The others users are not aware that a class is being moving, until the destination position has been chosen.

5.10 Conflict Detection Matrix

The Following Matrix presents the kind of conflicts generated between any two categories. Each kind of conflict has a different Resolution.

	CREATE INDEP.	CREATE DEP. NR	CREATE DEP. REST	MODIFY NR	MODIFY REST.	DELETE	GRAPH. DISTRIB
CREATE INDEPEND							
CREATE DEPEND NR						DELETE CONFLICT	
CREATE DEPEND RESTRICT			CREATE RESTRICT CONFLICT		MODIFY CREATE RESTRICT	DELETE CONFLICT	
MODIFY NR				MODIFY CONFLICT		DELETE CONFLICT	
MODIFY REST.					MODIFY RESTRICT CONFLICT	DELETE CONFLICT	
DELETE.							DELETE GRAPH CONFLICT
GRAPH DISTRIB							GRAPH CONFLICT

Figure 5.3Conflict Detection Matrix

5.10.1 Conflicts Management

In this section a description of the conflicts and its management is presented.

There are different types of conflicts. A few conflicts will be resolved by the system but most of them need to be resolved by the users. A conflict is resolved by the system only in special cases such as some Deletes and Graphical Information Updates.

In the face of a conflict: all user intentions are preserved in the shape of Conflicts objects that are associated with the UML Elements targeted by the conflictive operations. This information is accessible to all the participants of a session.

This model combines flexible consistency maintenance allowing temporary inconsistency for conflicts of type *"Modify Non Restrictive Versus Modify Non Restrictive"* and *"Graphical Versus Graphical"*. In the Graphical-Graphical conflict the inconsistency refers to Graphical Information not to UML Information. Whereas in other cases the management of conflicts is based on a Strict consistency maintenance model.

Why in some cases a temporary inconsistency is permitted and in other cases it is not?

Temporary inconsistency is permitted when the level of inconsistency can be recovered. The inconsistency is allowed for non restrictive values. So the level of inconsistency will not grow.

If the inconsistency were permitted for restrictive values these values will be different in all the sites. And some future operations would fail in some sites and in some others will not. The level of inconsistency would grow as some operations would be executed only in some sites. And this inconsistency state could never reach a consistency state as the system hasn't enough information.

Whenever is possible the flexible consistency model is selected, letting the user resolve the conflict at the moment he chooses.

The participants know at any moment, the level of consistency of the document, and which operations and participants have generated a conflict and which were the intentions of all the users participant in the conflict.

For most of the conflicts the state of the shared document is the same in all the sites.
A priority level is assigned to the operations thus:

1. CREATE operations.
2. MODIFY value.
3. DELETE operations.
4. GRAPHICAL Operations.

If a CREATE operation (create generalization) conflicts with a MODIFY operation, the create is executed and the MODIFY operation is stored as conflict.

This level of prioritization is based on the amount of information created/lost by any operation. So DELETE Operations have less priority than MODIFY or CREATE as Delete removes information whereas CREATE adds information to the model.

If a graphical Operation conflicts with a delete, the graphical Operation is ignored.

Below there is an Schema for the management of the different types of conflicts.

5.10.2 Conflict: Modify Non Restrictive Versus Modify Non Restrictive.

- Temporary UML Inconsistency.
- Maintain Local values (Optimistic execution) until conflict resolved.
- Create Conflict Information Objects with User Intentions.

5.10.3Conflict: Modify Restrictive Versus Modify Restrictive.

- Not permitted Inconsistency. Consistency Maintained at all sites.
- At all sites assign the default non restrictive value until conflict is resolved.
- Create Conflict Objects with User Intentions.

5.10.4Conflict: Create Restrictive Versus Modify Restrictive.

- Not permitted Inconsistency. Consistency Maintained at all sites.
- Perform Create.
- Create Conflict Objects with User Intentions.

5.10.5Conflict: (Create || Modify) Versus Delete

- Not permitted Inconsistency. Consistency Maintained at all sites.
- Perform Create or Modify.
- Create Conflict Objects with User Intentions.

5.10.6 Conflict: Create Restrictive Versus Create Restrictive

- Not permitted Inconsistency. Consistency Maintained at all sites.
- Not perform any Operation.
- Create Conflict Objects with User Intentions.

5.10.7 Conflict: Delete Versus Graphical

- Not permitted Inconsistency. Consistency Maintained at all sites.

- Perform Delete.
- Ignore Graphical Operation.
- Not Create Conflicts.

5.10.8 Conflict: Graphical Versus Graphical

- Graphical Inconsistency allowed.
- Not Create Conflicts.

5.11 Conclusion

In this chapter a Consistency Maintenance Framework for Collaborative Software Modeling tools have been proposed. The framework preserves all user intentions. For achieving user intention preservation application semantics are used.

In Software Modeling there are two types of information: UML model information and Graphical Information. Graphical Information has a lower priority level than UML information.

The detection and management of conflicts is based on the Abstract Syntax and Well Formedness rules of UML Specification 1.3 and in Time-Line dependencies.

At the end of the chapter a matrix for detecting conflicts is presented with an explanation of how each conflict can be treated.

The next chapter addresses Implementation issues. Basically describes how each component presented earlier in the architecture chapter has been implemented.

Chapter 6
Implementation

6.0.1 Introduction

The algorithms and schemas presented in previous chapters have been implemented in the **DArgoUML** prototype system. ArgoUML is fully implemented in Java. DArgoUML is the **D**istributed version of ArgoUML, which includes a Consistency Maintenance Framework based on Software Modeling Knowledge.

The effort for devising DArgoUML has been distributed in two phases:

- ***Phase I:*** To extend a single-user tool, ArgoUML, in order to obtain a distributed version. A Real-Time Software Modeling tool where multiple users can collaborate concurrently.

- ***Phase II:*** To devise a Consistency framework for the collaborative tool where the three consistency properties (convergence, causal order preservation and intention preservation) are maintained. The Consistency Framework is based on application dependant information.

The starting point for the ***phase I,*** was single-user modeling tool ArgoUML, version 0.10.1. This version had initially more that 800 classes for the User Interface Application. Besides ArgoUML manages two information models, the UML model implemented in the library ***NSUML*** by Novosoft and the graphical information library ***GEF*** implemented by Tigris.

These three components: ***ArgoUML, NSUML*** and ***GEF*** have been modified for distributing the tool. It has been a must during the development not to modify too much source code. However, the phase I, has been long and hard because of the amount of source code involved.

For the ***phase II***, the hardest task was not the implementation but the theoretical design of the Flexible Consistency Maintenance Framework.

This chapter addresses the implementation of *phase I* and *phase II*.

6.1 NSUML: The UML Model

ArgoUML is compliant with the OMG Standard for UML version 1.3. The code for the internal representation of an UML model is completely generated from the specification. To achieve this, a special metamodel library NSUML was developed by Novosoft.

Novosoft UML library provides an implementation of complete UML 1.3 physical metamodel, event notification, undo/redo support, reflective API, XMI loading/saving.

For the purpose of distributing ArgoUML a set of NSUML classes have been modified. The set includes all classes representing elements belonging to Class Diagrams: classes, interfaces, attributes, generalizations, dependencies, associations, … etc. They belong to the ***uml.foundation.core*** package:

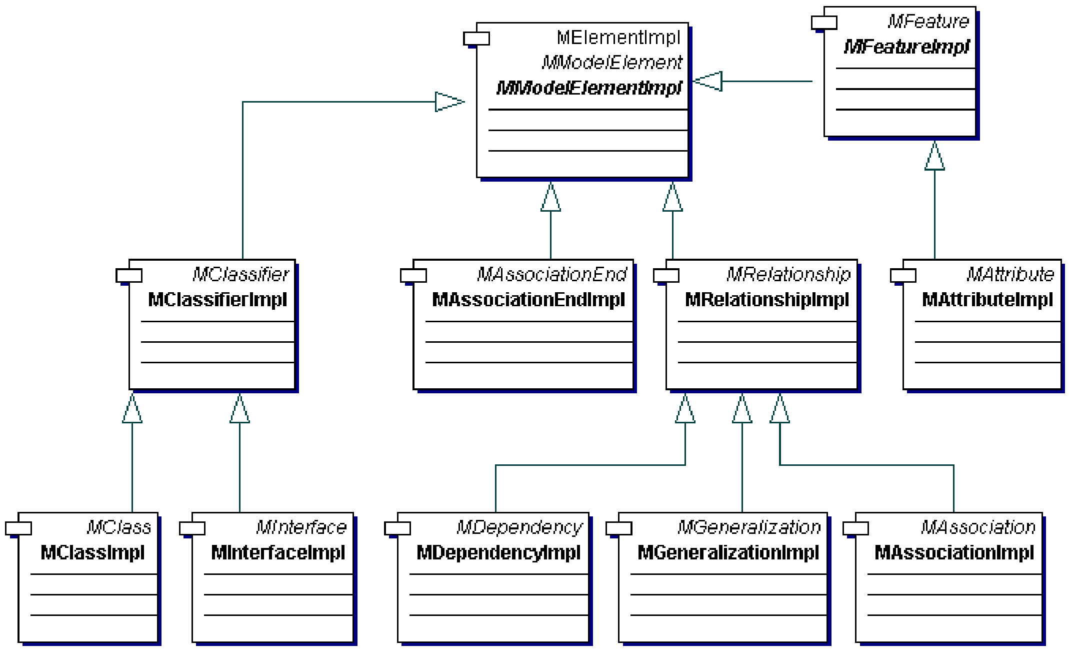


Figure 6.1UML model Elements in Class Diagrams

Some methods of these elements have been modified. The creation, deletion and modification of some attributes have been captured. In some cases when a change is applied to the model, the change has to be distributed to the rest of the participants. In other cases the change is not distributed. If the change was generated locally it will be distributed. But, if the change was generated remotely or the change is the result of a conflict resolution it will not be distributed. Otherwise an infinite loop would be generated. In figure 6.2 a schematic definition of the element MclassImpl is presented. MclassImpl is the implementation of a "class" element in a class diagram.

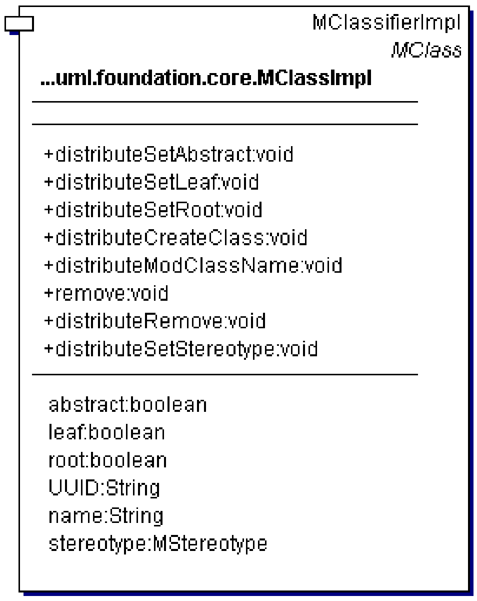


Figure 6.2 MClassImpl definition

Each element has been modified with the addition of ***"distribute methods"***. Besides each class setter and delete methods have been updated, so that they can check if the operation is local or not and launch or not the distribution. During the loading of ArgoUML several elements are created and accessed. During the loading of the application distribution is not enabled.

In figure 6.3 the classes involved in the process of distribution are presented: Elements of packages *ru.novosoft* (*NSUML*) and *org.tigris* (*GEF*) use class ***NetworkContext*** and ***ApplicationStatus*** to know if it is necessary to distribute. NetworkContext and ApplicationStatus classes are implemented with ThreadLocal mechanisms. ThreadLocal control management mechanisms are addressed further in this chapter.

The LocalOperationsDispatcher is responsible for launching all the changes to the network.

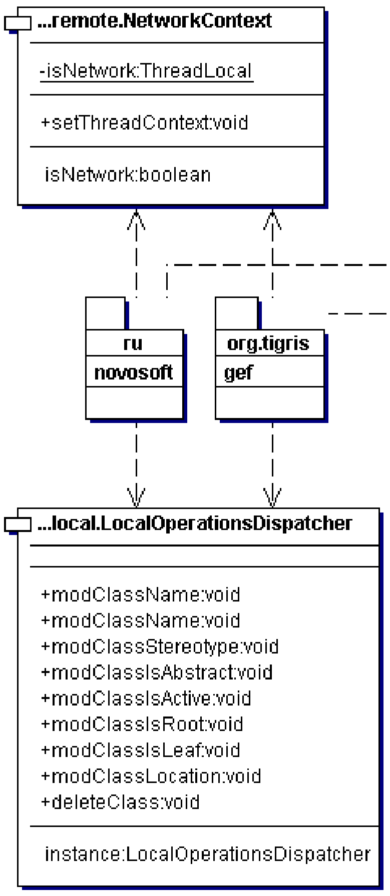


Figure 6.3Distribution of Local Operations

6.2 GEF Model

GEF is a library used to construct graph editing applications using a Node-Port-Edge model. It Model-View-Controller design based on the Swing Java UI library makes GEF able to act as a UI to existing data structures. It supports XML-based file formats based on the PGML standard.

As it has been explained earlier, graphical operations have a low level of priority. Only operations modifying the localization of some UML elements are distributed. The UML elements corresponding with the nodes of a graph are those whose localization will be distributed: Classes and Interfaces. Localization and resizing of edges (realization, dependencies, inheritance, association) are not distributed.

The modifications done in the graphical model are very similar to those done in NSUML model. Only the class Fig from the package org.tigris has been affected.

6.3 Real-Time Collaboration Channel

The Collaboration Channel is used for the transmission/reception of all the real time operations generated by all the participants in the session.

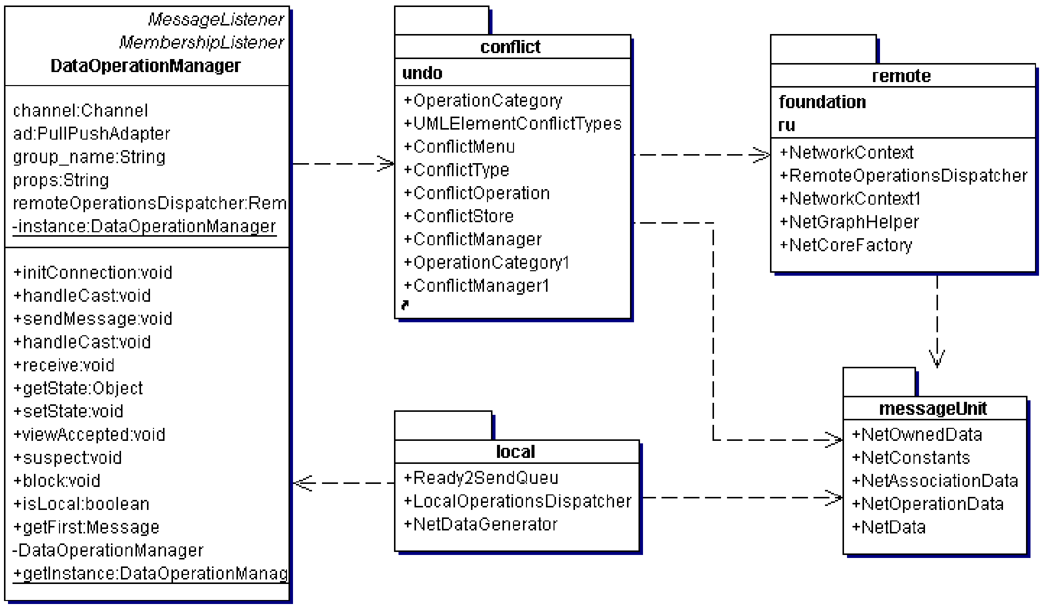


Figure 6.4 Real-Time Collaboration Channel

DataOperationManager: Responsible for sending/receiving the Real-Time operations to/from the channel:

a) Local Operations: It receives local operations from the LocalOperationsDispatcher in a serializable format (NetOperationData). The DataOperationManager encapsulates the data into a message and sends it to the network.

b) Remote Operations: It receives remote operations encapsulated in messages from the channel. It passes the data to the ConflictManager.

The *DataOperationManager* is implemented using a ***PullPushAdapter.*** It allows clients of a channel to be notified when messages have been received instead of having to actively poll the channel for new messages. This eliminates any need for the clients to allocate a separate thread for receiving messages.

Upon creation, an instance of *PullPushAdapter* creates a thread which constantly calls the Receive method of the underlying Transportable instance (e.g. a channel), blocking until a message is available. When a message is received, if there is a registered message listener, its Receive method will be called.

DataOperationManager implements MessageListener JavaGroups Interface, and in its receive method is implemented the actions to be done when a message is received.

Conflict, Remote and ***Local*** packages use MessageUnit data Classes for transferring the information.

MessageUnit: Contains all the class definitions necessary to transfer all types of operations.

6.4Local Operation Dispatcher

In figure 6.3 the process for distribution of Local Operations is presented. When any of the models are updated if the operation was generated locally specific data is send to the Local Operation Dispatcher, that encapsulates the data into a serializable format and send it to the DataOperationManager.

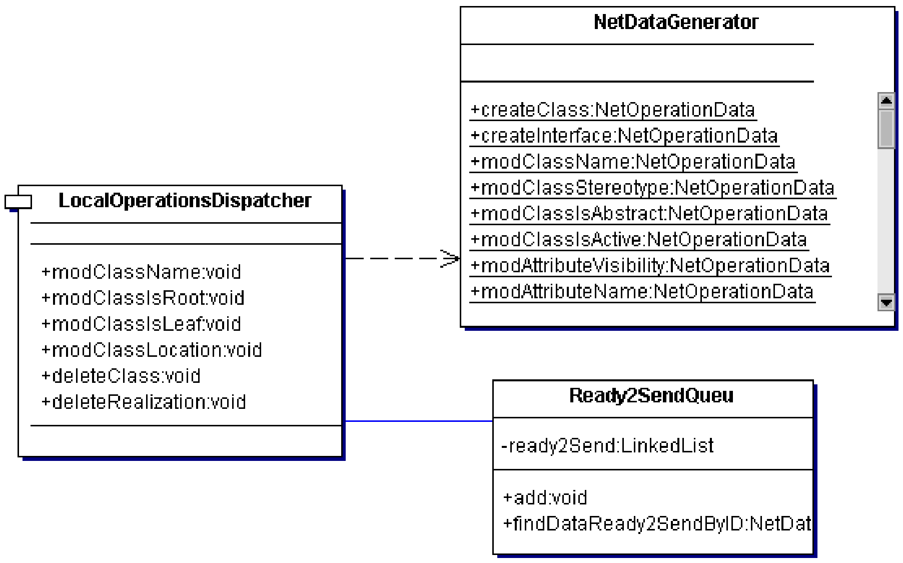

NetDataGenerator
+createClass:NetOperationData
+createInterface:NetOperationData
+modClassName:NetOperationData
+modClassStereotype:NetOperationData
+modClassIsAbstract:NetOperationData
+modClassIsActive:NetOperationData
+modAttributeVisibility:NetOperationData
+modAttributeName:NetOperationData
LocalOperationsDispatcher
+modClassName:void
+modClassIsRoot:void
+modClassIsLeaf:void
+modClassLocation:void
+deleteClass:void
+deleteRealization:void
Ready2SendQueu
-ready2Send:LinkedList
+add:void
+findDataReady2SendByID:NetDat

Figure 6.5 Local Operations Dispatcher

The ***LocalOperationsDispatcher*** uses the ***NetDataGenerator*** to generate serializable data with a specific format to be sent through the network. NetDataGenerator generates NetOperationData objects indicating the type of operation and the parameters. The data to be sent are queued until they are sent to the DataOperationManager.

6.5 Remote Operations Dispatcher

The *RemoteOperationsDispatcher* is module called to execute remote operations into the local module. It uses some utility classes as NetCoreFactory, and NetCoreHelper to manipulate UML elements and NetGraphHelper for manipulating the graph model.

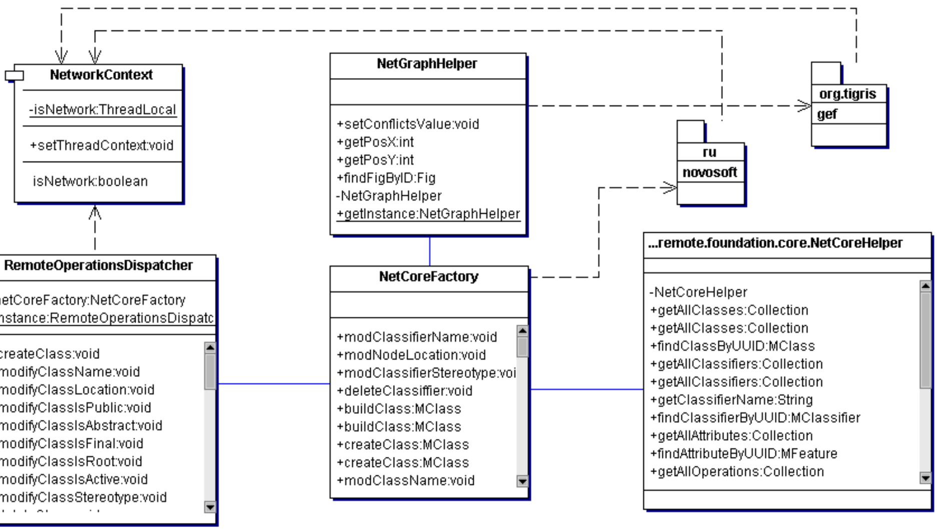

NetworkContext
-isNetwork:ThreadLocal
+setThreadContext:void
isNetwork:boolean
NetGraphHelper
+setConflictsValue:void
+getPosX:int
+getPosY:int
+findFigByID:Fig
-NetGraphHelper
+getInstance:NetGraphHelper
org.tigris
gef
ru
novosoft
RemoteOperationsDispatcher
-netCoreFactory:NetCoreFactory
-instance:RemoteOperationsDispatc
+createClass:void
+modifyClassName:void
+modifyClassLocation:void
+modifyClassIsPublic:void
+modifyClassIsAbstract:void
+modifyClassIsFinal:void
+modifyClassIsRoot:void
+modifyClassIsActive:void
+modifyClassStereotype:void
NetCoreFactory
+modClassifierName:void
+modNodeLocation:void
+modClassifierStereotype:voi
+deleteClassifier:void
+buildClass:MClass
+buildClass:MClass
+createClass:MClass
+createClass:MClass
+modClassName:void
...remote.foundation.core.NetCoreHelper
-NetCoreHelper
+getAllClasses:Collection
+getAllClasses:Collection
+findClassByUUID:MClass
+getAllClassifiers:Collection
+getAllClassifiers:Collection
+getClassifierName:String
+findClassifierByUUID:MClassifier
+getAllAttributes:Collection
+findAttributeByUUID:MFeature
+getAllOperations:Collection

Figure 6.6Remote Operations Dispatcher Working method

The RemoteOperationDispatcher updates the NetworkContext object to indicate that the current execution thread comes from the network, so, the operation should not be distributed. When the model (UML or graph) is updated, the NetworkContext value is checked. As result the operation is executed locally but not transmitted to the network.

6.6 Conflicts

6.6.1 ConflictManager

Acts as a "conflict detector": With a set of rules it can detect if two operations have a conflict relation and the conflict type. There are seven different types of conflicts. Every type has a set of actions associated (some conflicts are resolved by the system, others need to be resolved by the users, in some conflicts there is a temporarily inconsistency while in others the inconsistency is not permitted). The ConflictManager is the Implementation of the Consistency Maintenance Framework addressed deeply in 5.Consistency Maintenance Framework.

In a face of a conflict detected the ConflictManager decides if an operation is to be performed. To perform an operation the RemoteOperationsDispatcher will be called. The ConflictManager uses
the ConflictStore to maintain the information associated with the conflicts.

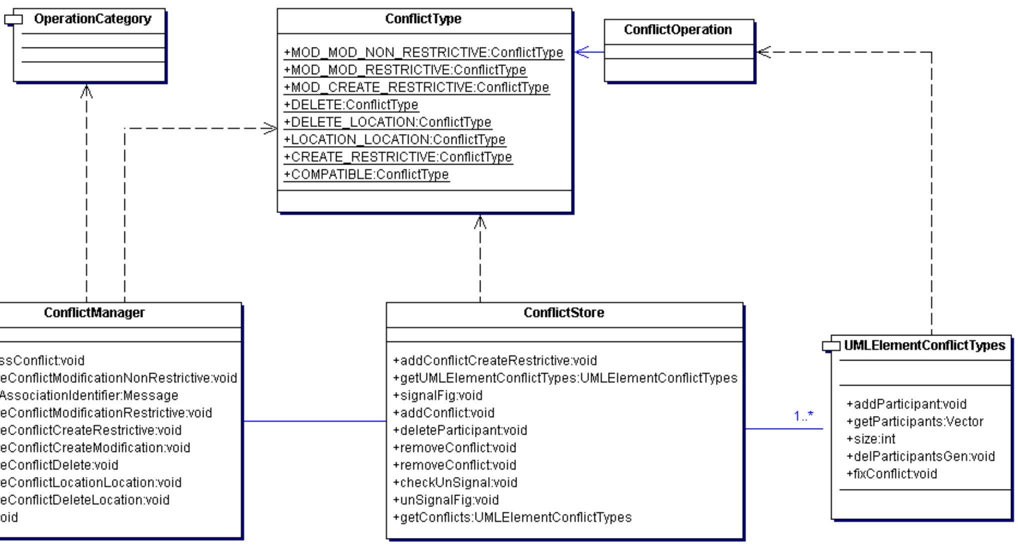

OperationCategory
ConflictType
+MOD_MOD_NON_RESTRICTIVE:ConflictType
+MOD_MOD_RESTRICTIVE:ConflictType
+MOD_CREATE_RESTRICTIVE:ConflictType
+DELETE:ConflictType
+DELETE_LOCATION:ConflictType
+LOCATION_LOCATION:ConflictType
+CREATE_RESTRICTIVE:ConflictType
+COMPATIBLE:ConflictType
ConflictOperation
ConflictManager
+processConflict:void
+resolveConflictModificationNonRestrictive:void
+checkAssociationIdentifier:Message
+resolveConflictModificationRestrictive:void
+resolveConflictCreateRestrictive:void
+resolveConflictCreateModification:void
+resolveConflictDelete:void
+resolveConflictLocationLocation:void
+resolveConflictDeleteLocation:void
-undo:void
ConflictStore
+addConflictCreateRestrictive:void
+getUMLElementConflictTypes:UMLElementConflictTypes
+signalFig:void
+addConflict:void
+deleteParticipant:void
+removeConflict:void
+removeConflict:void
+checkUnSignal:void
+unSignalFig:void
+getConflicts:UMLElementConflictTypes
1..*
UMLElementConflictTypes
+addParticipant:void
+getParticipants:Vector
+size:int
+delParticipantsGen:void
+fixConflict:void

Figure 6.7Conflict Management

6.6.2ConflictStore

Maintains all the conflict data relative to the UML elements in the system. The ConflictStore is always consistent among all participants in a session. The information is the same in every peer.

When a new element is added to the store, the UML element is graphically marked (red shadow) so the user is aware of conflicts associated with the element. In the same fashion if all the conflicts are deleted from an element the now “non-conflictive” element has to be unmarked.

It is implemented with a Hashtable: an entry for every UML element with a conflict associated. The key entries of the Hashtable are the unique Identifiers of the elements. The ConflictStore has methods for adding, removing conflict data to a UML element. Each UML element can have many conflicts associated. For each UML element, its associated conflict data is maintained in an object of type *UMLConflictType*.

6.6.3UMLConflictType

A UML element can have many conflicts associated. UMLConflictType is a Hashtable that maintains all the information related with the conflicts associated with one UML elements. The conflicts are categorized by the conflictive operation that generated the conflict. Then, the key of the Hashtable is of type *ConflictOperation*. For every ConflictOperation there can be many participants. UMLConflictType object has methods for adding/removing participants to the different types of conflicts, adding/removing new types of conflicts, and serving all this information to present to the user ***conflict awareness information***.

A participant in a conflict is a user that has generated an operation that do conflict with other operation generated concurrently by other user. The information maintained in the table is the messages received. A message has information about the generator and the type of the operation, and the user intentions of the participant.

6.7 Sharing Channel

The sharing channel enables the transmission of the application state. That is the basis for the latecomer support. The main class in the Sharing Channel is the ***Synchronization Manager***. It is implemented with a *MessageDispatcher,* a JavaGroups utility Class.

The ***MessageDispatcher*** provides in this case synchronous message sending with request-response correlation, matching the response with the original request. An instance of MessageDispatcher is created with a channel as an argument. It can be used in both *client and server role*: a client sends requests and receives responses and a server receives requests and send responses. MessageDispatcher allows an application to be both at the same time. To be able to serve requests, the RequestHandler.handle method has to be implemented.

In the case of the sharing channel, the first collaborator that joins the group (creates the group) is the coordinator. The coordinator will act as a server when the newcomers request the state of the application.

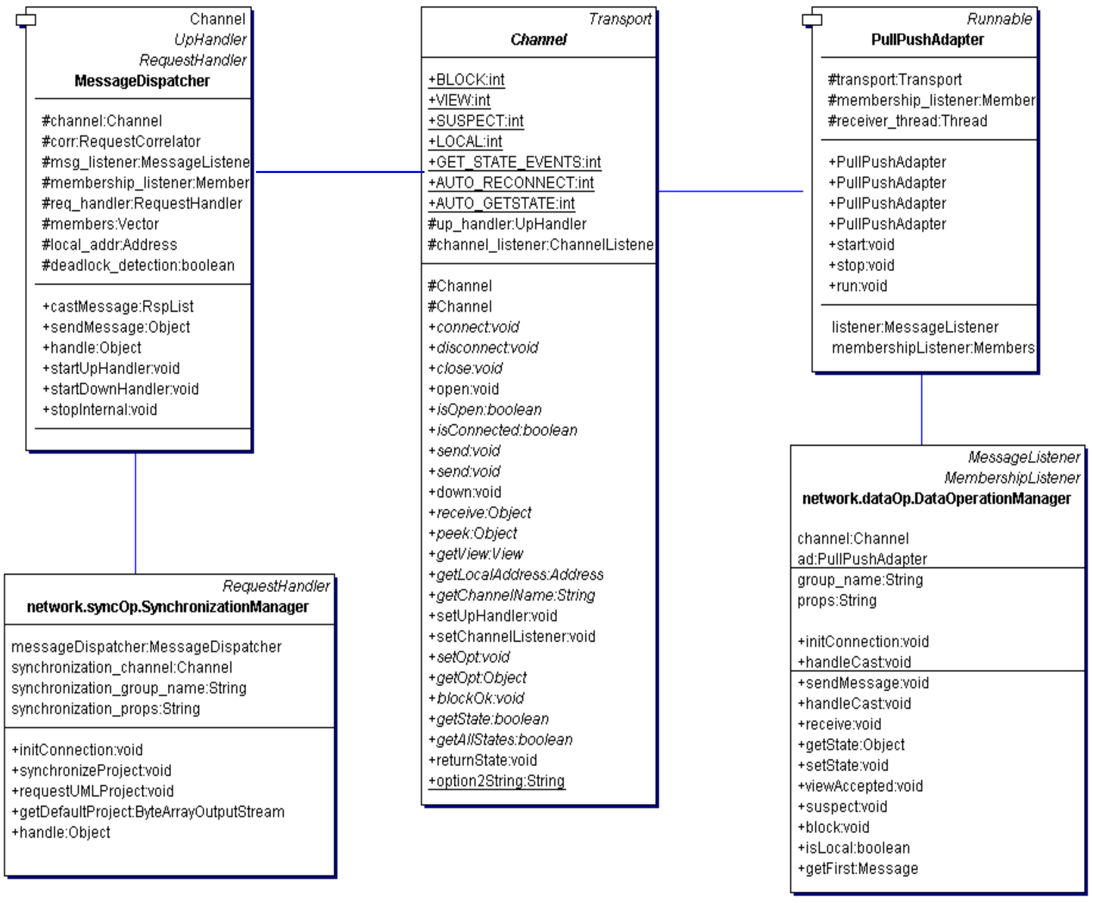
6.8 Conflict Awareness

The User Interface Application ArgoUML has been modified to provide Conflict Awareness to the collaborators on a session. For each element in the graph, Nodes (Classes and Interfaces) and Edges (Associations, Generalizations, Dependencies, Realizations) two modifications have been done:

a) Marking the element as Conflictive: All those elements that have at least a conflict associated are marked with a red Shadow.

b) Pop-Up Menus modified for consulting Conflict Info. The menu consists on submenus. There will be a submenu for every type of conflict associated with a element.

6.9 Communication Platform with JavaGroups



The architecture of JavaGroups is briefly introduced in 3.Architecture In Figure 5.X the structure of the communication platform for a participant is presented: There are two instances of the class Channel:

- Sharing Channel: That communicates different collaborators through the SynchronizationManager. The Synchronization Manager is implemented using a MessageDispatcher internally so a synchronous communication can be established where the requests are mapped with the responses. From all the participants the one that connected the first to the group is the coordinator and will act as a Server in this channel.

- Collaboration Channel: Communicates different participants through the DataOperationManager. This channel is used for sending Real-Time data. The DataOperation Manager is implemented using a Pull-Push-Adapter.

Chapter 7 Conclusions

7.1 Introduction

Real-time collaborative editing systems are included in the field of Computer Supported Collaborative Work (CSCW) systems, which allow users to view and design the same document simultaneously from geographically dispersed sites connected by networks.

In order to achieve high responsiveness, a replicated architecture combined with optimistic execution is adopted. With this schema local operations are executed immediately, independently of network latency, and the latency for reflecting other sites operations should be low.

This Thesis has focused on maintaining consistency in real-time collaborative software modeling tools. There are three inconsistency problems that appear in this type of systems: divergence, causality violation and user intention violation.

Divergence can be solved serializing the operations at all sites, and causality violation can be solved with a causal ordering communication protocol. However user intention violation is dependent on application semantics.

Work to date accommodate user intentions in the following ways:

- *Null effect:* In the face of a conflict all of the operations are rejected/undone. This Null effect does not preserve any user Intention. The work concurrently done by involved users is destroyed
- *Single operation effect*: Only the effect of one operation is preserved. This can be achieved by enforcing a serialized effect among all operations. The user Intentions are not preserved and only one user work can be preserved.
- *All operations effect*, based on multiple versions strategy two versions of the object will be created. In this way the effects of both operations are accommodated in two separate versions.

A Consistency Maintenance Framework for Collaborative Graphic Editing systems is addressed deeply in [5], where a novel multiple object version approach to conflict resolution is proposed. The proposed approach is able to preserve the work produced by multiple users in the face of a conflict and to minimize the number of object versions for accommodating combined effects of conflicting and compatible operations. *This work has been an important source of Ideas for my thesis.*

7.2 Evaluation

This Thesis makes several contributions in the area of Real-Time Collaborative Editing Systems:

Taking advantage of the richer UML semantics for Devising a Flexible Consistency Maintenance Framework where:

- ***There are different levels of inconsistencies.***
- ***The operations are categorized based on the possible conflicts they can generate.***
- ***The Conflicts are categorized based the possible resolution and the level of inconsistency they can generate.***

Most of the work to date has focussed on the area of text and graphics where the application semantics are poorer than in the case of UML diagrams. UML diagrams have two types of information: UML and graphical Information. The UML information can be maintained consistent even if the graphical information is not consistent. The UML information is independent of the diagram layout. Besides no graphical Information can be modelled with the UML specification 1.3.

In this work different levels of Inconsistencies for the Software Modelling environment have been identified:

- Graphical Inconsistency.
- UML Inconsistency:
- Temporal recoverable inconsistency.
- Not recoverable inconsistency.

Each level of inconsistency is treated in a different way. Not recoverable inconsistency is prohibited while temporal recoverable inconsistency is allowed as the documents will merge to a consistent version when the users decide to resolve the conflicts.

The operations for manipulating UML diagrams have been categorized and prioritised based on the possible conflicts they can generate. Different types of conflicts have been identified, each conflict with a specific resolution.

Most work to date have a simpler mechanisms for detecting conflicts, where there are no different types of conflicts and the resolution of the conflict is always performed in the same way no matter the semantic of the operations that do conflict.

UML semantics have also been exploited to advice distributed collaborators of the conflicting intentions of other users, thus improving group awareness. In a face of a conflict, the user intentions are preserved (recording them in conflict data associated with the targeted object).

In the face of a conflict, the system detects the type of conflict, then perform the actions associated with the conflict, in some cases temporal inconsistencies would be maintained, in other cases undo/reject operations will be performed, and in the case of graphical information the conflicts are ignored. But in all the cases the system is records all conflictive user intentions. The UML elements in the diagram that have conflict associated appeared marked on the interface, so the users are aware of other user intentions.

Although the work as been prototyped for the UML semantics, the framework, conflict matrix and approach of richer, application dependent conflict resolution may well be appropriate in different semantics of greater or equal semantic wealth.

A matrix for the detection of conflicts have been devised where the axis are the different categories of operations. The matrix shows the type of conflict that is generated when two operations belonging to two specific categories are generated concurrently.

DArgoUML: A single-user application, open source UML editing tool (ArgoUML) has been extended for distributed operation, using XML based standards.

Some Interesting properties of DArgoUML are:

Support for Real-Time Cooperation.
Flexible Consistency Maintenance Framework.
Latecomer Support.

Research revealed that there are very few distributed UML Case tool. As far as I know a commercial tool called Cittera and a few research projects. None of them address the *User Intention Preservation* problem and in the face of a conflict the user intentions are not preserved. They use traditional methods (locking, floor control, turn taking) for obtaining convergent results.

7.3 Future Work

7.3.1 Heterogeneous Collaboration in Software Modeling Tool

Many Collaborative Group Editors have had little acceptance, one of the main reasons is that those systems force all users to work with the same application often unknown for the users.

Researchers in collaborative work have traditionally been more interested in consistency maintenance and undo/transformation algorithms. Little attention has been paid to the usability of those interfaces.

In [9] this concern about User Interfaces is addressed: "Intellectual work, however, often emphasizes individualism so that individual preferences and priorities are respected. Being forced to use unfamiliar applications for the well being of the entire group, participants whose favourite applications are not chosen for sharing may feel frustrated or less productive because learning new interfaces is often not the focus of the group editing task itself."

The Project ICT (Intelligent Collaboration Transparency) [9] is being developed to integrate the benefits of group editors and application sharing while avoiding the above problems. The single-user editors in question are allowed to be heterogeneous so that collaborators can use familiar tools for group work.

There are many extended UML CASE Tools, like Rational, TogetherJ, Poseidon… etc.
With a Heterogeneous Collaborative Software Modelling Tool users could collaborate on the same Software model using their favourite application.

The heterogeneous architecture could be as presented in Figure 6.x.

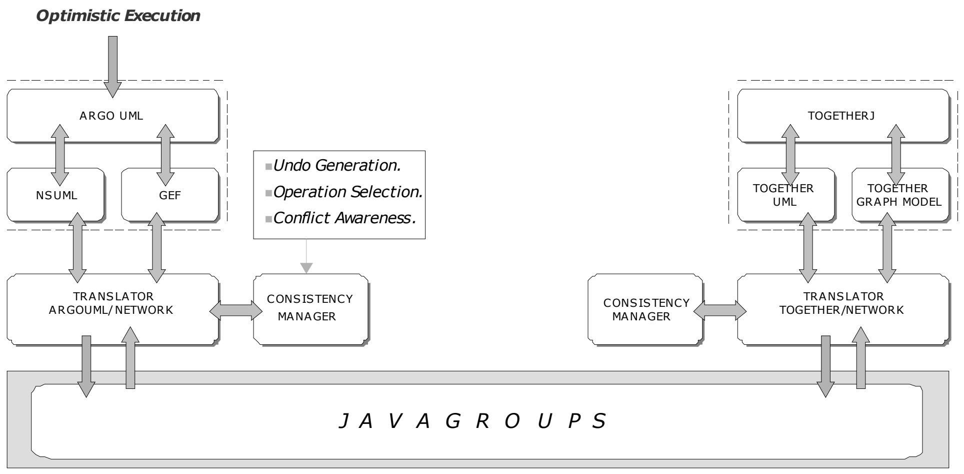


Figure 7.1 Heterogeneous Architecture

The Collaborative users generate local events that modify the local UML model and are distributed afterwards to the network. The fact that events are executed locally immediately after their creation is called *optimistic execution* and it is chosen over pessimistic execution to give good time response.

As this optimistic execution method is used, some undo operations could be generated by the Consistency Maintenance system to achieve consistency.

Usually the implementation of the UML Specification is proprietary of the UML CASE tool. For this incompatibility among implementations of the UML model the translator component is application specific. The Translator component translates abstract operations as "MODIFY_ATTRIBUTE_TYPE" into UML model changes and vice versa.

Actually UML Case tools can share information. Using XMI the Tools can export/import UML models generated by other UML CASE tool. *Software Modelling Tools actually can Share Information but cannot Collaborate (Real-Time) on Projects.*

Sharing is achieved sending the whole project information. Collaboration is achieved sending at Real-Time the operations generated by all the Collaborators in the Group.

One problem of XMI format is that there are many versions and often applications are not compatible. And another problem is that XMI does not include graphical information so that the graphic rendering is lost. Often the XMI information is extended with SVG or PGML.

In DArgoUML prototype on the Sharing channel an XMI file representing the whole UML project state is sent. This file is extended with graphical information in the form of PGML.

For the Collaboration Channel could work with a XMI extension for the transmission of Real Time Operations.

In Order to explain how XMI could be extended we will work only with the generation of Class Diagrams. The following is a subset of the most important operations that can be done over a class Diagram:

UML Element	UML Operation
Classes	OP_CREATE_CLASS OP_MODIFYIFY_CLASS_NAME OP_MODIFY_CLASS_IS_ABSTRACT OP_MODIFY_CLASS_IS_FINAL OP_MODIFY_CLASS_NAMESPACE OP_DELETE_CLASS
Generalization	OP_CREATE_GENERALIZATION OP_MODIFY_GEN_NAME OP_DELETE_GENERALIZACION

Basically there are three operations (CREATE, MODIFY and DELETE) over the UML model elements (classes, associations, attributes, generalizations, associations…. etc).

XMI could be extended for transmitting operations, including a tag like Operation type.

CREATE	<OPERATION Type =“CREATE”> <MODEL_ELEMENT> Foundation.Core.Class </MODEL_ELEMENT> <UUID> "127-0-0-1-4977e2:f6290c1cfe " </UUID> </OPERATION>
DELETE	<OPERATION Type =“DELETE”> <MODEL_ELEMENT> Foundation.Core.Class </MODEL_ELEMENT> <UUID> "127-0-0-1-4977e2:f6290c1cfe </UUID> </OPERATION>
MODIFY	<OPERATION Type = “MODIFY”> <MODEL_ELEMENT> Foundation.Core.Class </MODEL_ELEMENT> <UUID> "127-0-0-1-4977e2:f6290c1cfe" </UUID> <PROPERTY value=”ABSTRACT”> true </PROPERTY> </OPERATION>

This abstracts operations would be translated by the Translator on each group member into an operation to the model, in the same way the operations generated locally by the users could be translated into XMI.

A Future work over this thesis could be research on, design and prototype a Heterogeneous architecture for Collaborative Software Modelling tools.

7.3.2 Group Awareness: Future User Intentions Preservation (Semantic Preservation)

Collaborative work may be less efficient if user's meanings cannot be clearly understood by other users. " Syntactic preservation aims at promising the same operation execution order and the same result of all users' operations at all sites. But user's meanings may not be clearly understood by other users only through viewing the execution of operations". This problem is addressed in [26] and called Semantic Preservation Problem.

In [26] a semantic model to resolve this problem on the graphic editing environment is proposed including definition of semantic expressions, usage of semantic expressions, and semantic conflict resolution approach.

A Future work over this thesis could be the definition of a Semantic model for Software Modelling Environments where users could define their user intentions and make them known to the rest of the collaborators in the group. The semantic model should be application semantics dependent.

7.3.3 Consistency Maintenance Mechanisms for other Software Modelling tools operations

The work done in DArgoUML has been restricted to a set of operations for Class Diagrams. However in UML Specification there are 9 different types of diagrams. Only a few functionality have been examined.

Bibliography

[1] C. A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: Some Issues and Experiences. Communications of ACM 34, 1:39-58, January 1991

[2] T. Brinck, L. M. Gomez. The Design of the Conversation Board.
Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems – Posters and Short Talks, Posters: Improving Team Performance, p.42, 1992

[3] R. Johansen. Groupware: Computer Support for Business Teams
The Free Press, a division of Macmillan, Inc., New York, Published: 1988, ISBN:

[4] C. Sun, X. Jia, Y. Zhang, Y. Yang, D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction (TOCHI) 5, 1: 63-108, 1998

[5] C. Sun, D. Chen. Consistency maintenance in real-time collaborative graphics editing systems. ACM Transactions on Computer-Human Interaction (TOCHI) 9, 1:1-41, 2002.

[6] D. Chen: Consistency Maintenance in Collaborative Graphics Editing Systems. PhD Thesis. Griffith University Australia, 2001.

[7] H. Shen, C. Sun. RECIPE: a prototype for Internet-based real-time collaborative programming. Proceedings of the 2nd Annual International Workshop on Collaborative Editing Systems in conjunction with ACM CSCW Conference, December 2 -6, 2000, Philadelphia, Pennsylvania, USA.

[8] W. Keith Edwards: Flexible Conflict Detection and Management in Collaborative Applications. ACM Symposium on User Interface Software and Technology 1997: 139-148

[9] D. Li, R. Li. Bridging the Gap Between Single-User and Multi-User Editors: Challenges, Solutions, and Open Issues. Department of Computer Science. Texas A&M University. College Station, Texas 77843-3112 USA

[10] D. Li, R. Li. Transparent sharing and interoperation of heterogeneous single-user applications. Proceedings of the 2002 ACM conference on Computer Supported Cooperative Work (CSCW-02), pp. 246-255, ACM Press, November 16-20 2002.

[11] D. Yang, A. E. Saddik, N. D. Georganas. Latecomer Support and Client Synchronization for Synchronous Multimedia Collaborative Environments. Precedings of the 4th International Workshop on Collaborative Editing, Adjunct to the ACM Conference on Computer Supported Cooperative Work, New Orleans, Louisiana, USA, Nov. **2002**.

[12] N.Boulila, A.H.Dutoit, B.Bruegge. D-Meeting: an Object-Oriented Framework for Supporting Distributed Modelling Software.International Workshop on Global Software Development, International Conference on Software Engineering. Portland, Oregon, May 9, 2003.

[13] K.M.Hansen. The Knight Project: Supporting Collaboration in Object-Oriented Analysis and Design. University of Aarhus, Åbogade 34, DK-8200 Aarhus N, Denmark

[14] P. Stevens. Small-scale xmi programming: A revolution in uml tool use? *Automated Software Engineering*, 10:7–21, 2003.

[15] B.Ban. JavaGroups Userguide 2_0.
http://www.javagroups.com/javagroupsnew/docs/ug.html

[16] G. Coulouris, J.Dollimore, T.Kindberg. Distributed Systems Concepts and Design (3rd Edition), Addison-Wesley, 2001

[17] Real-time, Distributed, Unconstrained Collaborative Editing
http://www.cit.gu.edu.au/~scz/projects/reduce/

[18] GRAphics Collaborative Editing
http://www.cit.gu.edu.au/~scz/projects/grace/

[19] REal-time Collaborative Interactive Programming Environment
http://reduce.qpsf.edu.au/~hfshen/recipe/

[20] XML Metadata Interchange (XMI)
http://www.omg.org/technology/documents/formal/xmi.htm

[21] Unified Modelling Language
http://www.omg.org/uml/

[22] Cittera
http://www.canyonblue.com/

[23] ArgoUML Project Home
http://argouml.tigris.org/

[24] A. Ramirez, P. Vanpeperstraete, A. Rueckert, K. Odutola, J.Bennett, L.Tolke
A tutorial and reference description

[25] Novosoft UML library
http://nsuml.sourceforge.net/

[26] X.Wang, J. Bu, C.Chen. Semantic Preservation in Real-Time Collaborative Graphics Designing Systems. ACM CSCW 2002