Team Lab

- A Collaborative Environment for Teamwork

By

Guang Yang

Thesis
submitted in partial fulfillment of the requirements for
the Degree of Master of Science (Computer Science)

Acadia University
Fall Convocation, 2000

0-612-54543-1

Table of Contents

Table of Figures

Abstract

As we are entering the 21st century, demand for high quality software increases dramatically. Software development requires teams of developers working closely together, so it becomes more and more important to have a good collaborative environment in order to improve efficiency, productivity, and quality. Unfortunately most software development tools are designed for a single user or do not have a good environment for teamwork. This gap lead to the design of Team Lab.

Earlier work at Acadia University lead to the development of a collaborative virtual environment called MUM. This thesis is an attempt to validate MUM architecture by developing a MUM tool for integrated software development called Team Lab. Team Lab is a basic code management tool that provides a common repository for shared code, supports concurrent programming activity and class level version control.

The conclusion of the thesis is that the architecture of MUM is a workable foundation for a CVE with full support for software development tools.

DEDICATION

To my family and friends.

ACKNOWLEDGEMENTS

I would like to thank everyone who helped and supported me during design, implementation and writing of my thesis. Thanks to everyone who designed and implemented MUM, which is the basis of my thesis. Thanks to everyone who provided useful information to make it better. These people include Dr. Ivan Tomek, Dr. Rick Giles, David Murphy and Min Wu.

Specially, I would like to thank Dr. Ivan Tomek, my supervisor, for his many good suggestions and guidance. Thanks to Dr. Peter Hitchcock, my external examiner, Dr. Rick Giles, my internal examiner, Leslie Oliver, Acting Director, for their effort to examine my thesis.

Chapter 1 Introduction

The ideal collaborative environment should be an integration of multiple team-support features into a seamless whole. It should provide means for communication, creation of documents, private and shared spaces, security, and a variety of work tools. In other words, rather than working with individual isolated tools, work teams should work in a Collaborative Virtual Environment (CVE).

What are the main properties of software development tools required by a team of software developers? One of the most important features is that programmers should be able to share code easily. Code should be available to others as soon as a programmer finishes editing a unit of code and releases it to the team. Working in this way, developers always get the latest code, which is the premise of efficient teamwork. At present, this feature is supported by tools such as ENVY/Developer [1], StORE [2] and ObjectStudio V6 Team Edition for VisualWorks Smalltalk [3]. These tools implement code ownership, versioning, releases, and code configuration. However, the limitation of these tools is that it is restricted to code management and lacks support for the other features mentioned above and essential particularly for geographically dispersed teams.

Another desirable feature for collaborative software is that users should be able to work both online and offline with code synchronization performed when they switch from offline to online mode.

As mentioned above, a CVE should provide multiple ways of collaboration. In addition to code sharing, users need to exchange ideas when they are working. Although many chat room programs are available and widely used, support for communication should be an integral part of the environment rather than a standalone application. This would not only make their use easier and provide privacy, but it would also allow the team to capture project-related documentation and create and maintain corporate memory. Whenever there is a need for discussion, the team should also be able to establish a private online meeting by simply launching a suitable interface, and capture its content as a part of project documentation. Since the meeting space is not public, the discussion will not be interrupted by uninvited users. A shared whiteboard maintained as a part of the environment would also be nice for expressing and recording ideas.

Separation of individual projects and their parts, instant access to individual team members, and possibility of informal and chance encounters should be supported to emulate and complement the ideal situation of team members working in physical proximity and meeting face-to-face.

Another important feature is instant notification of team members when a relevant unit of work is released. As soon as a requirement document, design diagram, or a class is released or changed, interested users should be notified if they want. This can be implemented by a subscription mechanism that enables users to express their interests in work events such as classes being released, methods being modified, categories being removed, new code being added, etc.

By its very definition, all possible uses of a CVE emulating physical reality are unpredictable and the environment thus must be easily extendible and customizable. Whenever there is a need for new tools, users should be able to write them and add them to the existing environment without disrupting the operation of the system. This requires a suitable CVE architecture and implementation technology that facilitates run-time modification.

In consideration of the above-mentioned requirements, Team Lab was developed as an extension of MUM – Multi-Universe MOO [4], an event driven MOO (MUD Object Oriented) whose architecture was designed to facilitate implementation of the functionality outlined above. In the following sections, we will survey the principles of MUM, outline the functions and design of Team Lab, present several examples of its user interfaces, and summarize conclusions and plans for the future.

Chapter 2 Background

The rapid development of computer technology today requires good tools for teamwork due to the huge demand for high quality software. Teamwork often involves many developers that work closely together. The better the tools they use, the more productive they are.

Currently there are several products available on the market that support team work. In the Smalltalk environment, these tools include ENVY/Developer, StORE (Smalltalk Open Repository Environment) and ObjectStudio V6 Team Edition. ENVY/Developer, is a product of Object Technology International Inc, StORE and ObjectStudio V6 Team Edition are products of Cincom.

2.1 ENVY/Developer

ENVY/Developer is a very popular collaborative component development tool for Smalltalk developers. It supports concurrent development, shared repositories, version control, configuration management, and distributed development for small to very large teams. ENVY/Developer has following characteristics:

Integrated Repository:

It provides concurrent component-level access to streamline group development and the access control is at the component level. In addition, ENVY/Developer's incremental development with dynamic linking avoids costly and error-prone load builds. It implements code ownership and logs complete development history.

Organization of Software for Reuse

A component management environment must support the organization of software components to facilitate the management of their complex relationships. ENVY/Developer's applications are a mechanism for grouping collections of classes that implement an application to form a reusable component. Each component can then be configured for different environments and maintain dependencies. Component ownership, a critical success factor for object-oriented projects, is supported at the class and application level.

Version Control

It provides fine-grain versioning, including methods, classes, class extensions, applications and configuration maps. Since it keeps complete history of all changes, everyone in the team can get immediate access to all changes.

Configuration Management

Configuration management provides the basis for assembling components into final applications. ENVY/Developer's software configuration maps allow developers to assemble their Smalltalk components into complete systems. It is here that developers create subsystems, and assemble subsystems to create systems.

2.2 StORE and ObjectStudio V6 Team Edition

StORE and ObjectStudio V6 Team Edition are very nice tools that support teamwork. They enable management of software teams and their code from single to multiple developers, across a range of multiple projects.

StORE

StORE is an optional component of VisualWorks. It is a tool for version and component management for remote development with import/export to ENVY/Developer, although it does not require ENVY/Developer. Its new mid-tier versioning and configuration management tools ease code management and the check in/check out model simplifies remote development. StORE currently uses Oracle 7 as a source code repository providing a secure storage resource.

StORE users can work online with the shared database or offline with a local database. In offline mode it is also possible to work without a local database although some functions are restricted. Code is grouped by packages and bundles, which provide easy management of large amount of code. Users can use the Settings tool to set up package definitions. When a user finishes developing a package or wants to make the code available to the team, he or she can publish the package or the bundle containing it. This writes the new version to the StORE database and makes it publicly available.

StORE can be used for individual or team development as a stand-alone tool or in conjunction with ENVY/Developer.

ObjectStudio V6 Team Edition

ObjectStudio V6 Team Edition includes a repository management tool that enables the administration of project code and personnel with ease and efficiency. It provides object and version management, configuration management, access control, and cross-application integration throughout the application's life cycle.

ObjectStudio V6 Team Edition has following features:

- Realtime Multiuser Development.
- Versioning and History Management.
- User Authorization and Ownership Management.
- Release / Component Management.
- Repository Browsing.
- Backup and Copy Management.
- Split Development.
- Repository Import / Export.
- Team Management.

ObjectStudio V6 Team Edition tracks every change made to versioned classes and creates a new working version of the class accessible only to the developer who made the change. In the meantime, the original version is still accessible to other developers. Other developers can also create their own working versions. At any point in the process, a developer may revert to a previous version, compare a current version with a previous one, or merge differing versions into a third, new one.

Summary

In general, the three tools mentioned above are very well designed and powerful enough to support teamwork. They all have a shared code repository and provide version control and management. All in all, they are nice development tools but do not support collaboration. Users cannot use them to communicate with others while they are working on coding without the aid of other software. In order to find what code was changed by other developers, they have to browse the classes in the code repository or wait to be informed by their developers.

In considering these shortcomings, Team Lab addresses both code development and collaboration. It is not only a code management and development tool, but also a part of a collaborative virtual environment called MUM. The MUM environment will be explained later.

Chapter 3 Introduction to MUD and MOO

MUDs/MOOs are client-server applications that support multiple user collaborations. Their characteristics include real-time interactivity, networked service, multi-user capability, extensibility and exclusivity [5]. Since they are suitable as communication tools, we have used this model to develop MUM (Multi-Universe MOO), the foundation of Team Lab. This chapter provides the MUD/MOO background needed to understand Team Lab.

3.1 Definition

MUD is the acronym of Multi-User Dungeons and MOO stands for MUD, Object-Oriented. MUD also has some other translations such as:

- Multi-User Domain.
- Multi-User Dimension.
- Multi-User Dialogs.

MUDs started appearing about 20 years ago [5] and typically they are role-playing adventure games. They are client-server applications and allow multiple users to connect to the server at the same time. On the server side MUDs provide many kinds of objects that represent real world objects such as places, rooms, doors, people, tools, etc. These objects usually reside in a big place called the universe, which may include sub-places. User proxies in the universe are called avatars or agents. When a user connects to the server, he or she can use appropriate commands to instruct the avatar that represents him or her to do whatever he or she likes, for example, talk with other users, move around, create objects, find treasures, etc.

3.2 Brief History

Here is a brief history of different kinds of MUDs [6].

- First MUD: Roy Trubshaw and Richard Bartle developed the first multi-user game (MUD1) in 1979 using assembly language programming. The original version merely allowed a user (player) to move about in a precoded virtual universe. Later versions provided for more variation including objects and commands that could be modified online or offline.

- TinyMUD Original: The first of the Tiny family of MUDs, was written in August 1989 by Jim Aspnes. TinyMUD was created for players to "hang out", converse and build virtual worlds together. The environment is more social in orientation. It is much faster because it keeps the entire database in memory. The design assumed that the database would not grow too large.

- First MOO: The first MOO was created by Stephen White in 1990. In a MOO, every conceptual object is also an object in the implementation sense. Each object has a unique identifier number, which can be used to reference it, as well as other properties such as a name, a description, and a location.

- A popular form of MOO, LambdaMOO, was created by Pavel Curtis in 1990, derived from Stephen White's initial MOO server. LambdaMOO is a network-accessible, multi-user, programmable, interactive system, well suited for the construction of text-based adventure games, conferencing systems, and collaborative software. It uses its own programming language - Lambda MOO language to allow its users to create objects and extend functionality.

3.3 Uses of MUDs and MOOs

Initially, MUDs and MOOs were used only for gaming and socializing. In recent years, they have been increasingly used in education [7], research [8] and collaborative work.

Chapter 4 Overview of MUM

Team Lab is integrated into MUM and this chapter provides the necessary background. MUM (Multi-Universe MOO) was developed in 1998 and 1999 [4]. The purpose of this project was to test the uses of a virtual environment as a collaborative tool supporting geographically dispersed teams.

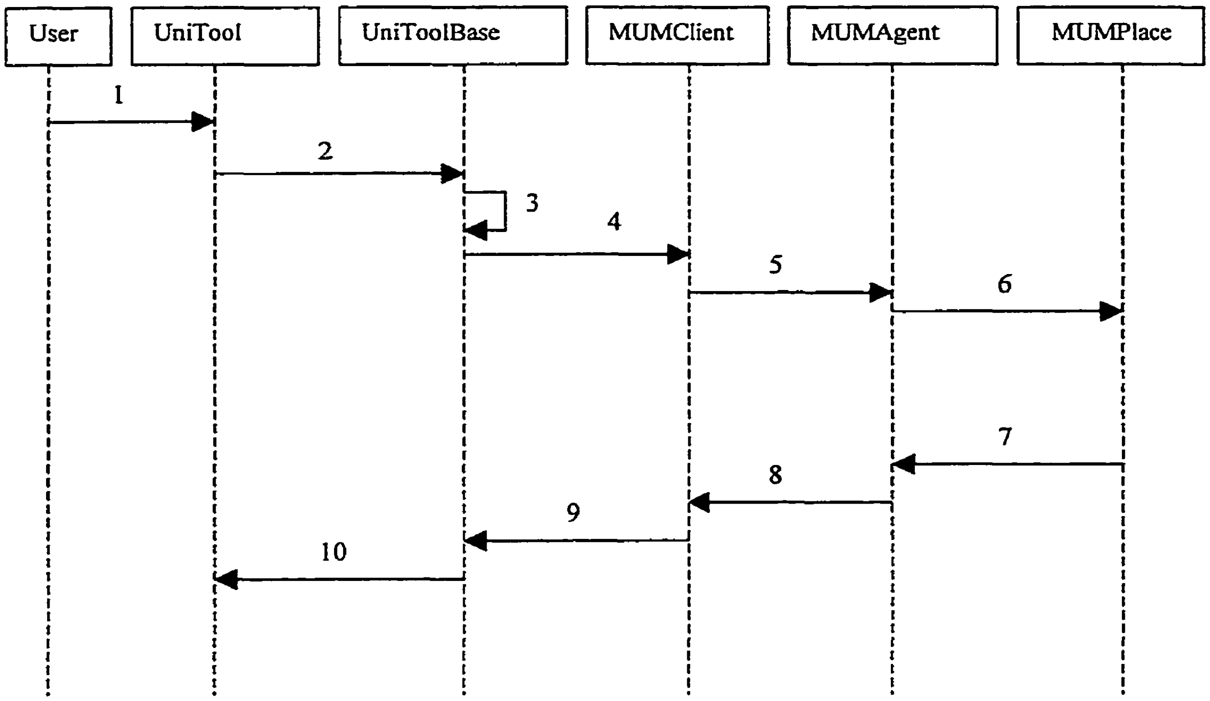
MUM is a collaborative virtual environment, a MOO, a CVE (Collaborative Virtual Environment) emulating physical world properties with simple text- and GUI interfaces. Although MOOs have grown out of recreational uses, their use has progressively extended to social and educational uses [9]. Empirical studies [10][11] have shown that MOOs are effective for collaboration even though limited by their lack of features supporting specific collaboration tasks. At least one commercial product was built on traditional MOO principles and is in production use [12]. MUM is an attempt to take advantage of the proven MOO potential and extend conventional implementation features by adding certain new principles and providing tools aimed at groups performing specific work tasks.

Objects in MUM are called EDOs (Event-Driven Objects). All objects in the universe, which can best be described as a real-world emulation, are EDOs. In fact, even the universe is an EDO.

Being an EDO means that communication with this object is only possible through events. Technically this means that each EDO has a process and a queue of pending events. When an event is put into an EDOs event queue for execution, its process is activated and when it gets its turn, in a round robin execution controlled by the VisualWorks process mechanism, its event handler executes events accumulated in its event queue.

Each EDO has certain basic properties such as name, location, owner, description, and ID. A dictionary matching of IDs to EDO references is held in the Registry which itself is an EDO. EDO's refer to one another only through IDs and when an EDO needs to access another object, the reference is obtained from the Registry.

EDOs exist both in the “server-side” universe and its “client-side” counterpart. On the server side, EDOs represent things such as places, agents, doors, and other emulated real-world objects. On the client side, EDOs are mainly the “base EDOs” corresponding to UI (User Interface) tools such as the launcher or the universal tool. The communication between the server- and the client-side occurs exclusively between the client EDO which resides on the client side (and in the “client universe”) and the corresponding Agent which resides on the server side (and in the “server universe”). As a consequence, when a user wants to interact with an EDO in the (server) universe, he uses his UI (a UI is not an EDO) to send a message to its “base” (an EDO), which sends an event to the client, which sends an event to the agent, which sends an event to the destination EDO (Figure 4.1). Communication in the opposite direction reverses this sequence.



1: User enters his or her message and presses “enter”

2: say: message

3: SelfSayEvent

4: ToolSayRequestEvent

5: ClientSayRequestEvent

6: AgentSayRequestEvent

7, 8, 9: ConfirmationEvent

10: speaker:say: message

Figure 4.1 “say” sequence diagram

The fact that the user needs some information about any EDO with which he wants to interact means that the client side must have sufficient information about it. Having the whole EDO duplicated on the client side would be very wasteful because it would basically require duplicating the universe on the client side and updating it completely when anything in the universe changes. To prevent this, EDOs are represented on the client side by their proxies, which contain only the necessary information such as name, ID, description, etc. In principle, the client side is aware only of the user's agent, its current MUM location (place), and all EDOs that are "visible" to it in this location. No other information is directly available. To avoid transporting EDOs to the client side, proxies have IDs in the universe registry and when an EDO needs to be accessed, access is always through the registry via its ID.

The concept of visibility is very important. Our interpretation essentially follows the real world situation in which a person can see its location (place), and all objects in this location including other persons and exits to other rooms. (In the real world, one could also see through an open door or through a window and the concept becomes more complicated. With our present user interface, we are ignoring this interpretation.)

Characteristics of MUM

Like existing MOOs, MUM provides support for multiple users: Multiple users function in the environment via proxies (avatars, agents) that interact with other users and the environment. Each person that enters MUM has a unique avatar that can be customized with various tools for easier interaction with the environment.

MUM allows multiple co-existing and interconnected universes (hence its name). The servers hosting these universes may reside on one or several machines, possibly on user machines along MUM clients. Users can connect to any universe via metaservers that hold directories of currently active universes, and user proxies can move from one universe to another with their current holdings (Figure 4.2).

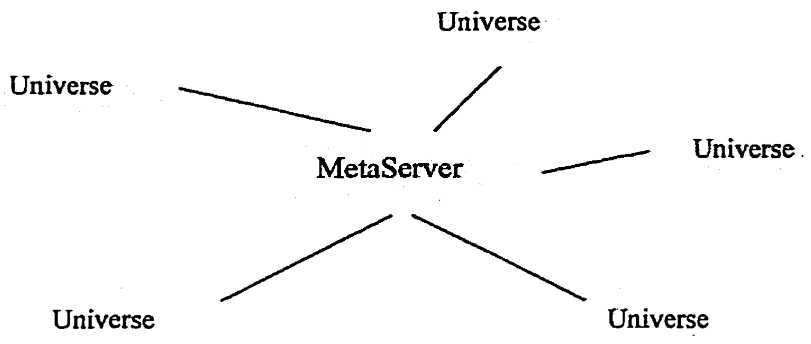


Figure 4.2 Metaserver and universes

MUM is an Internet based client-server application: MUM uses standalone GUI-based software to implement the client. The system consists of two main parts, the Server and multiple Clients. The Server implements the universal functionality, has a repository of tools, and stores the universe in which all avatars, tools, objects, and places exist and interact. Clients perform client-side operations, offloading processing from the Server and minimizing network traffic, and provide the interface through which users control their avatars.

MUM is Event-Driven (Figure 4.3): All objects in a MUM universe, such as agents, tools, and places, are Event-Driven Objects (EDOs). This means that all operations in the environment have the form of events and provide "hooks" allowing other events and users to subscribe to their occurrence. All EDOs have their own processes, event queues and event handlers that are responsible for handling incoming events, notifying subscribers, and interacting with other EDOs via new events. EDOs are autonomous and personalized in that they work independently and know how to control the execution of events and respond in an individualized fashion. Response to events may either be synchronous or asynchronous, as event execution is suspended whenever it needs to wait for the response from other EDOs, and resumes when the result comes back. Suspended events do not prevent other events from being executed. The event-driven mechanism enables users (or EDOs) to subscribe to any event that happens in MUM. Users can register their interest in events sent to an EDO and whenever the EDO processes those events the users are notified.

MUM supports extendible objects: The objects that make up the environment can be augmented or otherwise modified at run time with new functionality defining events that they can handle. The system can also be extended by adding new Tools with their Tool Manuals that provide user information as well as information required to download a tool to the client. This allows developers to customize the Universe that they wish to run, and personalize certain aspects of the system. The subject of tools is further explained below.

MUM's design is multi-threaded: Every object in the environment runs in its own thread. This allows for a more stable design by isolating one object's operation from the operation of other objects. Threads of inactive objects are suspended to reduce system load.

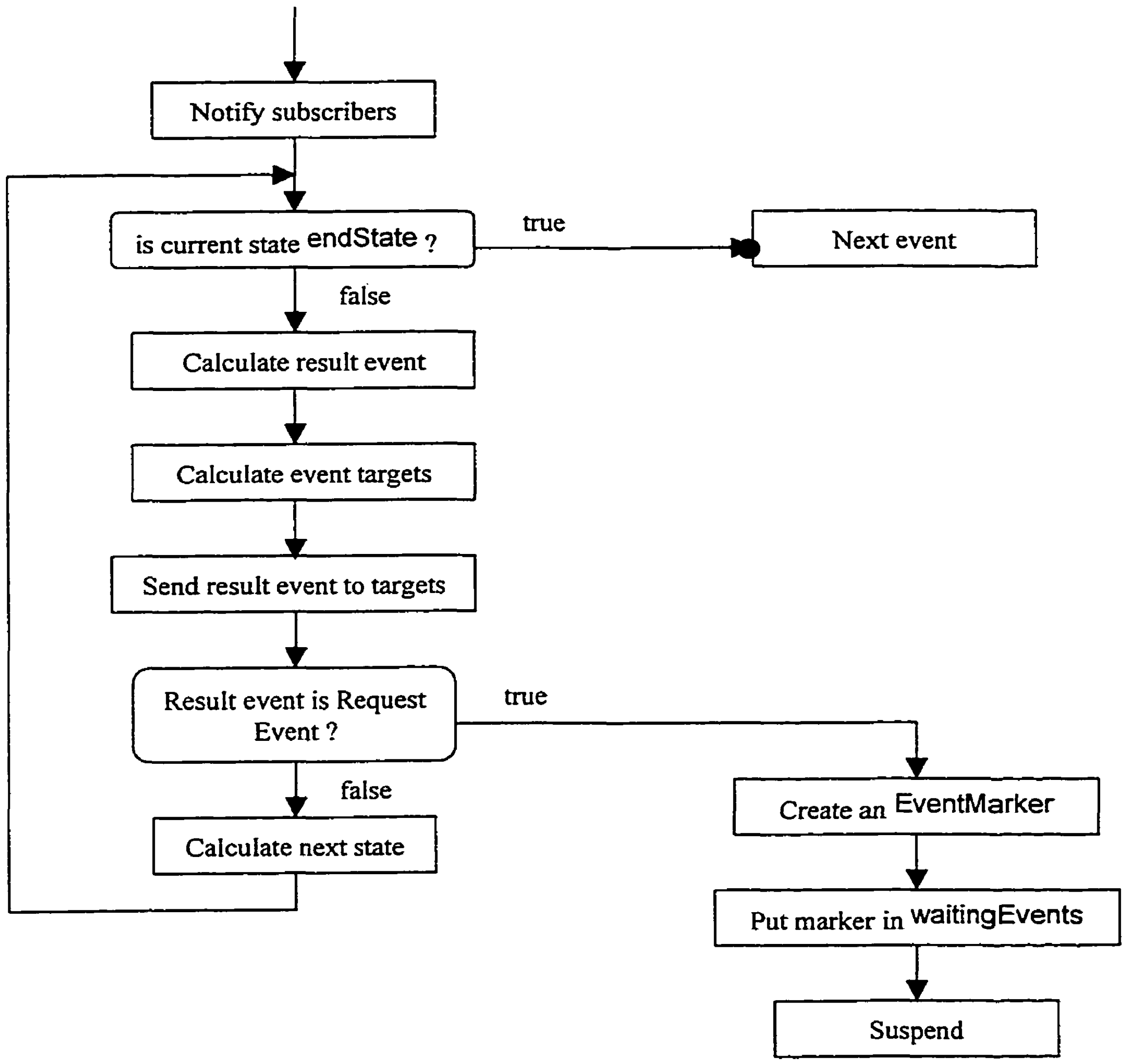
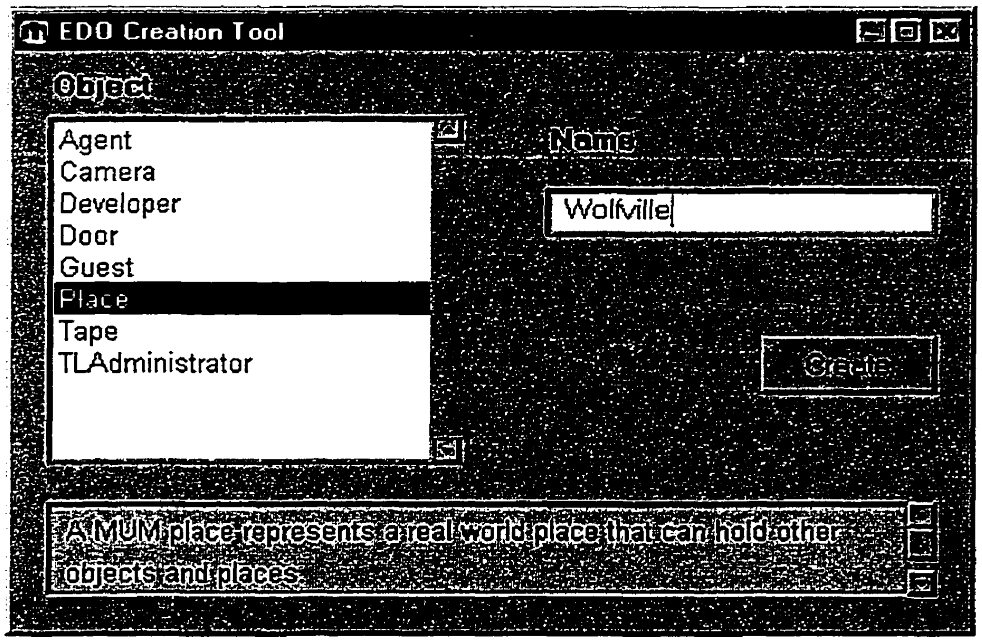


Figure 4.3 Simplified diagram of event execution

Automatic Code Update: When a client connects to a MUM universe, the program first checks if it has the latest code. If not, the program asks the user to download the new version. By clicking a button, the new code is transferred from the server and installed automatically. Since MUM is a large application, the code is divided into several parcels. During the update process, only the new parcels are downloaded in order to save time.

Tools: Tools are sophisticated EDOs with user-friendly interfaces that hide the internal complexity of event generation and dispatching. An example of a tool is the EDO Creation Tool (Figure 4.4) which allows the user to instantiate (build) EDOs. Users send events to a tool simply by operating the widgets in the tool's interface. A tool's interface can be loaded dynamically, so users do not have to have the interface code when they decide to use it. Each tool may have more than one interface and users can select any of them. Every MUM tool has a corresponding tool base, which is an EDO and responsible for translating messages to MUM events and vice versa.

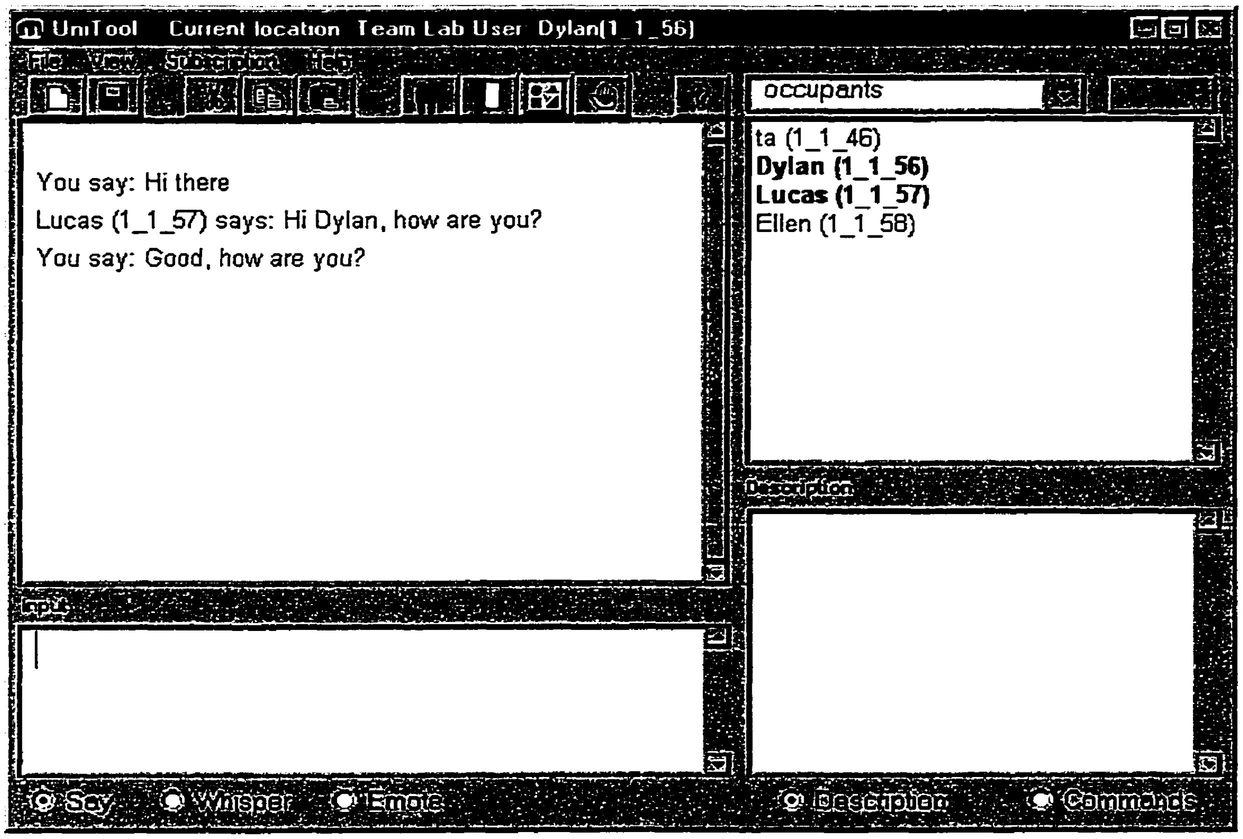


Wolfville

A MUM place represents a real world place that can hold other objects and places.

Figure 4.4 Object creation tool

At present, MUM provides a limited number of general-purpose tools including the UniTool (Figure 4.5), which provides standard MOO communication and supports navigation, access to objects, and subscription to events, an EDO Creation Tool, which simplifies the object creation in MUM, a Property Tool, which enables the users to modify EDO properties, and a MUM Camera, which can be used to record communications in places. Team Lab provides a set of additional tools that will be explained later.


UniTool Current location Team Lab User Dylan(1_1_56)
File View Subscription Help
occupants
Description
Input
Say
Whisper
Emote
Description
Commands

You say: Hi there
Lucas (1_1_57) says: Hi Dylan, how are you?
You say: Good, how are you?

ta (1_1_46)
Dylan (1_1_56)
Lucas (1_1_57)
Ellen (1_1_58)

Figure 4.5 Unitool

Chapter 5 Team Lab Overview

Team Lab is a collaborative development tool for Smalltalk programmers. Since it was built over MUM, it has all MUM characteristics. Its integration into a virtual environment makes it a good basis for teamwork. To put Team Lab into perspective, this chapter describes Team Lab along with other popular tools for Smalltalk software development teams.

5.1 Existing environments and Team Lab

Smalltalk environments such as VisualWorks [2], like other interactive development environments, are single-user environments and developers use an "image" residing on their own machine, and System Browsers and other tools to write and edit the code in their images (1). In order to share code with others in a single-user environment, a user must file the code out to a file and transfer the file to other team members. This mode of operation may be acceptable for simple applications that involve only one or a few programmers, but becomes difficult to coordinate when there are more members. Consequently, many Smalltalk teams use a third-party product such as ENVY/Developer or StORE, extensions of the basic environment in which developers own code modules and share a database with archived versions of an application (Figure 5.1). Neither StORE nor ENVY/Developer provide any collaboration support beyond code management.

(1) A Smalltalk image is a snapshot of the state of its interactive development environment and the basis of Smalltalk's operation.

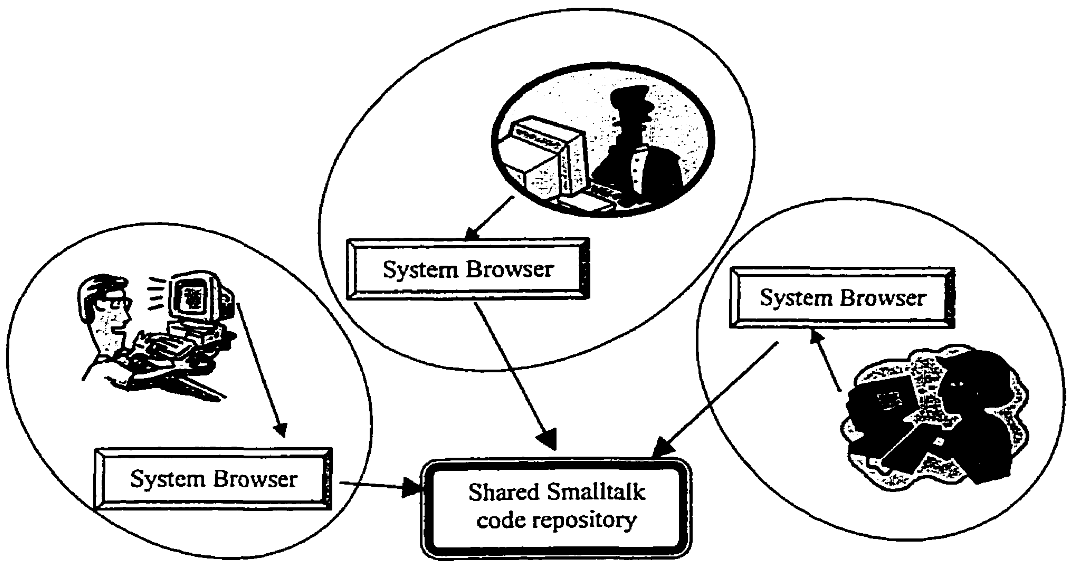


Figure 5.1 Teamwork using Smalltalk

Team Lab was designed for code development in Smalltalk and is written in VisualWorks Smalltalk. It is a client-server application. Users have their code stored in a code repository on the server instead of locally. The code repository becomes available when developers login, so they always have up-to-date code. This does not mean that developers cannot work offline. They can also work on their own images without connecting to the server and whenever there is the need for code release or update, they can connect to the server and transfer the code to or from the code repository. A remote browser, TL Browser, is designed to browse and edit the code in the code repository (Figure 5.2). TL Browser has all the basic functions of a Smalltalk System Browser but provides additional features for class-level version control, management, security, etc.

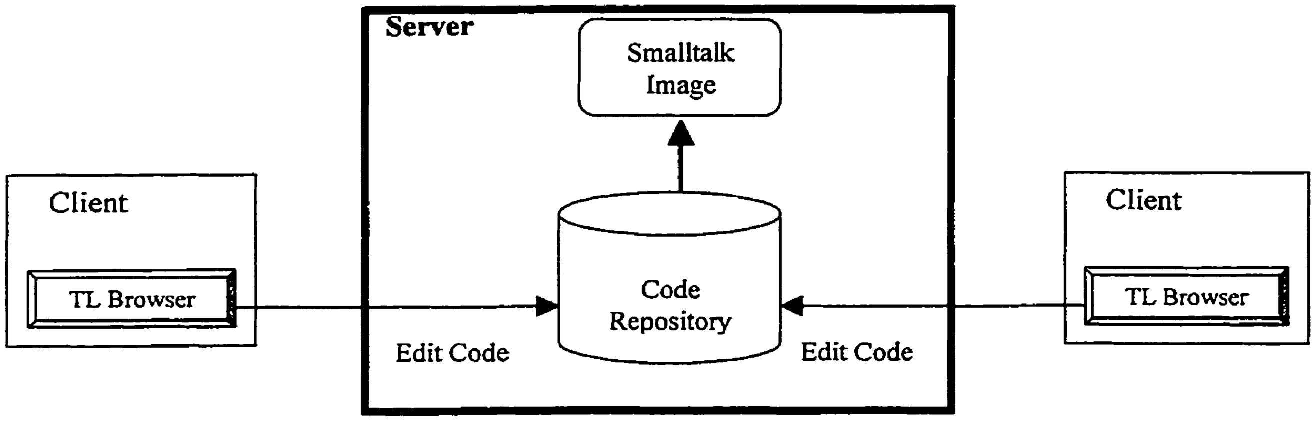


Figure 5.2 Team Lab overview

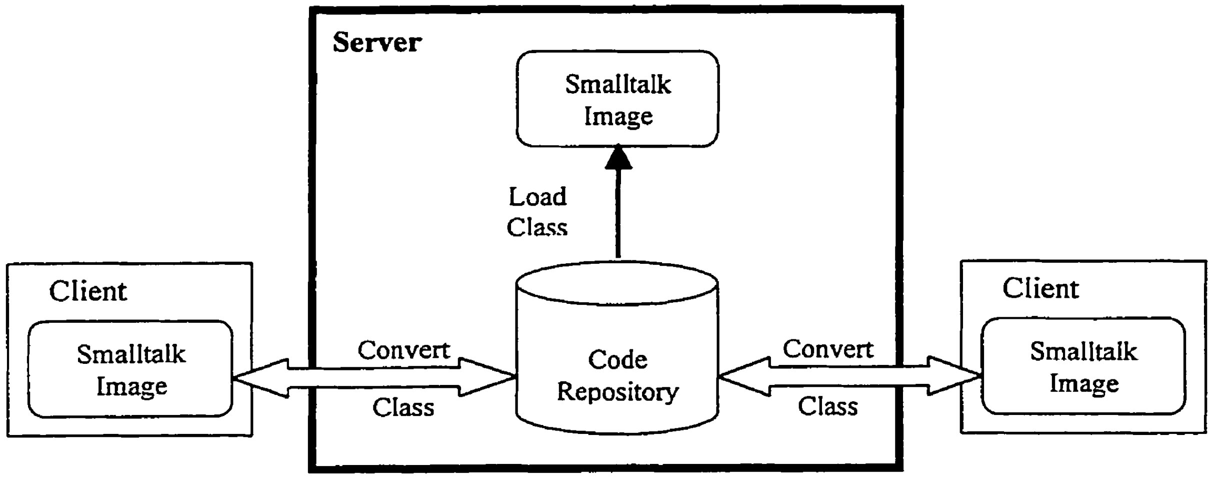


Figure 5.3 Class conversion

Team Lab uses MUM as its foundation, so all Team Lab objects including the code repository, categories, classes, protocols and methods, reside in a MUM universe. The definitions of these objects are not same as those corresponding definitions in Smalltalk since they use MUM conventions. They are event-driven objects (EDOs) and Team Lab can convert them to corresponding Smalltalk objects and vice versa. Users can transfer classes between the client image (Smalltalk objects) and the code repository (Team Lab objects), converting them to the appropriate representations, or from the code repository to the server Smalltalk image (Smalltalk objects) via the load operation (Figure 5.3). These operations are explained in more details in the next chapter.

5.2 The benefits of using MUM infrastructure

As a MUM tool, Team Lab inherits all of MUM's features and benefits from them as follows:

- Event-based operation: Most Team Lab classes are subclasses of EDO or MUMPlace and inherit their basic characteristics. The most important one is that they are event-driven. Instances of these classes have their own event handlers, event queues and threads. This makes it possible to subscribe to events relevant to the development process.

- Subscription and notification: Users can subscribe to any event that Team Lab objects understand and whenever the event occurs, the subscribers are notified. As an example, users can subscribe to events such as the release of a class, the change of a method, and the addition of code.

- Multiple ways of collaboration: While engaged in development, users can use the UniTool to communicate with others, the MUMCamera to record communications, and the Property Tool to modify the properties of objects.

- Extendibility: Users can create MUM objects using the EDO Creation Tool to build new places and objects. When new tools are implemented for MUM, they are immediately available to Team Lab users.

Chapter 6 Concept and Design Details

Being a part of MUM, Team Lab uses MUM concepts. Especially important for Team Lab is MUMPlace, the class modeling physical space in the real world.

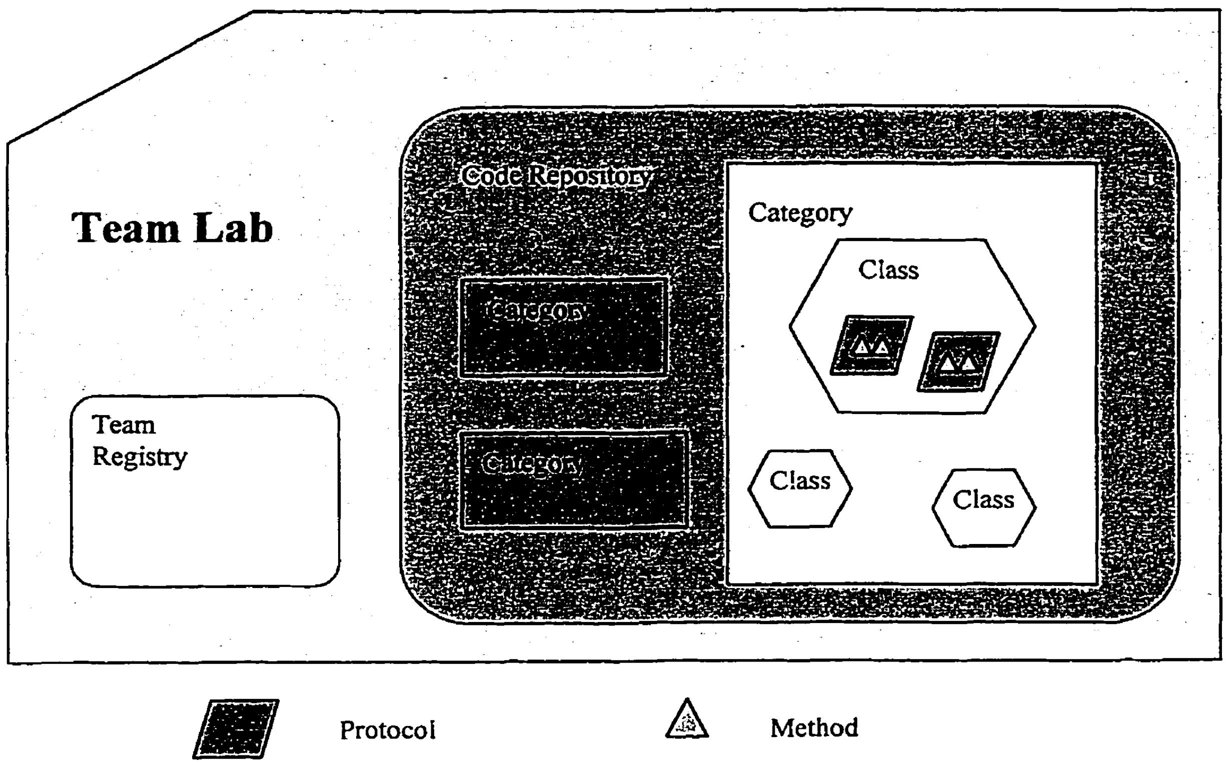


Figure 6.1 Overall Team Lab structure

From users' point of view, Team Lab is a place in the MUM universe that holds sub-places such as team registry, code repository, categories, classes, protocols and methods (Figure 6.1 and 6.2). The reason why we choose MUMPlace as their super class is that places can contain sub-places as well as other objects, using the same tree structure as categories, classes, protocols and methods in Smalltalk. In addition, users can enter each place in order to browse contents in the place and if they are interested in any object they can subscribe to the events the object understands and be notified whenever the events occur. For example, users can subscribe to events such as classes being released, created, modified, etc. The event notification mechanism enables users to have the latest information about the work done by their partners. This avoids users wasting their time to wait for other's work.

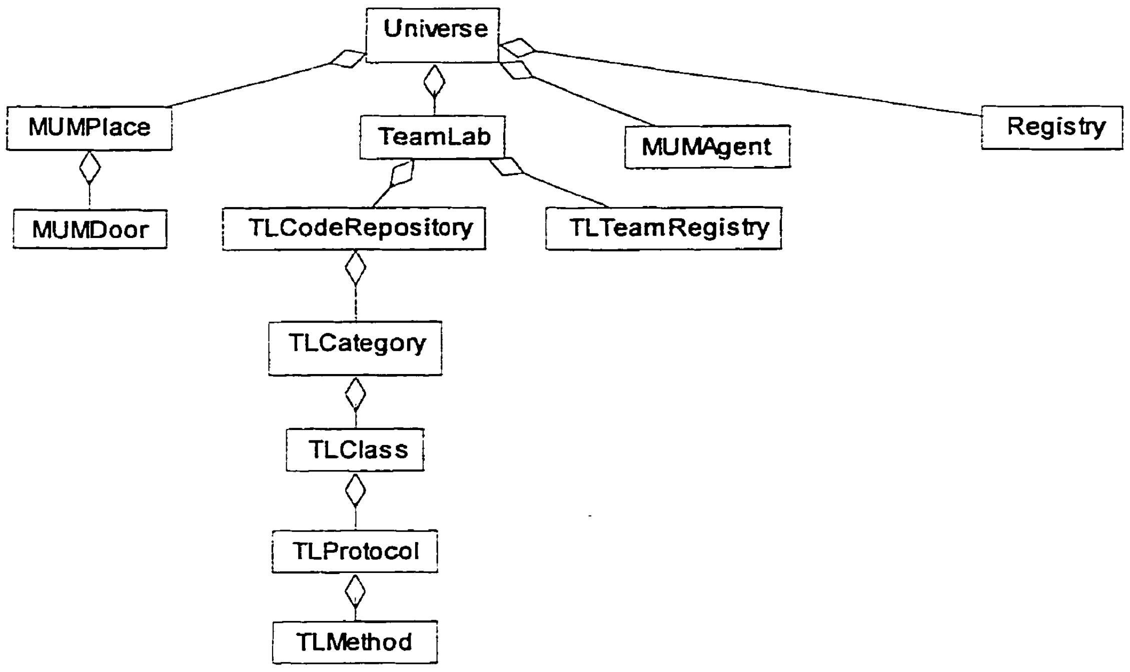


Figure 6.2 Class diagram showing aggregation relationship

Team Lab uses the following four main concepts: Team Lab agent, Team Lab team, code repository and client side tool. These will be described in the following sections.

6.1 Team Lab Agent

Users can use one of two kinds of Team Lab agents, namely TLDeveloper and TLAdministrator. Both are subclasses of MUMPersonifiedAgent (Figure 6.3).

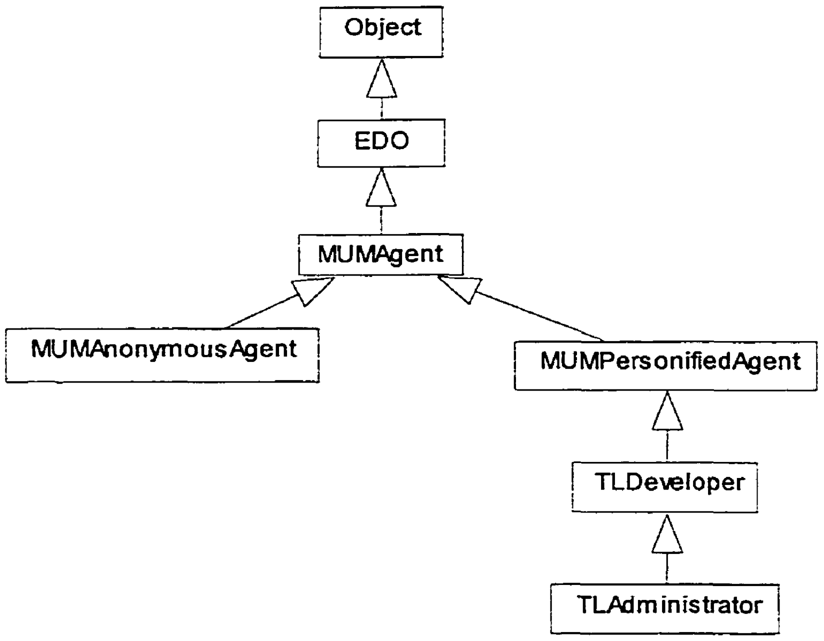
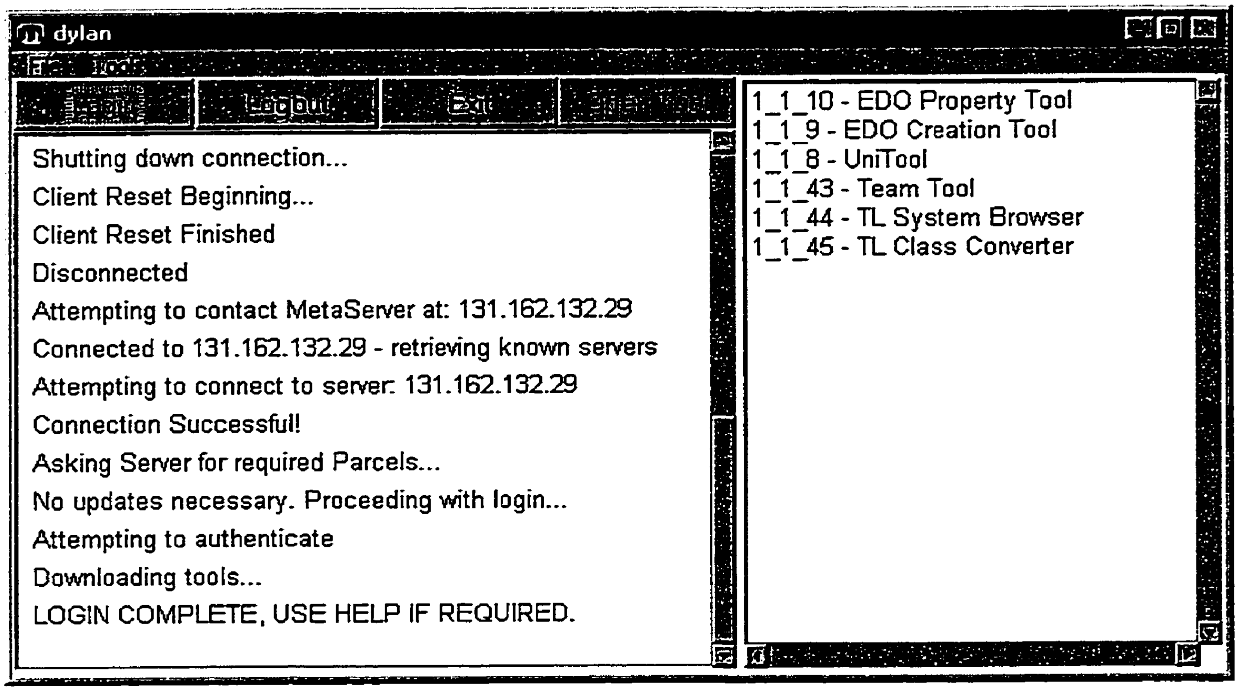


Figure 6.3 Class hierarchy diagram

6.1.1 TLDeveloper

TLDevelopers are the lowest level Team Lab agents and extensions of ordinary MUM agents. They understand more events than normal MUM agents and also have more tools. When a normal MUM agent connects to a MUM universe, he or she automatically gets a UniTool, an EDO Creation Tool and a Property Tool. In addition to these tools, a TLDeveloper also gets a TL Browser, a TL Class Converter and a TL Team Tool. TLDevelopers use these three tools to access the code repository and to obtain team information (Figure 6.4).



```
Shutting down connection...
Client Reset Beginning...
Client Reset Finished
Disconnected
Attempting to contact MetaServer at: 131.162.132.29
Connected to 131.162.132.29 - retrieving known servers
Attempting to connect to server: 131.162.132.29
Connection Successful!
Asking Server for required Parcels...
No updates necessary. Proceeding with login...
Attempting to authenticate
Downloading tools...
LOGIN COMPLETE, USE HELP IF REQUIRED.
```

```
1_1_10 - EDO Property Tool
1_1_9 - EDO Creation Tool
1_1_8 - UniTool
1_1_43 - Team Tool
1_1_44 - TL System Browser
1_1_45 - TL Class Converter
```

Figure 6.4 Launcher with tools available to Team Lab developers

6.1.2 TLAdministrator

TLAdministrator is the agent responsible for managing Team Lab environment, such as creating and removing teams, developers, etc. Usually there is only one instance of TLAdministrator in each universe, but it is possible to have more. TLAdministrator is a subclass of TLDeveloper, so it has all the power that a TLDeveloper has. Its additional functionality includes management in Team Lab.

6.2 Team Lab team management

Team management depends on Team Lab team (TLTeam) and team registry (TLTeamRegistry) (Figure 6.5). These two concepts are explained below:

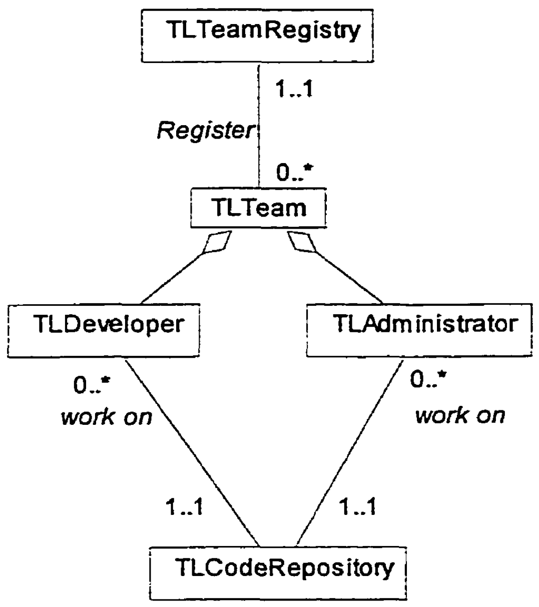


Figure 6.5 Team Lab Team

6.2.1 Team Lab Team

A team is a group of software developers working together on a project. Each team has one team leader who is also a developer and responsible for managing developers on the team. TLDevelopers can be assigned to be team leaders by TLAdministrator. Developers may be added to a team or removed at any time. Only the team leader and TLAdministrator have the power to make changes to teams. A developer may work in several different teams at the same time.

6.2.2 TLTeamRegistry

TLTeamRegistry is a registry of all Team Lab teams identified by their IDs. TLDevelopers can get information about available teams from the registry but they are not allowed to make any changes to it. TL Team Tool works together with TLTeamRegistry and TLTeam and thus provides facilities for managing teams.

6.3 Code Repository

The code repository is the central database of code objects. It resides on the server (Fig. 6.1) and holds all code objects created by developers and shared by them. Objects in the code repository are classified as category, class, protocol and method objects. The definitions of these objects are similar to the corresponding terms in Smalltalk. The following subsections explain the code repository and code objects in more detail.

6.3.1 Code Repository

The code repository is a place in Team Lab. Objects representing Smalltalk code written by TLDevelopers is stored here. The code repository looks like the Smalltalk system organization, which holds category information but it is not exactly the same. For example, in Visual Works Smalltalk some information is stored in arrays while in the code repository it is stored in OrderedCollections because the code repository class is a subclass of MUMPlace, which is an event-driven object and uses OrderedCollections to store its content information.

Smalltalk code in the code repository is represented by TLCategory objects and users can add, remove or modify them by using TL Browser. Because the repository is a subclass of MUMPlace, users can also "enter" the code repository and browse its contents using UniTool although this would be much less convenient than using TL Browser. Typically there is only one instance of code repository in a Team Lab and one Team Lab in a MUM universe, but it is possible to have more.

6.3.2 TLCategory

TLCategories are "places" in the code repository and represent Smalltalk categories of classes in the Team Lab environment. They may contain sub-places – TLClasses. TLCategories can be converted into Smalltalk categories and vice versa.

6.3.3 TLClass

TLClasses are "places" in TLCategories and represent Smalltalk classes. They may contain TLProtocols. TLClasses can be converted into Smalltalk classes and vice versa. Classes know their owners and versions and it is thus possible to have more than one class definition with a single class name and different versions. Only class owners can make changes to their classes unless the classes are public.

TLClasses use the following specialized concepts:

Public: Classes can be set to "public". If a class is public, everyone can make changes to it. To synchronize editing of public code, methods of a public class are protected by locks. If a method is edited by a user, it is locked and others cannot make changes to it. After the lock is removed (the user "accepts" the code), it is available to all users again.

Class Conversion: Smalltalk class definitions in Team Lab are EDOs in MUM and so they are not exactly the same as class definitions in the underlying VisualWorks

Smalltalk environment. In VisualWorks, Smalltalk class definitions also contain more information such as subclasses, while in TeamLab, only essential information is stored in TLClasses. However, classes can be converted between Team Lab and VisualWorks. If there are classes in a user image that do not exist in the code repository and the user wants to share them with other users, he or she can transfer them to the code repository. Users may also want to transfer classes written by others from the code repository to their own images for local testing or changes because if a class is not public other users are not allowed to change it directly in the code repository. After a user modifies a class, he or she can transfer it back to the code repository by giving it a different version - different versions of a class may be owned by different users.

There are two ways to convert classes (Figure 5.3). One is conversion between the server Smalltalk image and the code repository in MUM and the other is between the client image and the code repository. The former is performed by the load function and can be used for testing (details follow). The latter is used for code synchronization and offline programming.

<u>Load</u>: Classes in the code repository are objects in a MUM universe but they are not class objects in the server Smalltalk image. To load a TLClass from the code repository into the server Smalltalk image means to create its Smalltalk version and install it in the server Smalltalk image. A class may have several versions, but only one of its versions can be loaded into the server Smalltalk image at one time.

A class may also be unloaded. This removes the class from the server Smalltalk image, but leaves the corresponding TL object in the code repository.

6.3.4 TLProtocol

TLProtocols are "places" in TLClasses that represent Smalltalk method protocols in Team Lab environment. They can contain TLMethods. TLProtocols can be converted into Smalltalk protocols and vice versa.

6.3.5 TLMethod

TLMethods are "objects" in TLProtocols representing methods in Team Lab environment. They contain method code and can be converted into Smalltalk methods and vice versa.

6.4 Design – the Main Classes

This section introduces the main Team Lab classes. Since there are more than two hundred event classes that are very similar, most of them are not listed here.

6.4.1 TeamLab

Description: TeamLab is a subclass of MUMPlace and its instance is a place in MUM universe. It holds all Team Lab places and objects.

Superclass: MUMPlace

Instance Variables:

teamRegistry	<String>	ID of the team registry.

codeRepository	<String>	ID of the code repository.
teamLabTools	<OrderedCollection>	A collection of tools that are needed in Team Lab environment.

6.4.2 TLAdministrator

Description: TLAdministrator is an agent responsible for managing Team Lab environment, such as creating and deleting teams and developers.

Superclass: TLDeveloper

Instance Variables:

6.4.3 TLDeveloper

Description: Developers are agents who can work in Team Lab environment. They understand more events than normal MUM agents and have access to more tools.

Superclass: MUMPersonifiedAgent

Instance Variables:

teams	<OrderedCollection>	IDs of teams the developer is currently working in.

6.4.4 TLTeam

Description: TLTeam represents a team working in Team Lab environment. It knows all the developers in the team.

Superclass: EDO

Instance Variables:

teamLeader	<String>	ID of the team leader.

developers	<OrderedCollection>	IDs of developers currently in this team.

6.4.5 TLTeamInfoDescriptor

Description: An instance of this class provides basic information about a Team Lab team and is used to transfer team information between clients and servers.

Superclass: Object

Instance Variables:

teamLeader	<String>	ID of team leader.
members	<OrderedCollection>	IDs of team members.

6.4.6 TLTeamRegistry

Description: Instances of this class hold IDs of all teams in a MUM universe.

Superclass: MUMPlace

Instance Variables:

teams	<OrderedCollection>	IDs of Team Lab teams in this universe.

6.4.7 TLCategory

Description: Instances of this class represent categories in Team Lab. They can be converted into Smalltalk categories.

Superclass: MUMPlace

Instance Variables:

classes	<OrderedCollection>	IDs of classes in the category.

loaded	<Boolean>	True if the category is loaded into the server Smalltalk image.

6.4.8 TLClass

Description: Instances of this class represent Team Lab classes. They can be converted into Smalltalk classes.

Superclass: MUMPlace

Instance Variables:

version	<String>	Version of the class.
classOwner	<String>	ID of the class owner.
instanceVars	<String>	String representation of instance variables.
classVars	<String>	String representation of class variables.
pools	<String>	String representation of pools.
classInstanceVars	<String>	String representation of the class instance variables.
protocols	<OrderedCollection>	IDs of the class' protocols.
category	<String>	Name of the class' category.
superClassName	<Symbol>	The name of its super class.
comment	<String>	Comment for this class.
loaded	<Boolean>	True if the class is loaded into the server Smalltalk image.
released	<Boolean>	True if the class is released.
current	<Boolean>	True if the class is the current class.

Public	<Boolean>	True if the class is available to public.

6.4.9 TLClassDefinition

Description: Instances of this class provide basic information about Team Lab classes. They are used to transfer information about classes between servers and clients. This class and its subclass TLClassDetail work as class proxies but they are used in different situations. TLClassDefinition is used by TLBrowser whereas TLClassDetail is used by TLClassConverter. TLClassDefinition carries less information and thus reduces network load.

Superclass: Object

Instance Variables:

superClass	<Symbol>	Super class name.
name	<String>	The name of the class.
instanceVars	<String>	String representation of instance variables.
classVars	<String>	String representation of class variables.
pools	<String>	String representation of pools.
category	<String>	Category name.
classInstanceVars	<String>	String representation of class instance variables.
comment	<String>	Comment for this class.

6.4.10 TLClassDetail

Description: Instances of this class provide all information about Team Lab classes and corresponding metaclasses. They are used to transfer information about classes between servers and clients. Team Lab class or Smalltalk class can be created by using this information. In most cases such as getting a list of classes in category, only its super class is used in order to reduce network load.

Superclass: TLClassDefinition

Instance Variables:

classProtocols	<OrderedCollection>	Class protocols.
instanceProtocols	<OrderedCollection>	Instance protocols.
classMethods	<OrderedCollection>	Class methods.
instanceMethods	<OrderedCollection>	Instance methods.
version	<String>	Version number of the class.

6.4.11 TLCodeRepository

Description: An instance of this class holds all the code objects in Team Lab. It is an EDO and contains other Team Lab objects. There is only one instance of TLCodeRepository in each Team Lab.

Superclass: MUMPlace

Instance Variables:

categories	<OrderedCollection>	IDs of categories in Team Lab environment.

6.4.12 TLProtocol

Description: Instances of this class represent class and instances protocols in Team Lab environment. They can be converted into Smalltalk protocols.

Superclass: MUMPlace

Instance Variables:

methods	<OrderedCollection>	IDs of methods in this protocol.
className	<Symbol>	Class name to which the protocol belongs.
meta	<Boolean>	True if the protocol is a class protocol.

6.4.13 TLMethod

Description: Instances of this class are EDOs representing methods in Team Lab environment. They can be converted into Smalltalk methods.

Superclass: EDO

Instance Variables:

text	<Text>	Text representation of the source code for this method.
className	<Symbol>	The name of the class to which the method belongs.
protocol	<String>	The protocol name in which the method is.
meta	<Boolean>	True if it is a class method.
CurrentUser	<String>	ID of the user who is editing the method.

6.4.14 TLMethodDefinition

Description: Instances of this class provide information about Team Lab methods. They are used to transfer method information between servers and clients. Team Lab methods or Smalltalk methods can be created from the information.

Superclass: Object

Instance Variables:

name	<String>	Name of the method.

text	<Text>	Text representation of the code for the method.
protocol	<String>	The protocol name in which the method is.
className	<Symbol>	Name of the class to which the method belongs.

6.4.15 TLCompiler

Description: Instances of this class are used to compile Team Lab code and check syntax errors of class definitions and methods.

Superclass: Compiler

Instance Variables:

6.4.16 TLBrowser

Description: TLBrowser is a code browser for the Team Lab code repository. It has the same basic functions as a Smalltalk System Browser and additional functions for Team Lab class management (Figure 7.2).

Superclass: MUMTool

Instance Variables:

category	<String>	Selected category.
className	<Symbol>	Selected class.
meta	<Boolean>	False for viewing instance methods, true for class methods.
protocol	<Symbol>	Selected protocol.
method	<Symbol>	Selected method.

textMode	<Symbol>	Indicating the nature of the currently viewed text such as class definition, class comment or method definition.
categoryList	<ValueHolder>	Category list value holder.
classList	<ValueHolder>	Class list value holder.
protocolList	<ValueHolder>	Protocol list value holder.
methodList	<ValueHolder>	Method list value holder.
textValue	<ValueHolder>	Text value holder.
metaHolder	<ValueHolder>	Value holder of the meta instance variable.
infoHolder	<ValueHolder>	Value holder the feedback information about an operation.
categories	<OrderedCollection>	Categories in the code repository.
classes	<OrderedCollection>	Classes in the category.
protocols	<OrderedCollection>	Protocols of the currently selected class.
methods	<OrderedCollection>	Methods in the selected protocol.
currentHolder	<ValueHolder>	Value holder of the "current" state.
loadedHolder	<ValueHolder>	Value holder of the "loaded" state.
releasedHolder	<ValueHolder>	Value holder of the "release" state.
allHolder	<ValueHolder>	Value holder of the "all" state.
publicHolder	<ValueHolder>	Value holder of the "public" state.

6.4.17 TLBrowserBase

Description: TLBrowserBase is a tool base that works with TLBrowser. In MUM, a tool base is a part of a UI on the client side and is responsible for translating messages to events or events to messages.

Superclass: MUMToolBase

Instance Variables:

6.4.18 TLClassConverter

Description: TLClassConverter is the UI tool responsible for transferring classes between the client image and the code repository (Figure 7.3).

Superclass: MUMTool

Instance Variables:

infoHolder	<ValueHolder>	Value holder of the feedback information about the operation being performed.
TLCategoryList	<ValueHolder>	TL category value holder.
localClassList	<OrderedCollection>	Local class value holder.
localCategory	<ValueHolder>	Local category value holder.
TLCategory	<ValueHolder>	TL category value holder.
TLClassList	<ValueHolder>	TL class value holder.
localCategoryList	<ValueHolder>	Local category value holder.
categories	<OrderedCollection>	Categories in the code repository.
classes	<OrderedCollection>	Classes in the selected category.
updating	<Boolean>	True if information is being updated.

6.4.19 TLClassConverterBase

Description: TLClassConverterBase is a tool base that works together with TLClassConverter. It is responsible for translating messages to events and events to messages for use in MUM environment.

Superclass: MUMToolBase

Instance Variables:

6.4.20 TLTeamTool

Description: TLTeamTool is the UI tool for managing teams. It can be used to create, modify and remove Team Lab teams (Figure 7.1).

Superclass: MUMTool

Instance Variables:

nonmemberHolder	<ValueHolder>	Non-member value holder.
memberHolder	< ValueHolder >	Team member value holder.
infoHolder	< ValueHolder >	Value holder of the feedback information of operations.
selectedTeam	<String>	Selected team.
teamHolder	< ValueHolder >	TL team value holder.
teams	<OrderedCollection>	All teams in the universe.
members	<OrderedCollection>	Member proxies of the team.
nonmembers	<OrderedCollection>	Non-member proxies.
developers	<OrderedCollection>	Proxies of all available developers.

nameHolder	<String>	Name value holder for creating new teams.
leaderHolder	<String>	ID of the team leader.

6.4.21 TLTeamToolBase

Description: TLTeamToolBase is a tool base that works together with TLTeamTool. It is responsible for translating messages to events or events to messages for use in MUM environment.

Superclass: MUMToolBase

Instance Variables:

6.4.22 AgentAddCategoryRequestEvent

Description: Instances of this class are sent by TLDevelopers to TLCodeRepository to request it to add a new category.

Superclass: RequestEvent

Instance Variables:

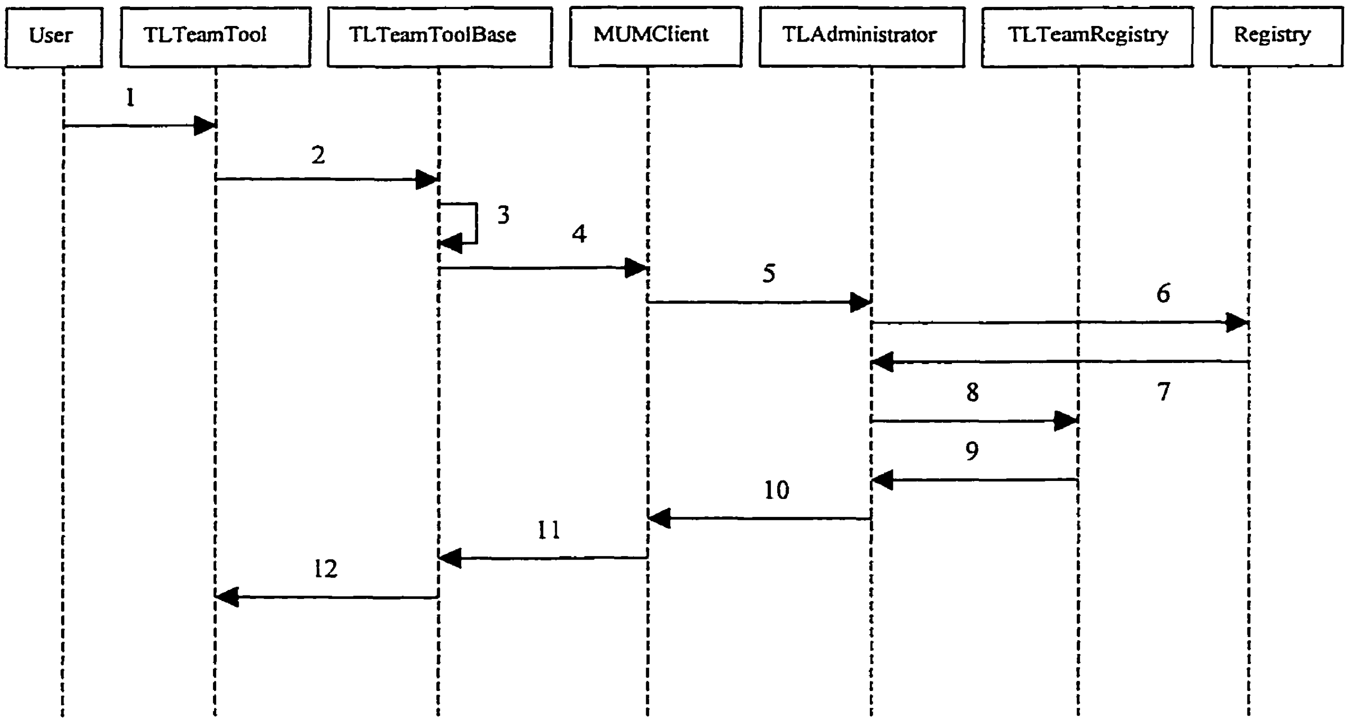
category	<String>	Name of the category to be added.
agent	<String>	ID of the agent who send this event.

Note: Team Lab contains definitions of many similar event classes.

6.5 Sequence Diagram

This section contains a series of UML sequence diagrams illustrating Team Lab's operation in selected use cases.

6.5.1 Create Team



1: User clicks on "Create" button

2: addTeam message

3: SelfAddTeamEvent

4: ToolAddTeamRequestEvent

5: ClientAddTeamRequestEvent

6: AgentEDOCreationRequestEvent

7: ConfirmationEvent

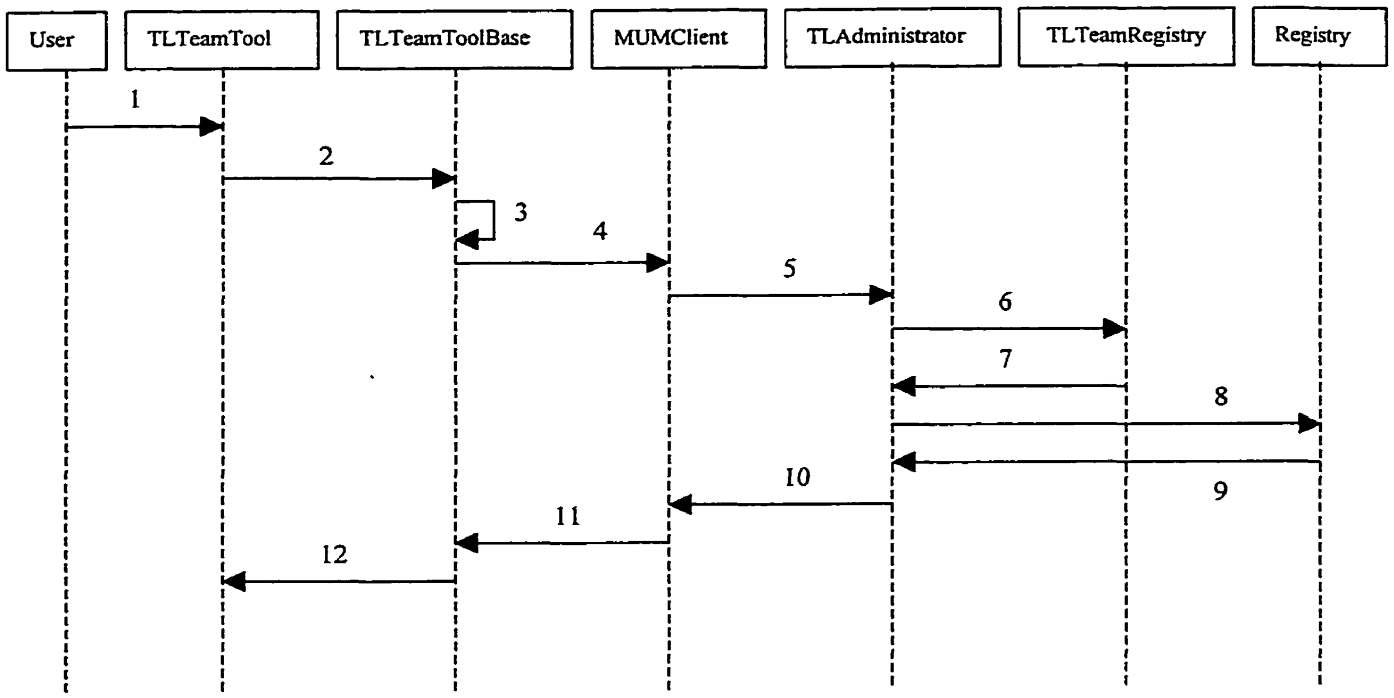
8: AgentAddTeamRequestEvent

9, 10, 11: TLConfirmationEvent

12: confirmation: message

Figure 6.6 Create a team

6.5.2 Remove Team



1: User clicks on “Remove” button

2: removeTeam message

3: SelfRemoveTeamEvent

4: ToolRemoveTeamRequestEvent

5: ClientRemoveTeamRequestEvent

6: AgentRemoveTeamRequestEvent

7: TLConfirmationEvent

8: AgentEDODeletionRequestEvent

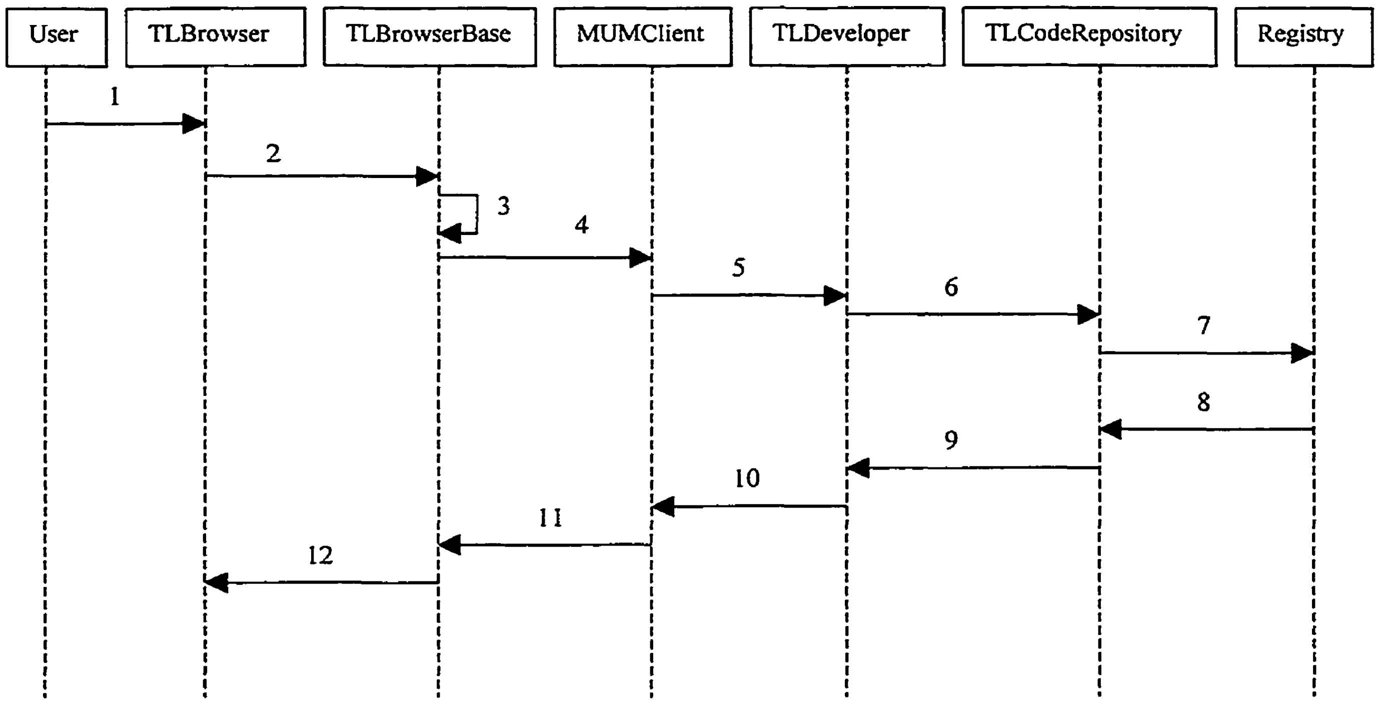
9: ConfirmationEvent

10, 11: TLConfirmationEvent

12: confirmation: message

Figure 6.7 Remove a team

6.5.3 Add Category



1: User selects "add" menu item

2: addTLCategory message

3: SelfAddTLCategoryEvent

4: ToolAddCategoryRequestEvent

5: ClientAddCategoryRequestEvent

6: AgentAddCategoryRequestEvent

7: EDOCreationRequestEvent

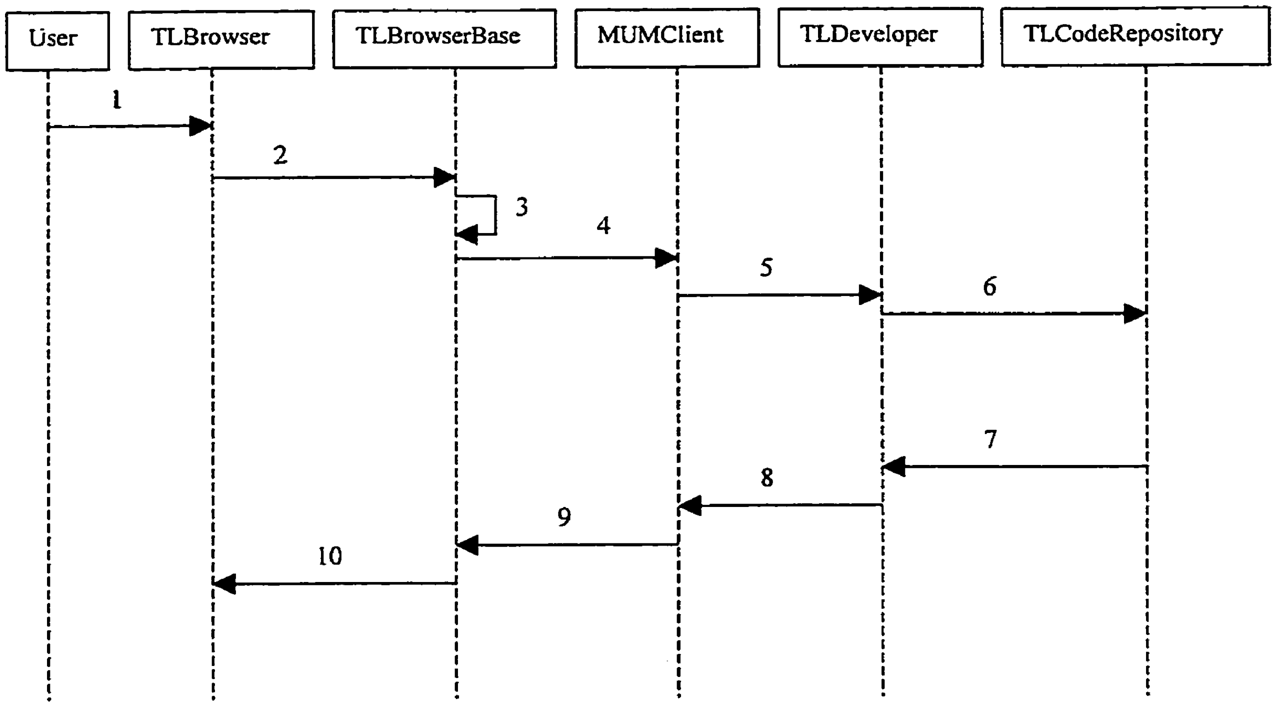
8: ConfirmationEvent

9, 10, 11: TLConfirmationEvent

12: confirmation: message

Figure 6.8 Add a category

6.5.4 Remove Category



1: User selects "remove" menu item

2: removeTLCategory message

3: SelfRemoveTLCategoryEvent

4: ToolRemoveCategoryRequestEvent

5: ClientRemoveCategoryRequestEvent

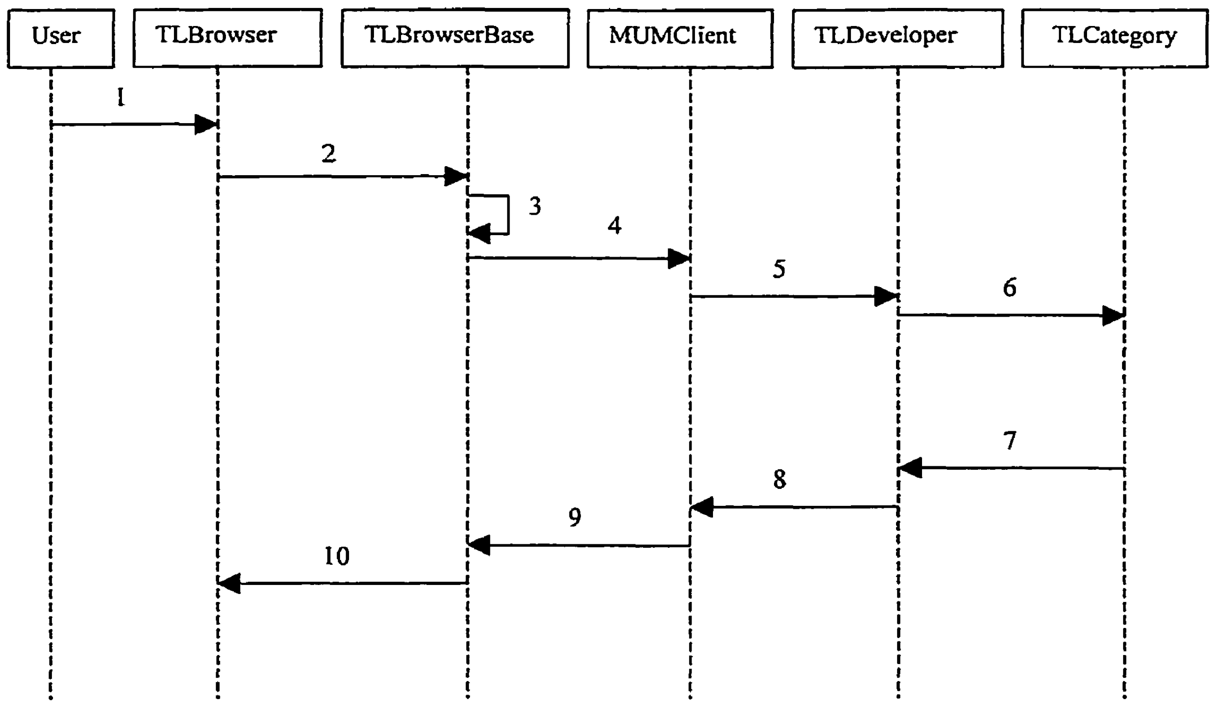
6: AgentRemoveCategoryRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.9 Remove a category

6.5.5 Rename Category



1: User selects "rename" menu item

2: renameTLCategory message

3: SelfRenameCategoryEvent

4: ToolRenameCategoryRequestEvent

5: ClientRenameCategoryRequestEvent

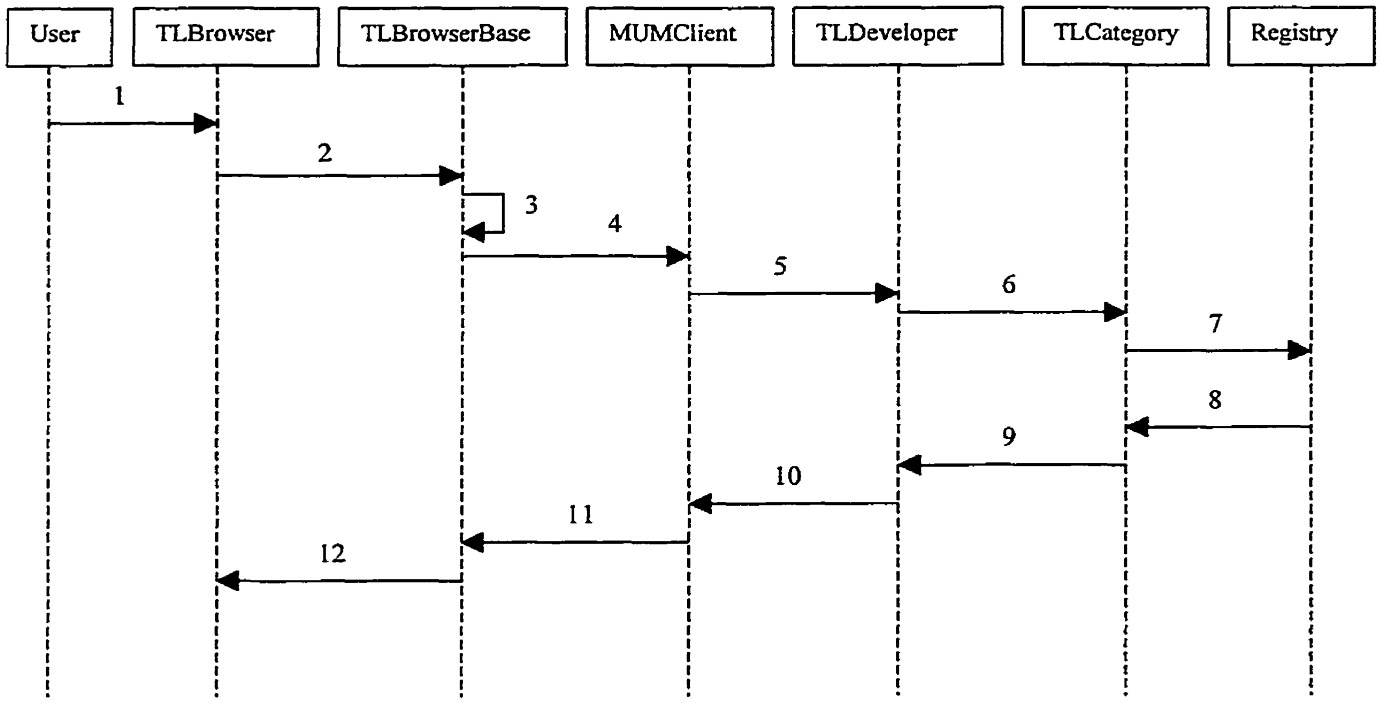
6: AgentRenameCategoryRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.10 Rename a category

6.5.6 Define Class



1: User selects “accept” menu item

2: CreateNewClass

3: SelfAddClassEvent

4: ToolAddClassRequestEvent

5: ClientAddClassRequestEvent

6: AgentAddClassRequestEvent

7: EDOCreationRequestEvent

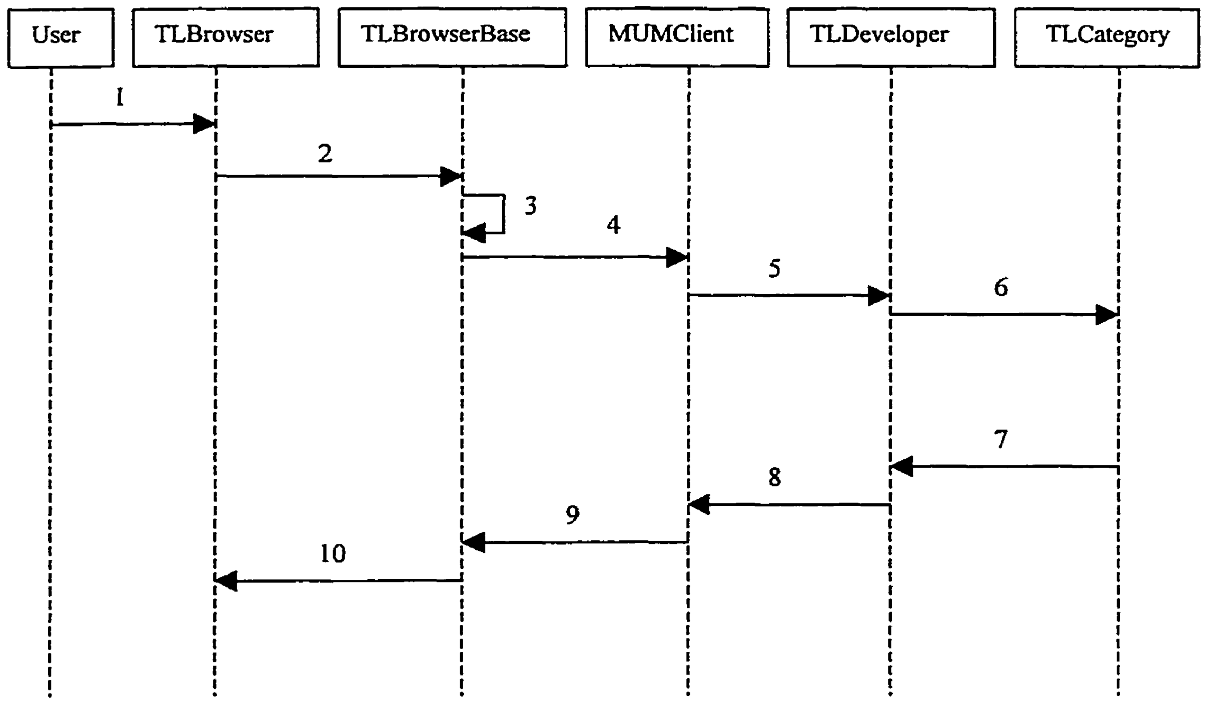
8: ConfirmationEvent

9, 10, 11: TLConfirmationEvent

12: confirmation: message

Figure 6.11 Define a class

6.5.7 Remove Class



1: User selects "remove" menu item

2: removeClass message

3: SelfRemoveClassEvent

4: ToolRemoveClassRequestEvent

5: ClientRemoveClassRequestEvent

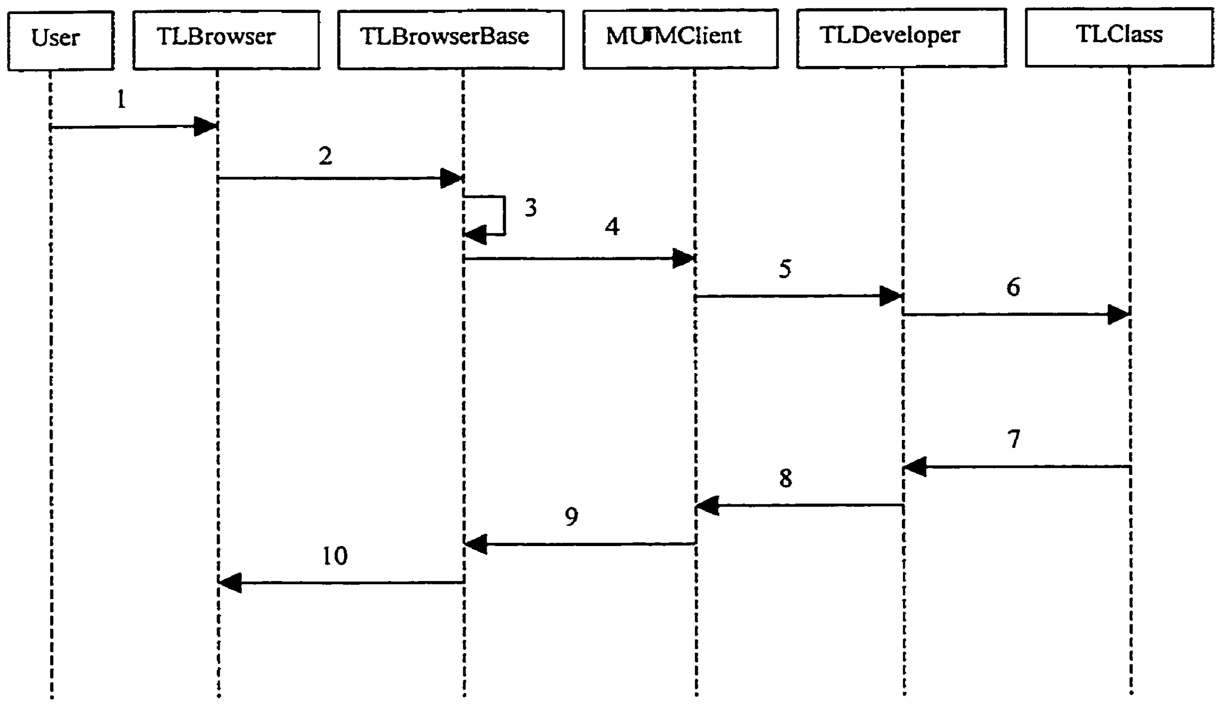
6: AgentRemoveClassRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.12 Remove a class

6.5.8 Load Class



1: User selects "load" menu item

2: loadClass message

3: SelfLoadClassEvent

4: ToolLoadClassRequestEvent

5: ClientLoadClassRequestEvent

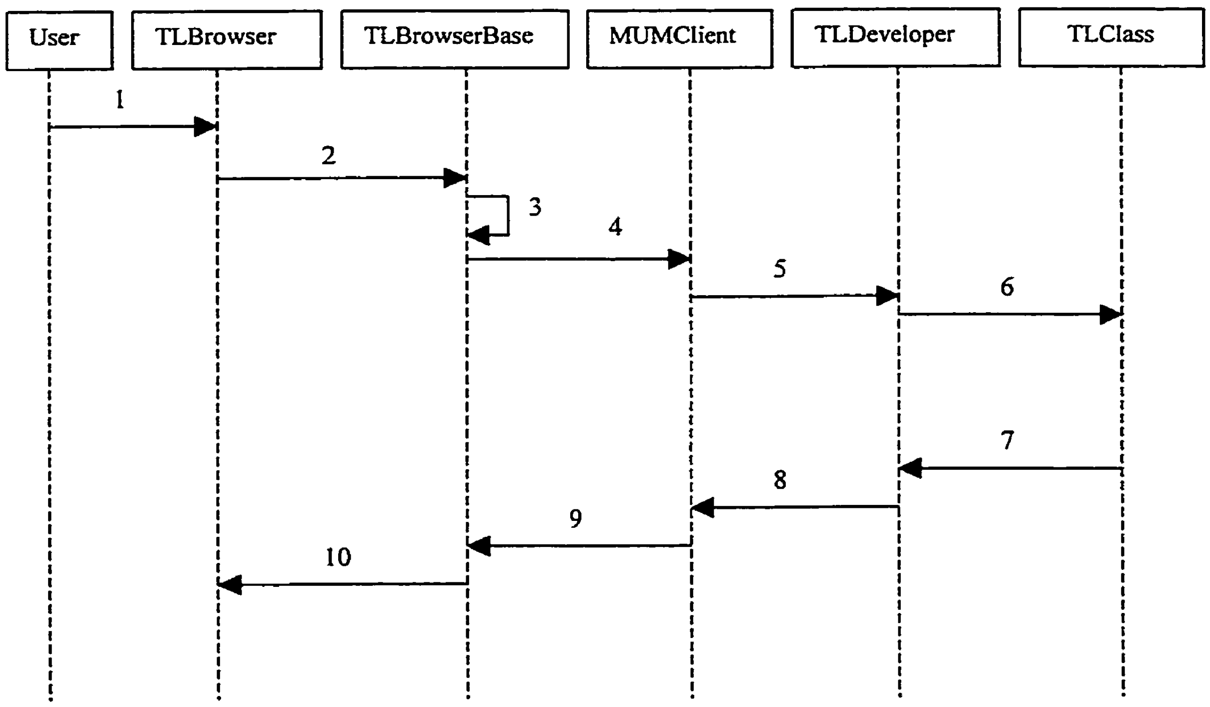
6: AgentLoadClassRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.13 Load a class

6.5.9 Unload Class



1: User selects “unload” menu item

2: unloadClass message

3: SelfUnloadClassEvent

4: ToolUnloadClassRequestEvent

5: ClientUnloadClassRequestEvent

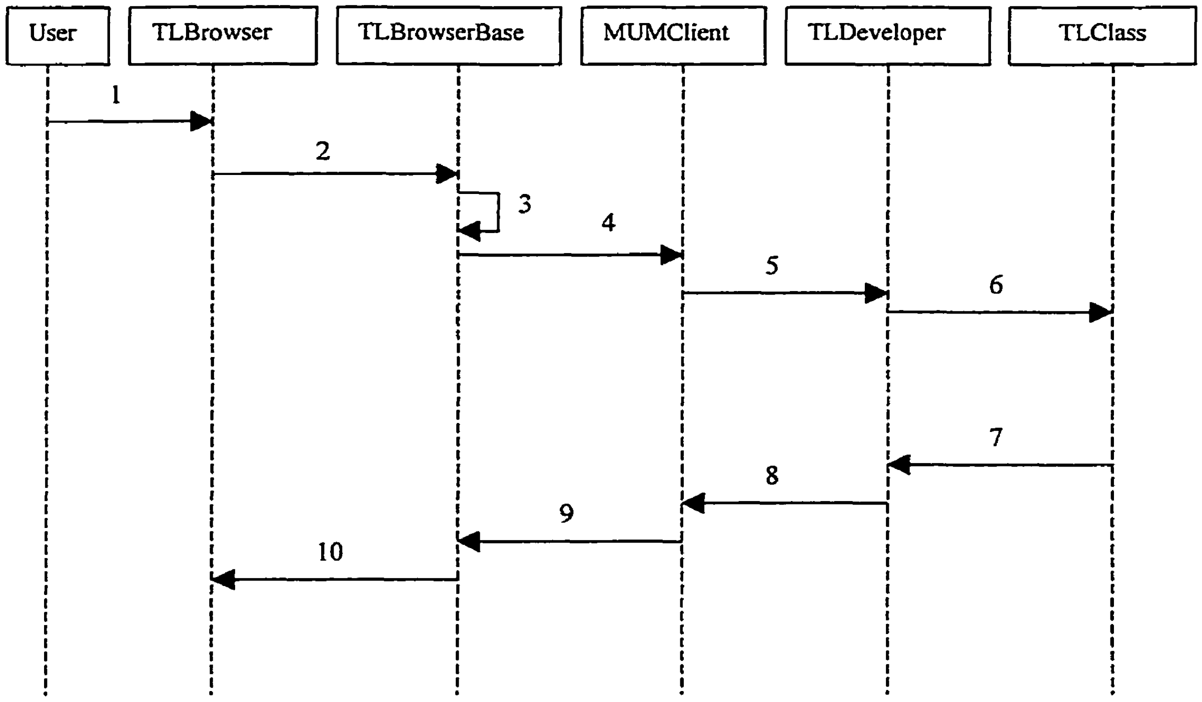
6: AgentUnloadClassRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.14 Unload a class

6.5.10 Release Class (similar Set Public, Set Private, Set Current)



1: Users selects "release" (or "set public", "set private", "set current") menu item

2: setClassState message

3: SelfSetClassStateEvent

4: ToolSetClassStateRequestEvent

5: ClientSetClassStateRequestEvent

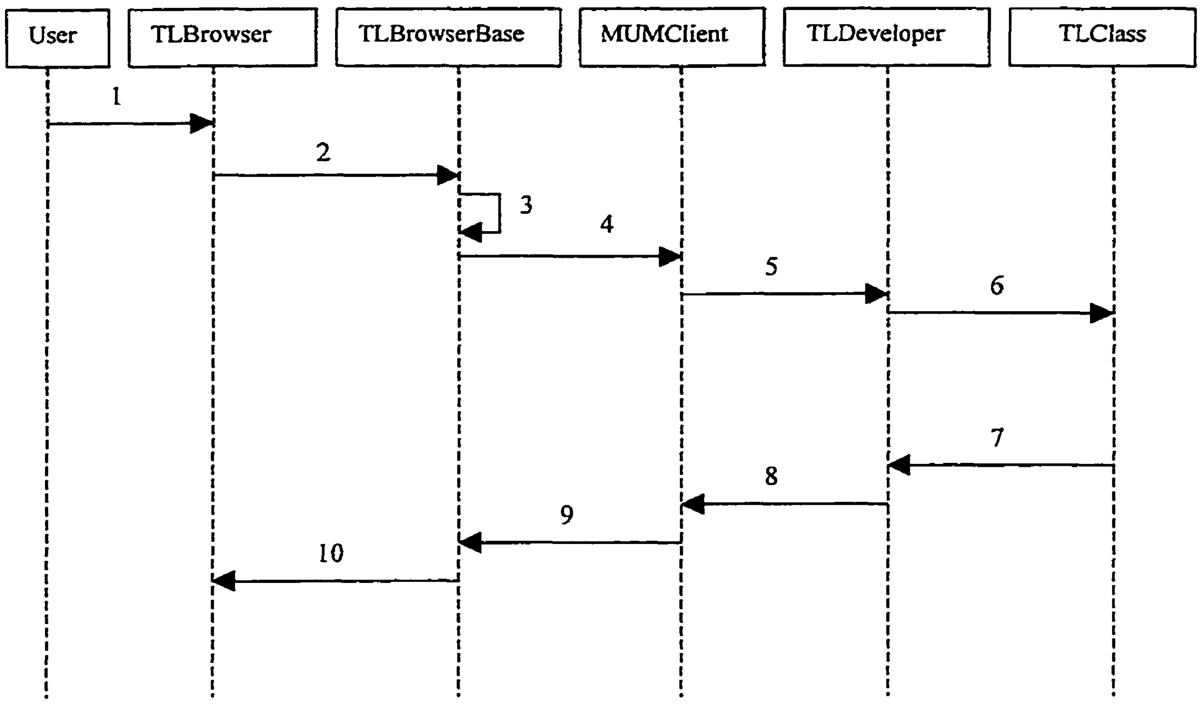
6: AgentSetClassStateRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.15 Release a class (set public, set private, set current)

6.5.11 Set Class Version



1: User select "set version" menu item

2: setVersion message

3: SelfSetVersionEvent

4: ToolSetVersionRequestEvent

5: ClientSetVersionRequestEvent

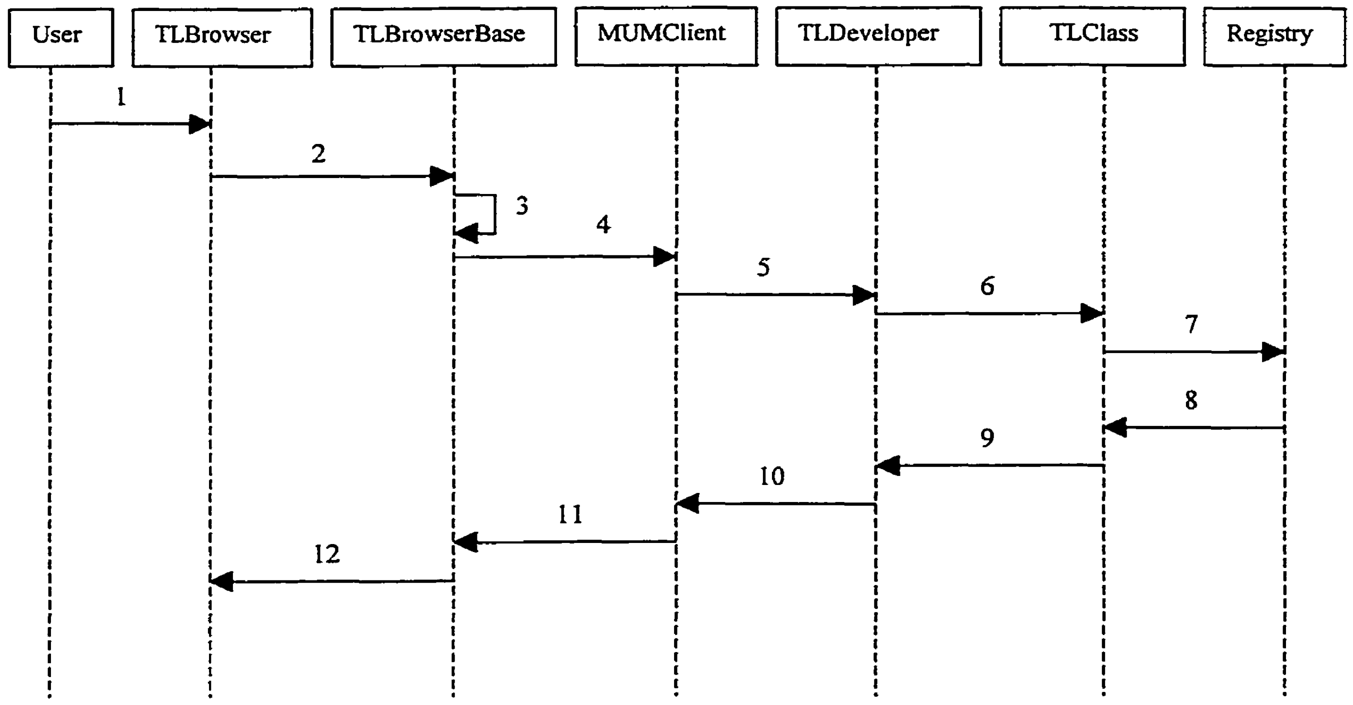
6: AgentSetVersionRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.16 Set class version

6.5.12 Add Protocol



1: User selects "add" menu item

2: addTLProtocol message

3: SelfAddProtocolEvent

4: ToolAddProtocolRequestEvent

5: ClientAddProtocolRequestEvent

6: AgentAddProtocolRequestEvent

7: EDOCreationRequestEvent

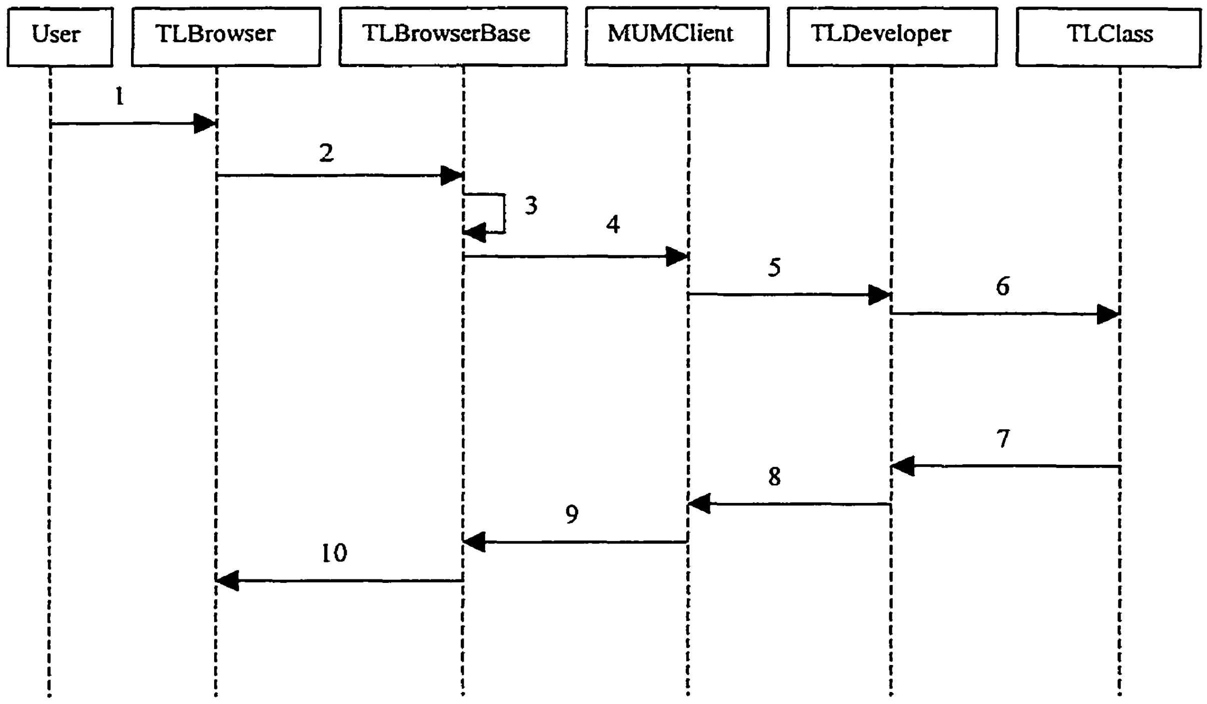
8: ConfirmationEvent

9, 10, 11: TLConfirmationEvent

12: confirmation: message

Figure 6.17 Add a protocol

6.5.13 Remove Protocol



1: User selects “remove” menu item

2: RemoveProtocol

3: SelfRemoveProtocolEvent

4: ToolRemoveProtocolRequestEvent

5: ClientRemoveProtocolRequestEvent

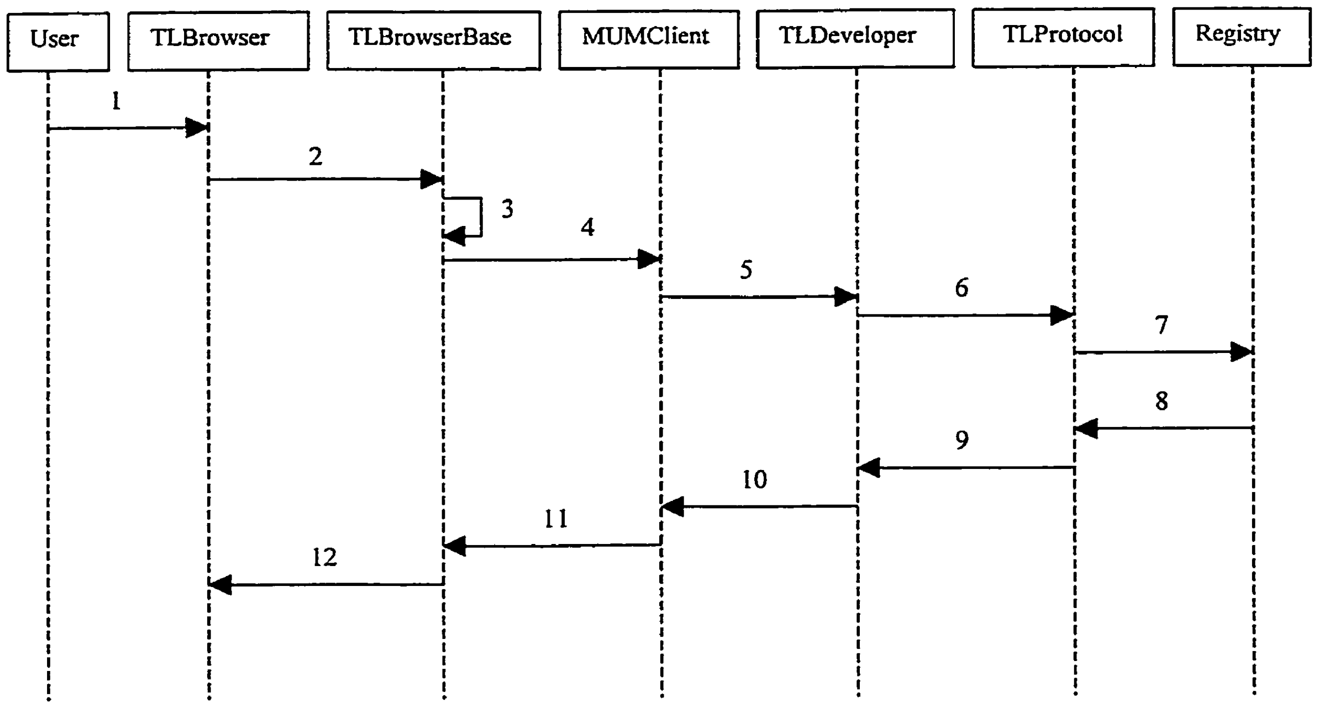
6: AgentRemoveProtocolRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.18 Remove a protocol

6.5.14 Accept Method



1: User selects "accept" menu item

2: addTLMethod message

3: SelfAddMethodEvent

4: ToolAddMethodRequestEvent

5: ClientAddMethodRequestEvent

6: AgentAddMethodRequestEvent

7: EDOCreationRequestEvent

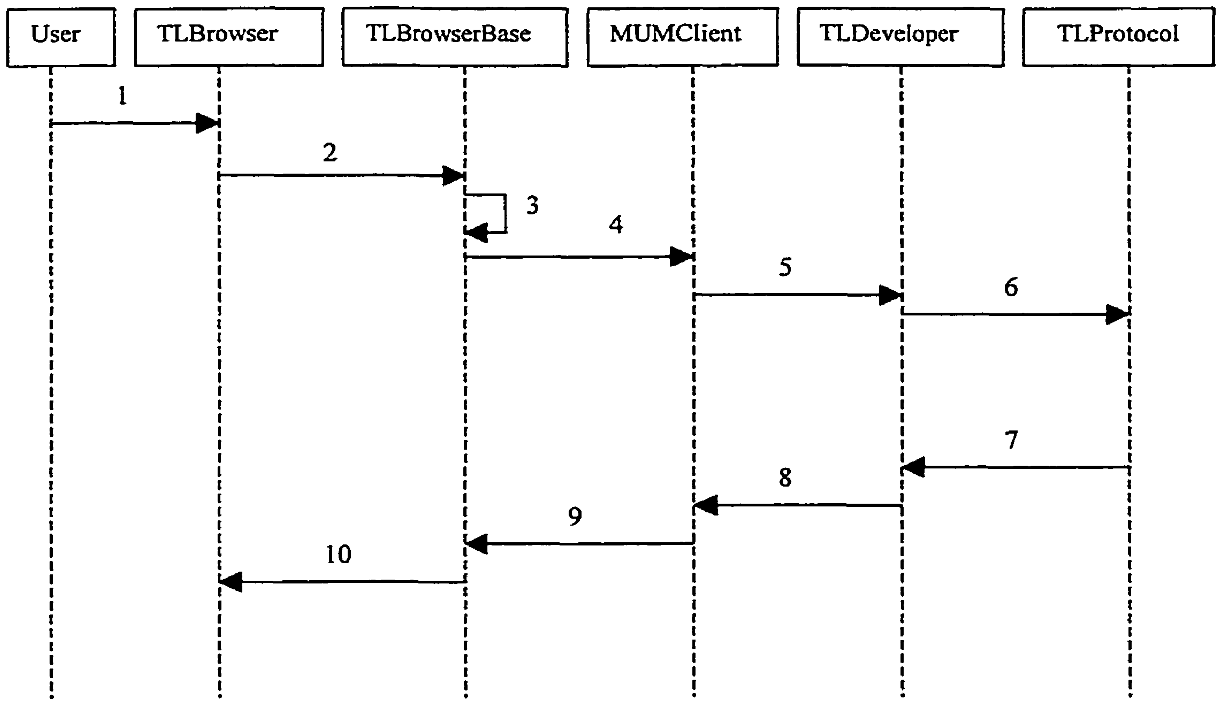
8: ConfirmationEvent

9, 10, 11: TLConfirmationEvent

12: confirmation: message

Figure 6.19 Define a method

6.5.15 Remove Method



1: User selects “remove” menu item

2: removeMethod message

3: SelfRemoveMethodEvent

4: ToolRemoveMethodRequestEvent

5: ClientRemoveMethodRequestEvent

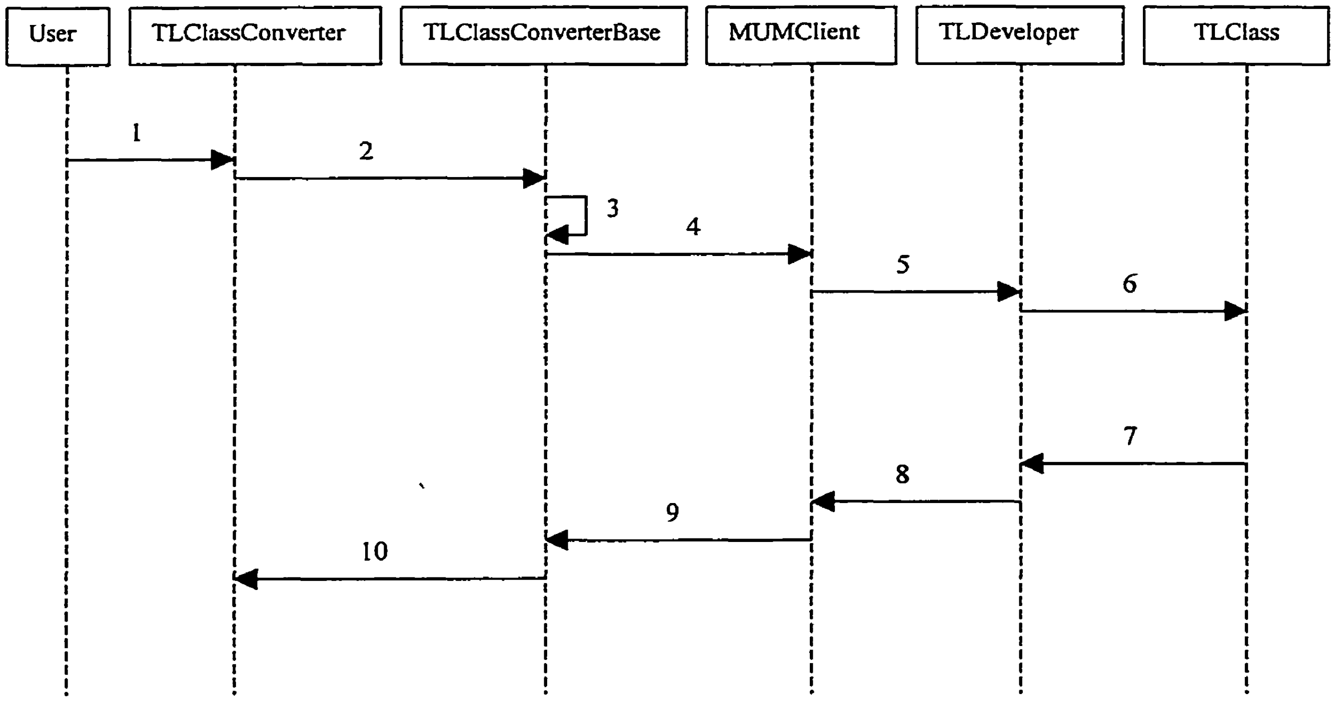
6: AgentRemoveMethodRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.20 Remove a method

6.5.16 Transfer Class from TL Code Repository to Local Image



1: User clicks on "<<" button

2: getTLClass message

3: SelfClassDetailEvent

4: ToolClassDetailRequestEvent

5: ClientClassDetailRequestEvent

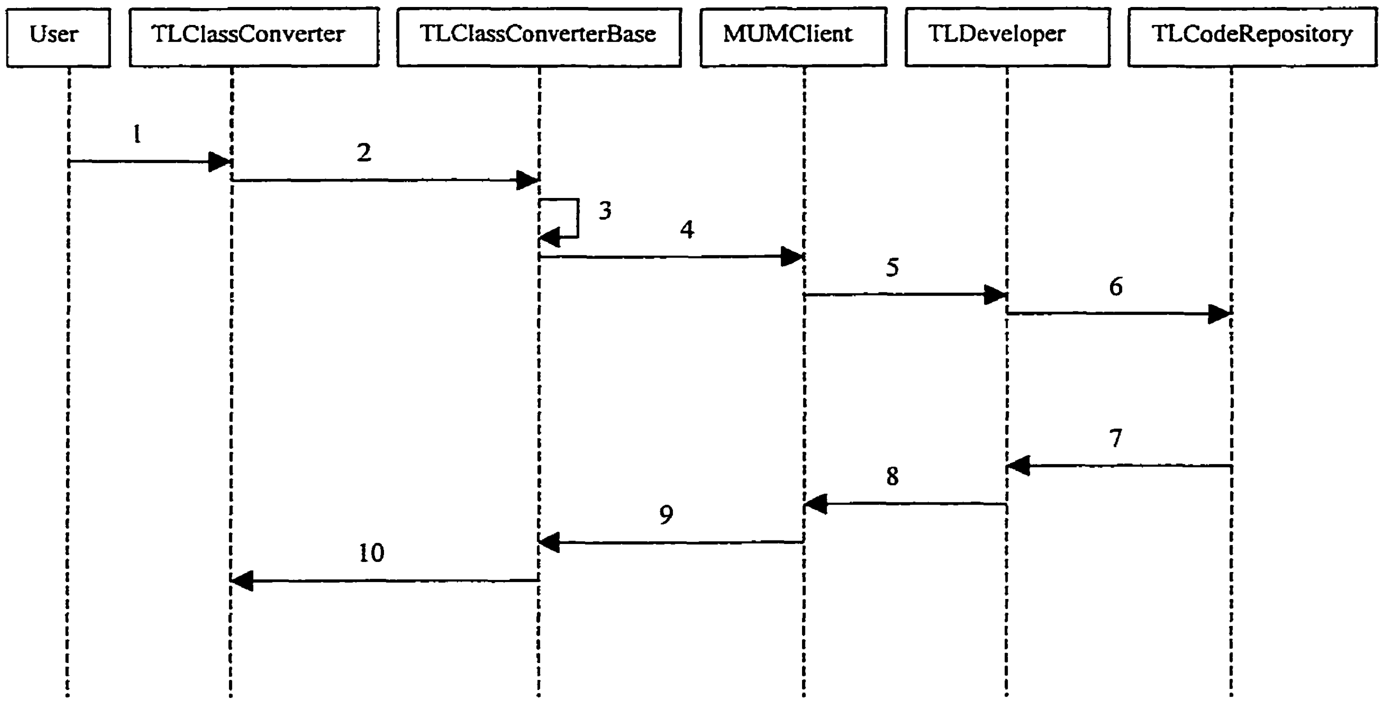
6: AgentClassDetailRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

Figure 6.21 Transfer a class from the code repository to the local image

6.5.17 Transfer Class from Local Image to TL Code Repository



1: User clicks on “>>” button

2: transferClass message

3: SelfTransferClassEvent

4: ToolTransferClassRequestEvent

5: ClientTransferClassRequestEvent

6: AgentTransferClassRequestEvent

7, 8, 9: TLConfirmationEvent

10: confirmation: message

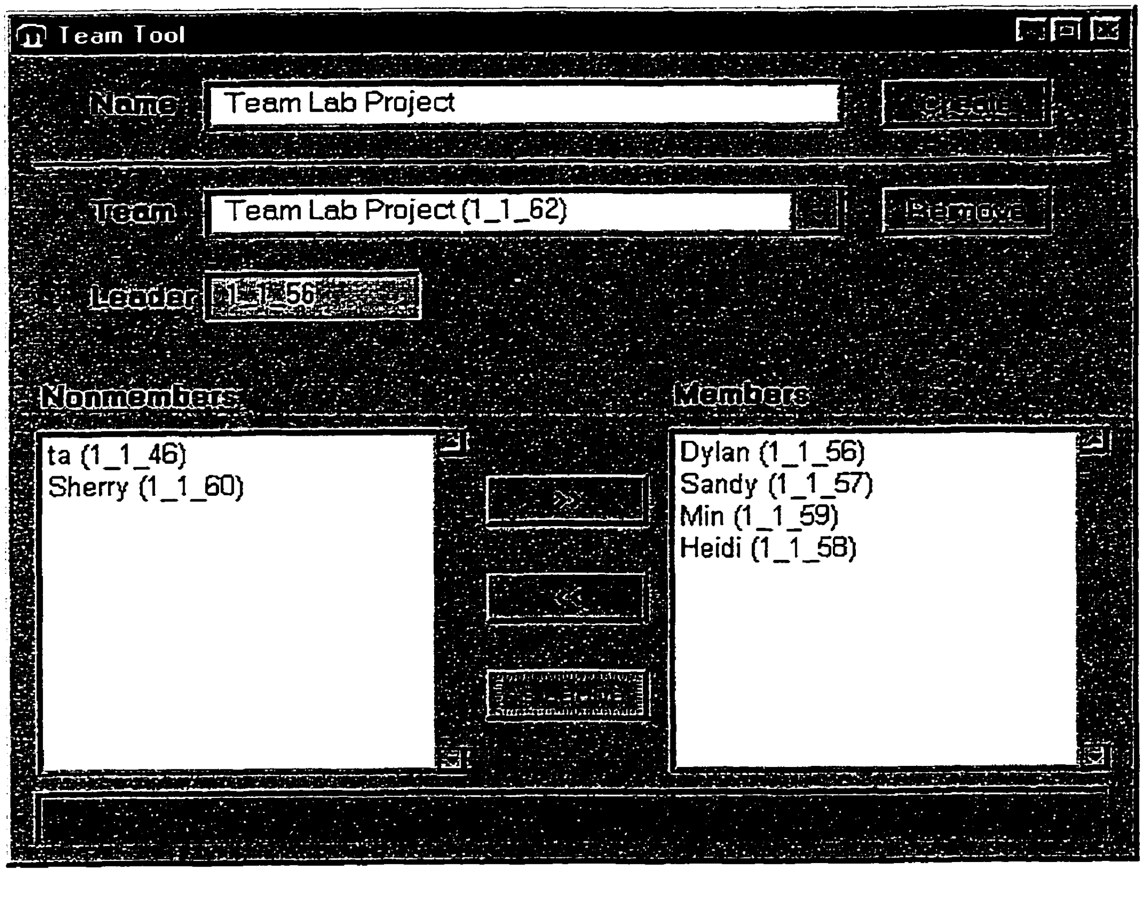
Figure 6.22 Transfer a class from the local image to the code repository

Chapter 7 Functionality and Implementation

This chapter describes the uses and functionality of the three Team Lab client side tools and some implementation details.

7.1 Team Tool

The Team tool (Figure 7.1) is used for managing teams. It is used mainly by the administrator and team leaders and its basic functions are listed below:



Name: Team Lab Project

Team: Team Lab Project (1_1_62)

Leader: 1_1_56

Nonmembers:
ta (1_1_46)
Sherry (1_1_60)

Members:
Dylan (1_1_56)
Sandy (1_1_57)
Min (1_1_59)
Heidi (1_1_58)

Figure 7.1 Team tool

Create a team: To create a team, enter the team name in the "Name" text box on top of the window and press the "Create" button. Since MUM objects use IDs, it is possible to use the same name for multiple teams. Only TLAdministrator can perform this action.

Remove a team: From the "Team" combo box, select the team to be deleted and press the "Remove" button. Only TLAdministrator can perform this action.

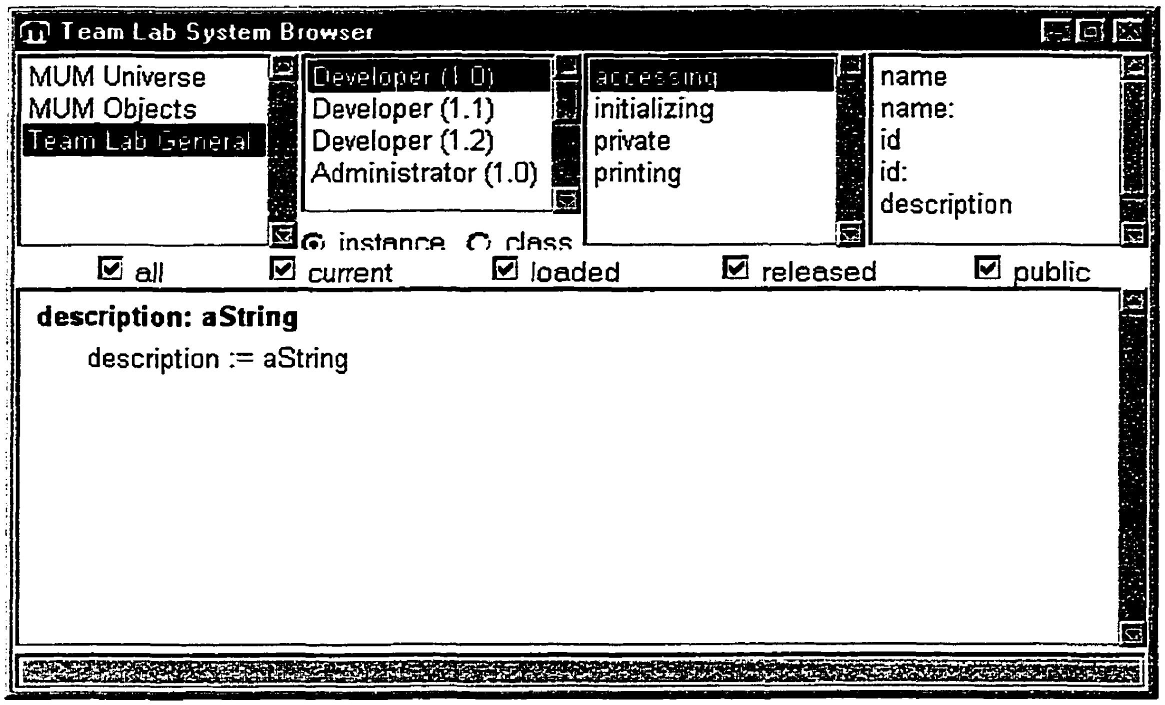
Add a team member: Only TLDevelopers and TLAdministrator can be added to a team. After selecting the desired team from the "Team" combo box, all available developers are listed in the "Members" and "Nonmembers" list boxes respectively. Available developers include all instances of TLDevelopers in the universe. Select a developer from the "Nonmembers" list and press the ">>" button and the developer will be added to the team. Only TLAdministrator and the leader of the team can perform this action.

Remove a team member: After selecting the desired team in the team combo box, all available developers are listed in the "Members" and "Nonmembers" list boxes respectively. Select a developer from the "Members" list and press "<<" button. The developer will be removed from the team. Only TLAdministrator and the leader of the team can perform this action. This operation only removes the developer from the team but if a user wants to remove an existing developer from the "Nonmember" list (from Team Lab environment), he or she must use MUM UniTool to do it.

Assign team leader: After selecting the desired team in the "Team" combo box, all available developers are listed in the "Members" and "Nonmembers" list boxes respectively. Select a developer from the "Members" list and press "As Leader" button. The developer will become the leader of the team. Team leader's ID is displayed in the "Leader" text box. Only TLAdministrator and the leader of the team can perform this action. If the team has no leader assigned yet, only TLAdministrator can perform this action.

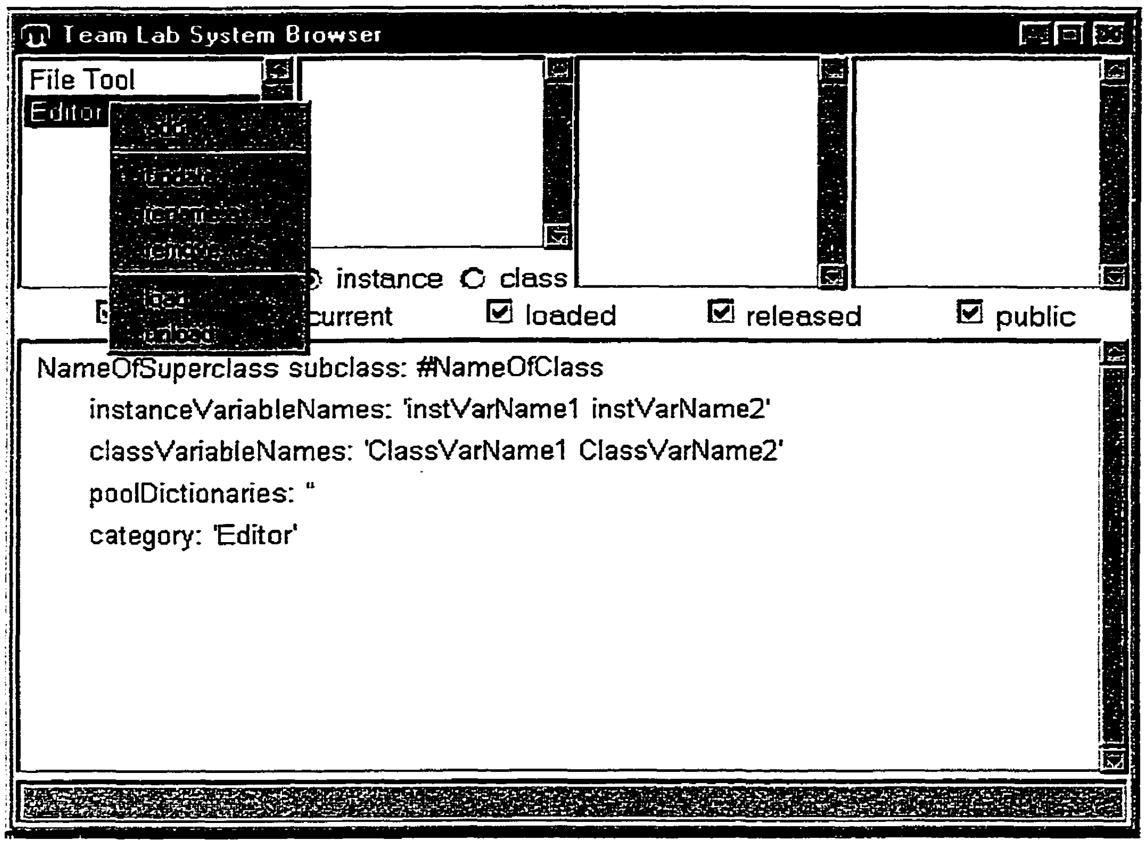
7.2 TL Browser

TL Browser (Figure 7.2) is very similar to the standard Smalltalk System Browser and has the same basic functions. The main differences are that it is used remotely to browse the code in the code repository located in the server and that it provides some additional features such as class level version control and security.



```
description: aString
    description := aString
```

Figure 7.2 TL Browser



☐ current ☑ loaded ☑ released ☑ public

```
NameOfSuperclass subclass: #NameOfClass
    instanceVariableNames: 'instVarName1 instVarName2'
    classVariableNames: 'ClassVarName1 ClassVarName2'
    poolDictionaries: ''
    category: 'Editor'
```

Figure 7.3 Category menu of TL Browser

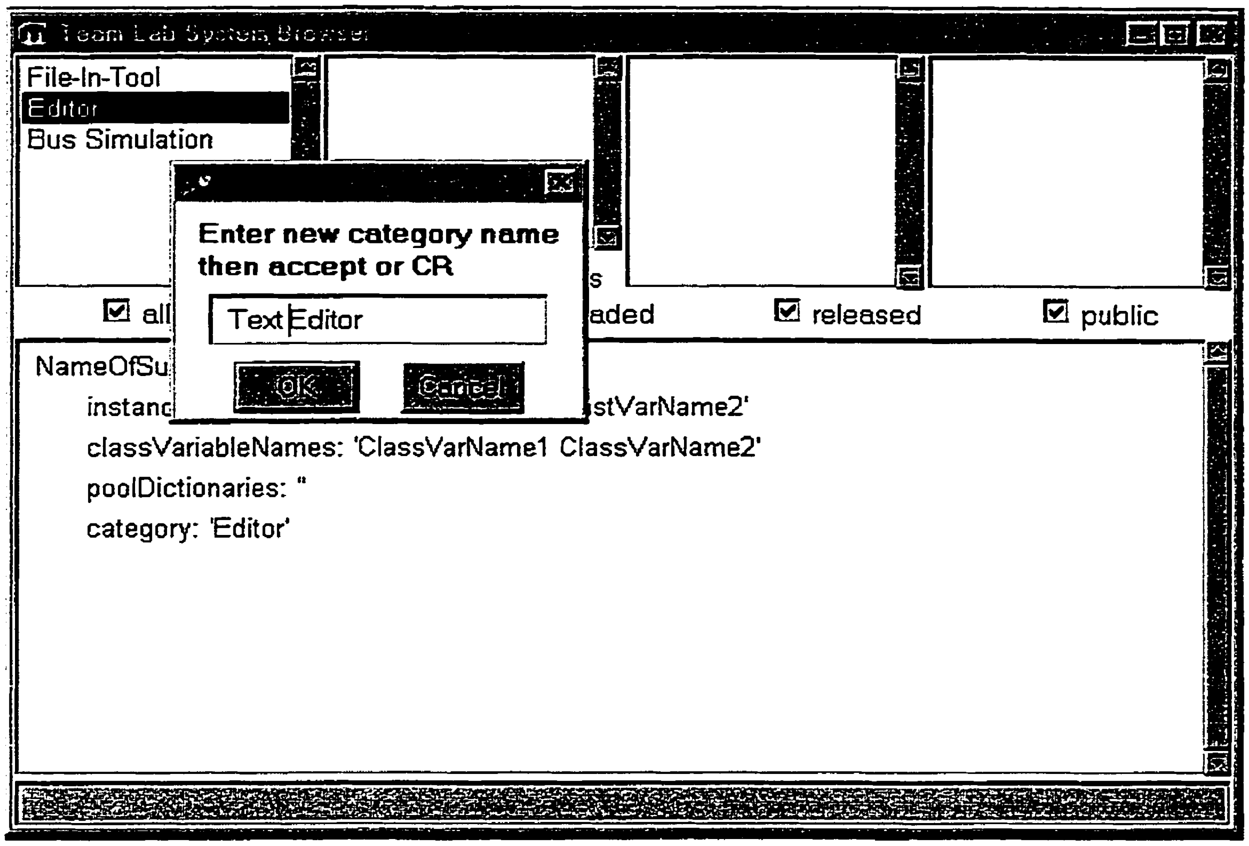
Users can perform the following operations upon categories using TL Browser (Figure 7.3).

add: Add a new category in the code repository. The program will ask the user to input the new category name (Figure 7.4). If the name is already exist in the code repository, the action will have no effect.

update: Update the category list to reflect changes in the code repository.

rename as: Rename the selected category. The program will ask the user to provide a new name.

remove: Remove the selected category if it is empty or all the classes in the category that are owned by the user.



```
NameOfSu
    instanc                                   stVarName2'
    classVariableNames: 'ClassVarName1 ClassVarName2'
    poolDictionaries: ''
    category: 'Editor'
```

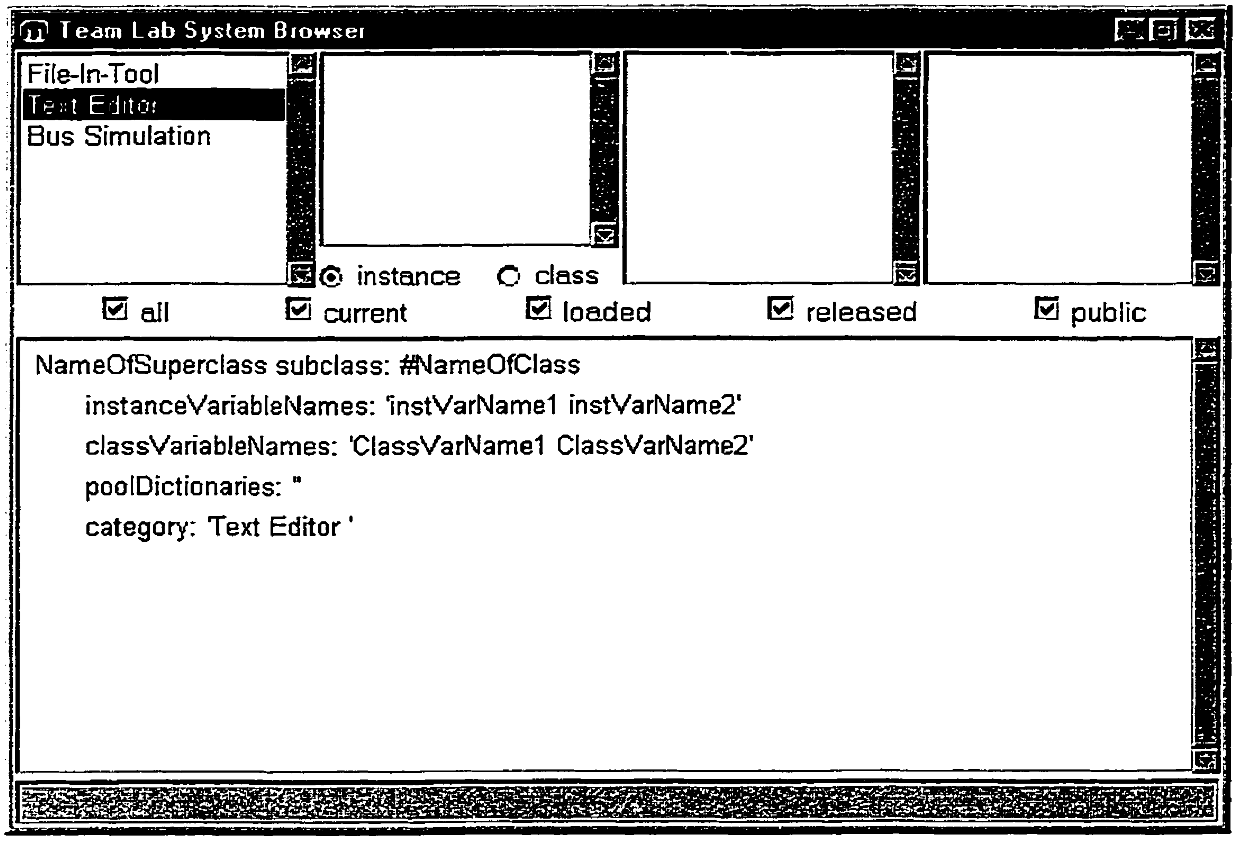
Figure 7.4 New category name request dialog

load: Load the selected category into server Smalltalk image. It also loads all classes in the category provided that they are owned by the user. Otherwise only classes owned by the user will be loaded.

unload: Remove the selected category with all its classes from the server Smalltalk image. If the user is not the owner of the category and all the classes in the category, only classes owned by the user will be unloaded.

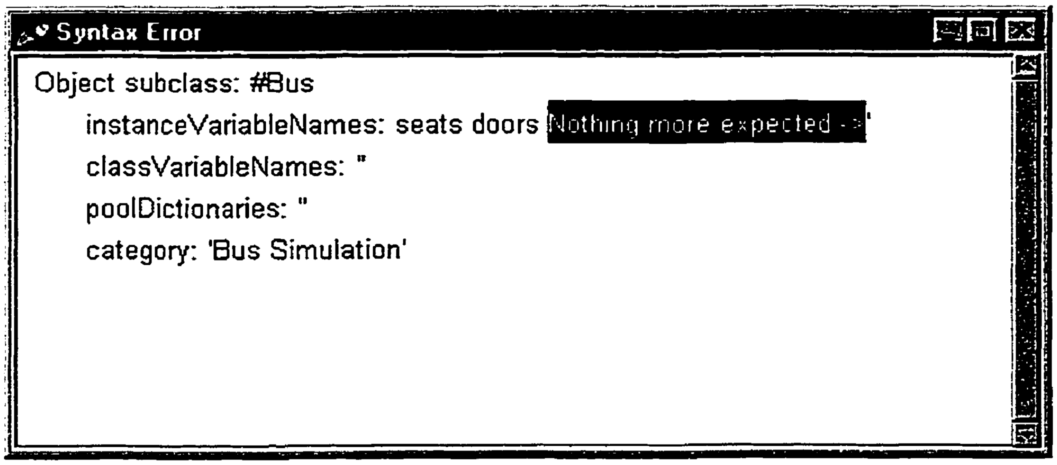
Users can perform the following class operations using TL Browser:

Define a class: Users can write and compile ("accept") class definitions by using the class template provided by the browser (Figure 7.5).



```
NameOfSuperclass subclass: #NameOfClass
    instanceVariableNames: 'instVarName1 instVarName2'
    classVariableNames: 'ClassVarName1 ClassVarName2'
    poolDictionaries: ''
    category: 'Text Editor '
```

Figure 7.5 Class template



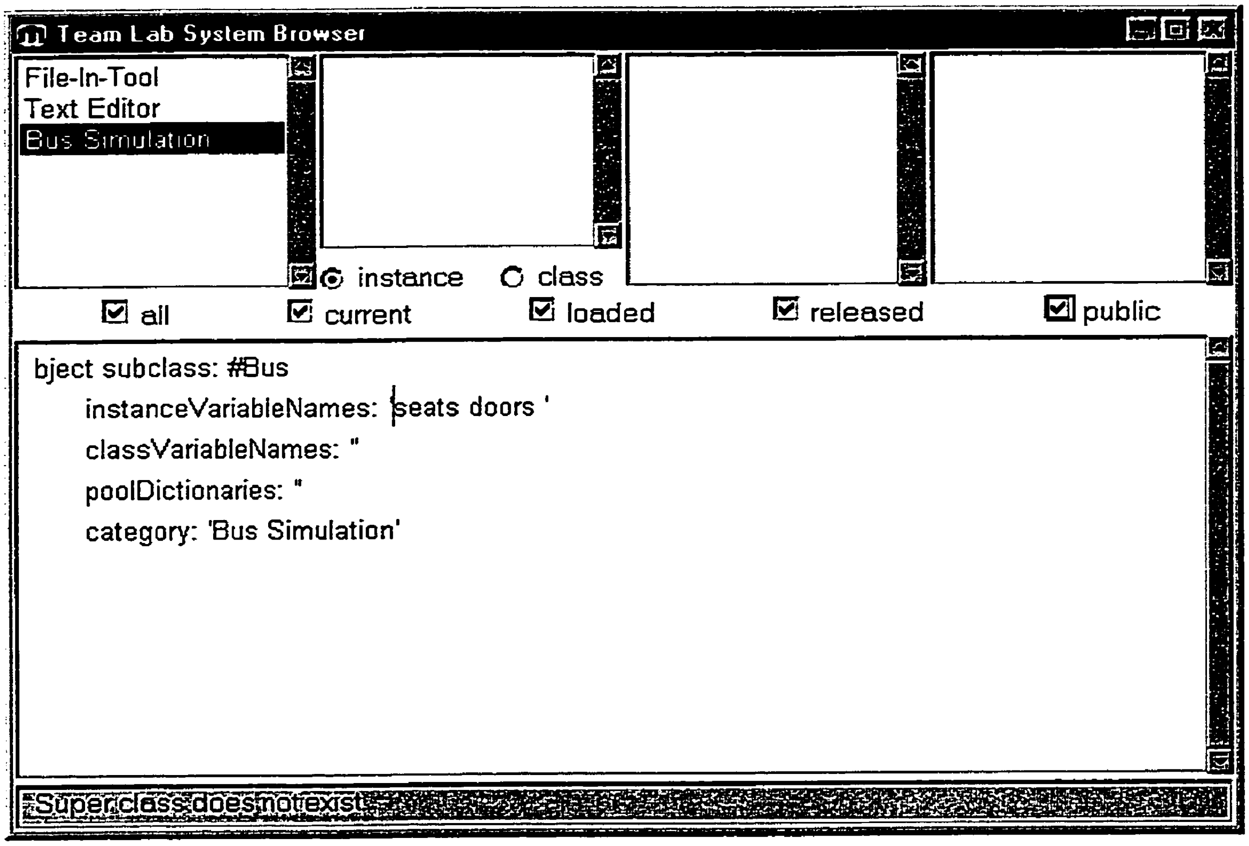
```
Object subclass: #Bus
    instanceVariableNames: seats doors Nothing more expected ->'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Bus Simulation'
```

Figure 7.6 Syntax error of class definition

Upon "accept", the browser first checks for syntax errors and prompts the user if there is any (Figure 7.6).

In Smalltalk System Browsers this kind of syntax error is displayed in the text area of the browser. In TL Browser it is not displayed in the same area. The reason is that during the process of compiling, a variety of error messages may be produced by the Smalltalk compiler and they are directed to the text area of the Smalltalk Browser. Some of the messages may be not valid in Team Lab since the code it uses is in the code repository in the server instead of in the local image. An example is the "super class not found" error discussed in the next paragraph. The compiler TLCompiler that TL Browser uses is a subclass of the Smalltalk class "Compiler" and some methods of the Smalltalk compiler are "primitive" code. The code for these primitive methods is not available to users, so it is not possible to overwrite them without knowing what they do. In this case users do not have a choice to select where an error message should go unless they write a new compiler. In order to avoid displaying these unwanted error messages in the text area, TL Browser ignores them and uses a separate window to display the wanted error messages.

Example: Handling of error message "The super class is not found": When compiling the code of a class definition, a Smalltalk compiler checks for the existence of the super class in the local image and if it cannot find one it sends a message to the browser to display. TL Browser gets that message too, but it ignores it because it may be not correct. Instead of displaying the message, TL Browser checks for the existence of the super class in the server Smalltalk image and the code repository. If the super class is not found, a "super class not found" error is displayed in the information box at the bottom of the browser (Figure 7.7), otherwise the class is saved in the code repository.

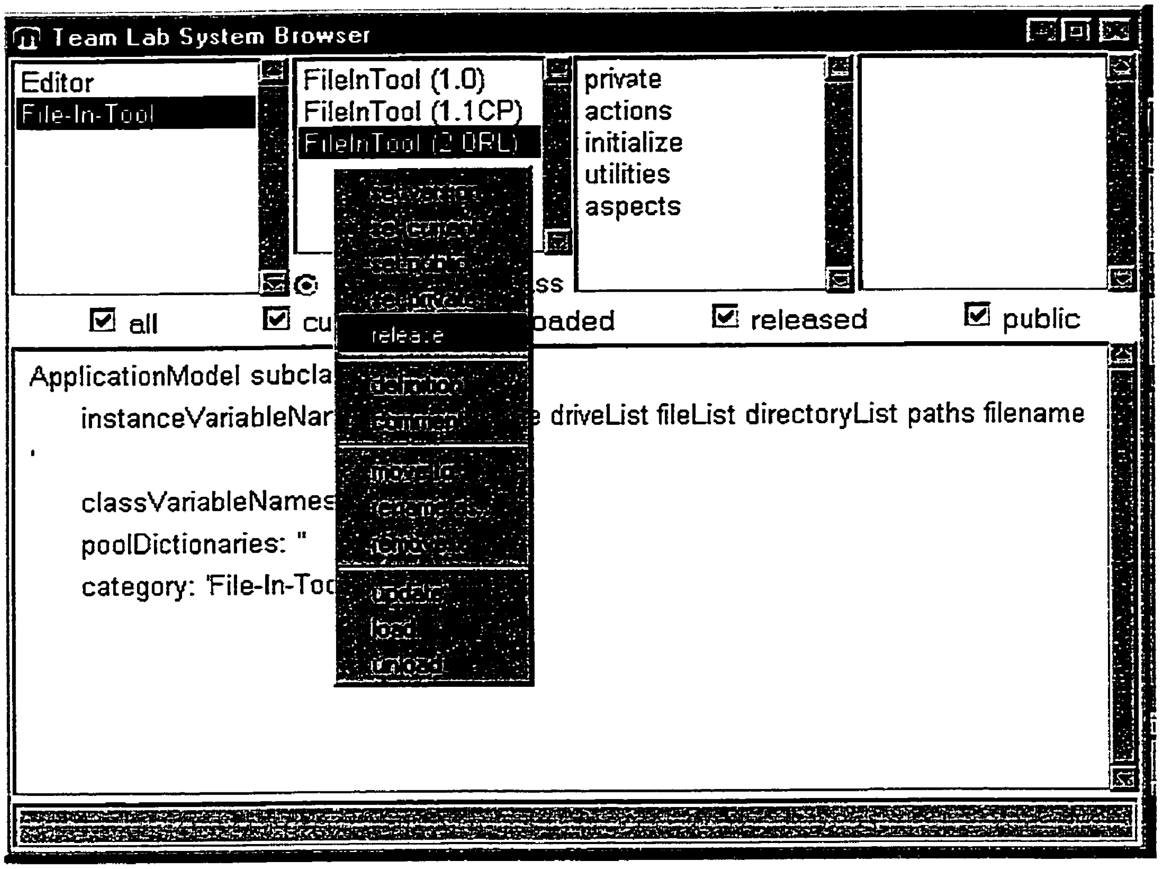


```
bject subclass: #Bus
    instanceVariableNames: 'seats doors '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Bus Simulation'
```

Superclass does not exist

Figure 7.7 Information area

Most operations on classes can be performed from the <operate> menu in the class list (Figure 7.8).

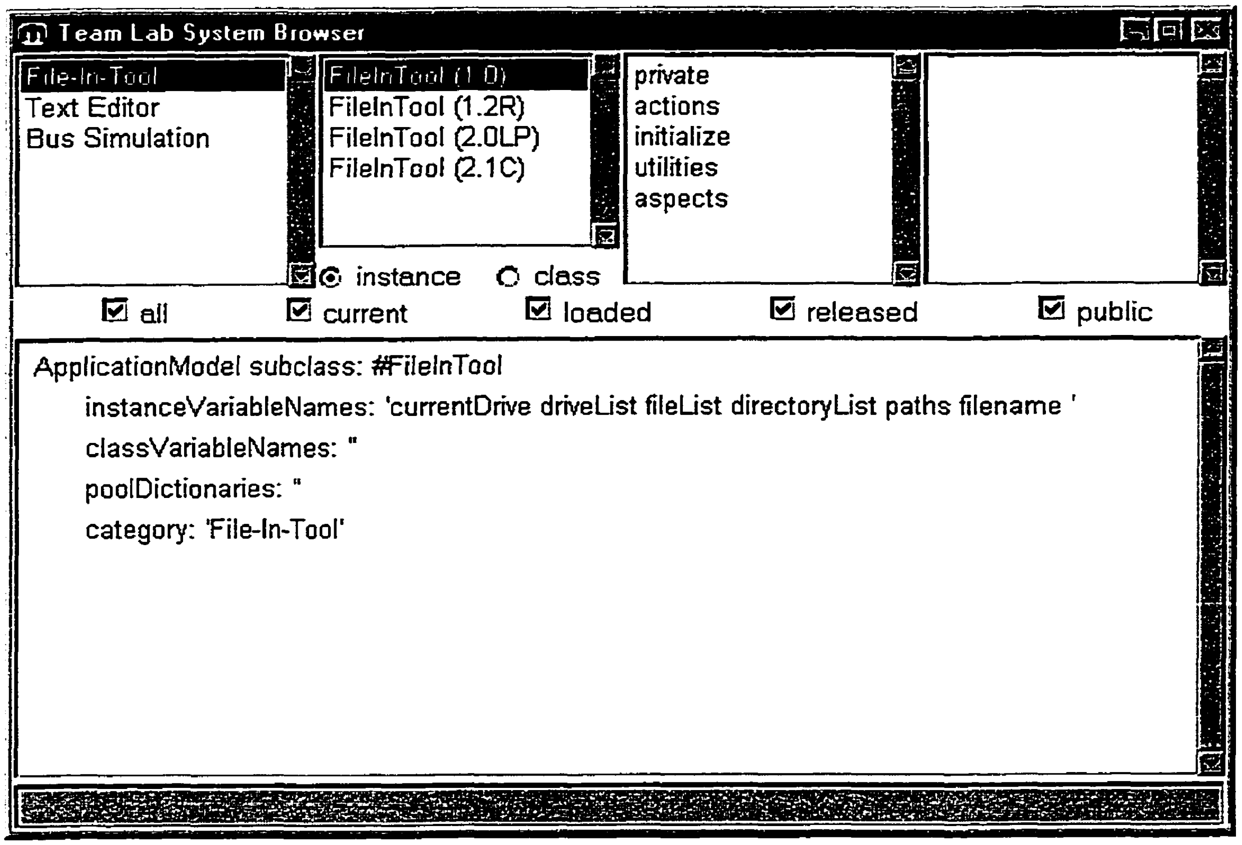


```
ApplicationModel subcla
    instanceVariableNar                e driveList fileList directoryList paths filename
'
    classVariableNames
    poolDictionaries: ''
    category: 'File-In-Toc
```

Figure 7.8 Class menu

set version: Users can give a class a version by selecting "set version". The browser will ask the user to input a version number. The default value is 1.0. In the example in Fig 7.8, multiple versions of class FileInTool are shown as FileInTool(1.0), FileInTool(1.1CP), FileInTool(2.0RL). The meaning of the letters attached to the version numbers is described below.

set current: Users can attach the "current" indicator to a class. The indicator can be used, for example, to indicate that this is the class the owner is currently working on. A current class is marked by a capital letter "C" after its version number (Figure 7.9). The purpose of the indicator is documentary only.



```
ApplicationModel subclass: #FileInTool
    instanceVariableNames: 'currentDrive driveList fileList directoryList paths filename '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'File-In-Tool'
```

Figure 7.9 Class versions and their indicators

set public: A class can be set as public. If a class is public and not locked, every developer can make changes to it, including adding/removing protocols, renaming protocols, adding/removing methods as well as modifying methods. A public class is marked by a capital letter "P" after its version number (Figure 7.9).

set private: A private class cannot be changed by anyone except its owner and only the owner can set it to public. A private class is not marked by any letter.

release: A class can be released by its owner. Released classes cannot be changed any more. A released class is marked by a capital letter "R" after its version number (Figure 7.9).

load: Classes can be loaded into the server Smalltalk image. While loading a class whose category does not exist in the server Smalltalk image, the category will be created. All protocols and methods in the class are loaded at same time. A class may have several versions. Only one of its version can be loaded at any given time. If one version is already in the server Smalltalk image and the user selects to load another version, the former version will be unloaded first. A loaded class is marked by a capital letter "L" after its version number (Figure 7.9).

unload: A loaded class can be unloaded. This removes the class from the server Smalltalk image but leaves it in the code repository.

The middle part of the browser contains check boxes labeled "all", "current", "loaded", "released" and "public". Users can use them to filter classes that are displayed in the class list. By default, all classes are displayed (Figure 7.9).

Classes can also be moved to other categories, removed and renamed. The text area in the lower part of the browser can display either the class definition or the class comment as in a Smalltalk System Browser.

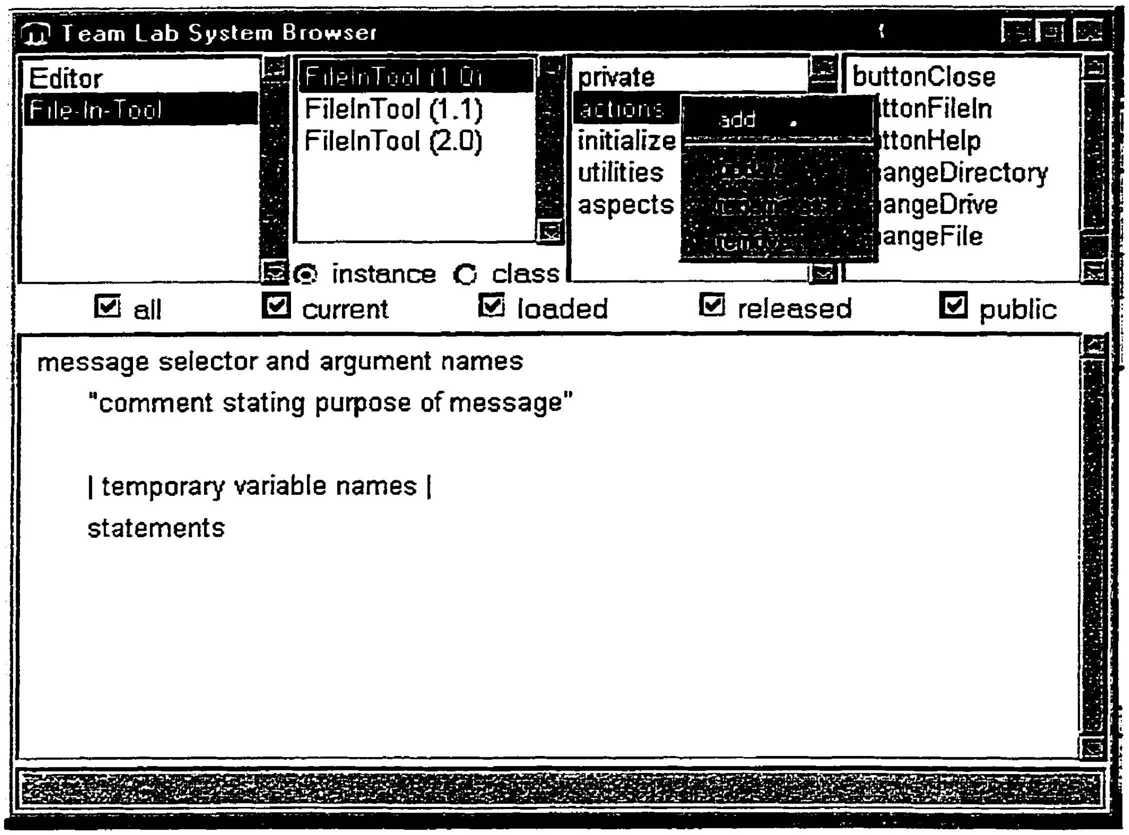
The function of the two radio buttons "instance" and "class" is the same as in the regular browser. By default, the selection is "instance", so the browser displays instance protocols and methods of the class. When the "class" radio button is selected, the browser displays class protocols and methods.

The protocol list in the browser displays all the protocols of the selected class (Figure 7.10). Users can add, rename and remove protocols.

add: Add a protocol. To be able to add a protocol, the user must be the owner of the class or the class must be public.

rename as: Rename the protocol. The user must be the owner of the class or the class must be public.

remove: Remove the protocol. The user must be the owner of the class or the class must be public.



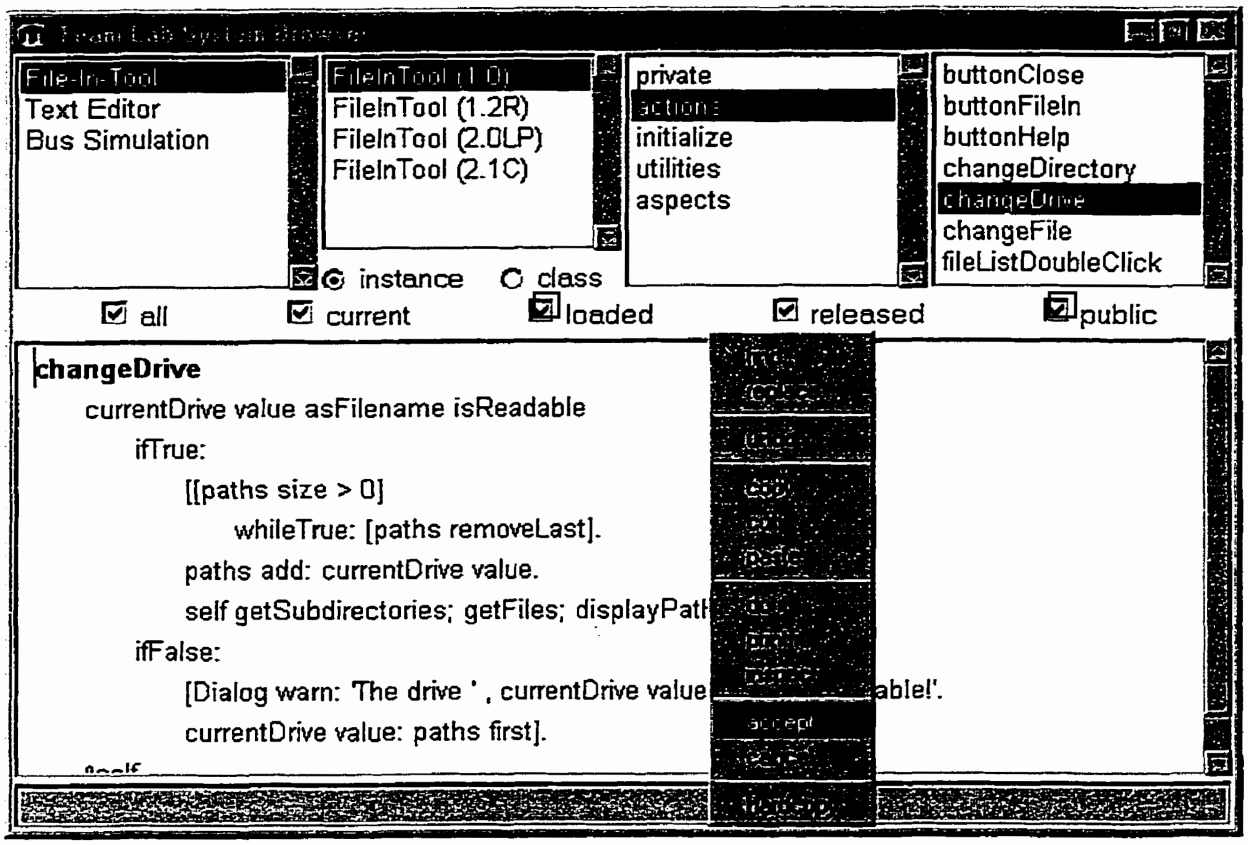
```
message selector and argument names
    "comment stating purpose of message"

    | temporary variable names |
    statements
```

Figure 7.10Protocol menu

The text area in the lower part of the browser is also used for displaying and editing method definitions (Figure 7.11). After writing or editing a method, users can compile and save it by selecting "accept" from the <operate> menu of the text area. To add or modify a method, the user must be the owner of the class or the class must be public. If there is an error in the method, the browser will prompt the user in a separate window (Figure 7.12).

Methods can be moved to another protocol or removed. The user must be the owner of the class or the class must be public.



File-In-Tool
Text Editor
Bus Simulation

FileInTool (1.0)
FileInTool (1.2R)
FileInTool (2.0LP)
FileInTool (2.1C)

private
actions
initialize
utilities
aspects

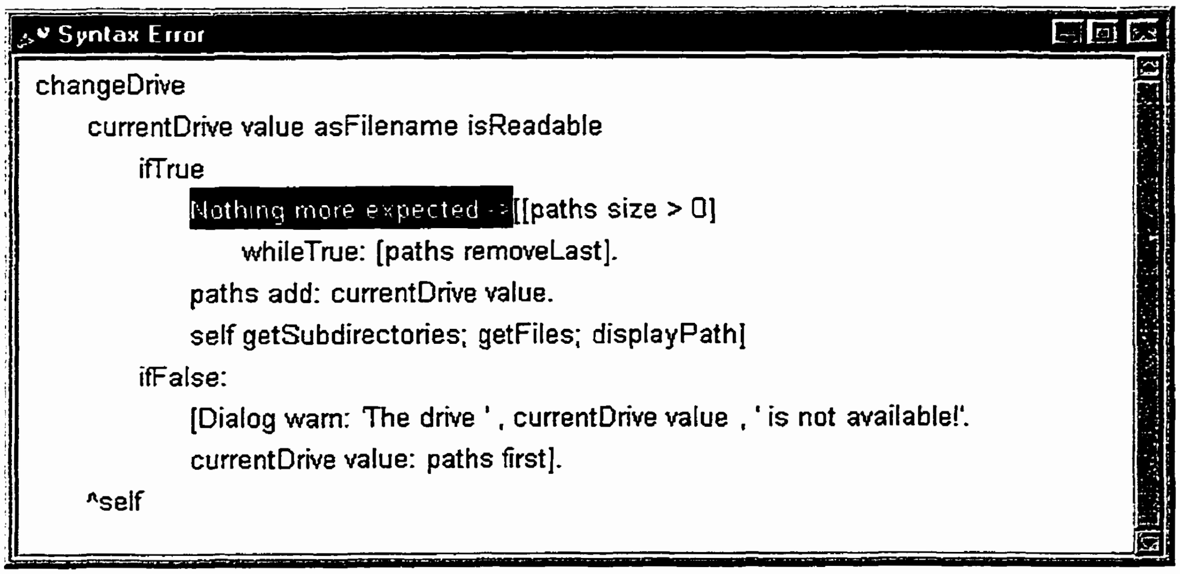
buttonClose
buttonFileIn
buttonHelp
changeDirectory
changeDrive
changeFile
fileListDoubleClick

[✓ instance] class

☑ all ☑ current ☑ loaded ☑ released ☑ public

```
changeDrive
    currentDrive value asFilename isReadable
        ifTrue:
            [[paths size > 0]
                whileTrue: [paths removeLast].
            paths add: currentDrive value.
            self getSubdirectories; getFiles; displayPat
        ifFalse:
            [Dialog warn: 'The drive ' , currentDrive value                able!'.
            currentDrive value: paths first].
```

Figure 7.11 Operate menu



```
changeDrive
    currentDrive value asFilename isReadable
        ifTrue
            Nothing more expected ->[[paths size > 0]
                whileTrue: [paths removeLast].
            paths add: currentDrive value.
            self getSubdirectories; getFiles; displayPath]
        ifFalse:
            [Dialog warn: 'The drive ' , currentDrive value , ' is not available!'.
            currentDrive value: paths first].
    ^self
```

Figure 7.12 Error message window for methods

7.3 TL Class Converter

TL Class Converter (Figure 7.13) is a tool for transferring class definitions between clients and servers. Its purpose is to allow developers to work offline and write code in their own images. When they connect to the server, they can use the converter to transfer their classes to the code repository or to download code from it.

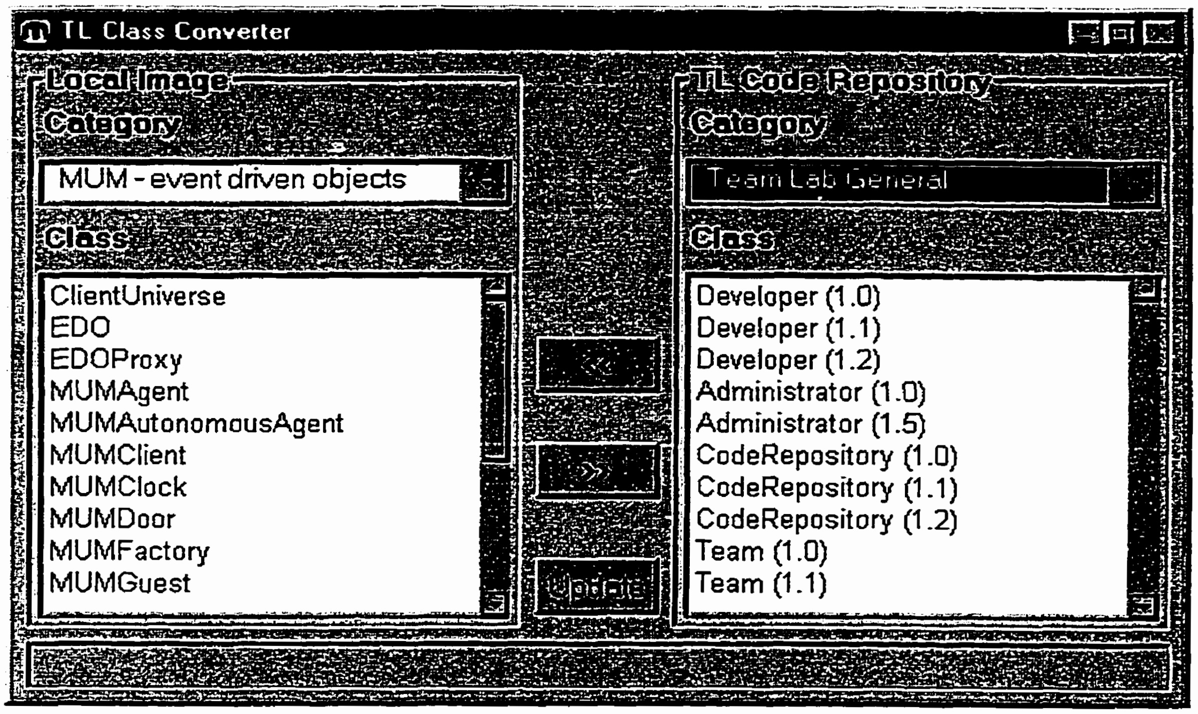


Figure 7.13 TL Class Converter

The combo box on the left side of the TL Class Converter displays categories present in the local image of the user. When a user selects a category, all classes in the category are listed in the list below it. The combo box and the list on the right are responsible for displaying categories and classes in the code repository respectively.

To transfer a class to the code repository, users must select a class in the local class list and click “>>”. The class converter will ask the user to enter a version number (Figure 7.14) because every class in the code repository must have a version. The default version is 1.0.

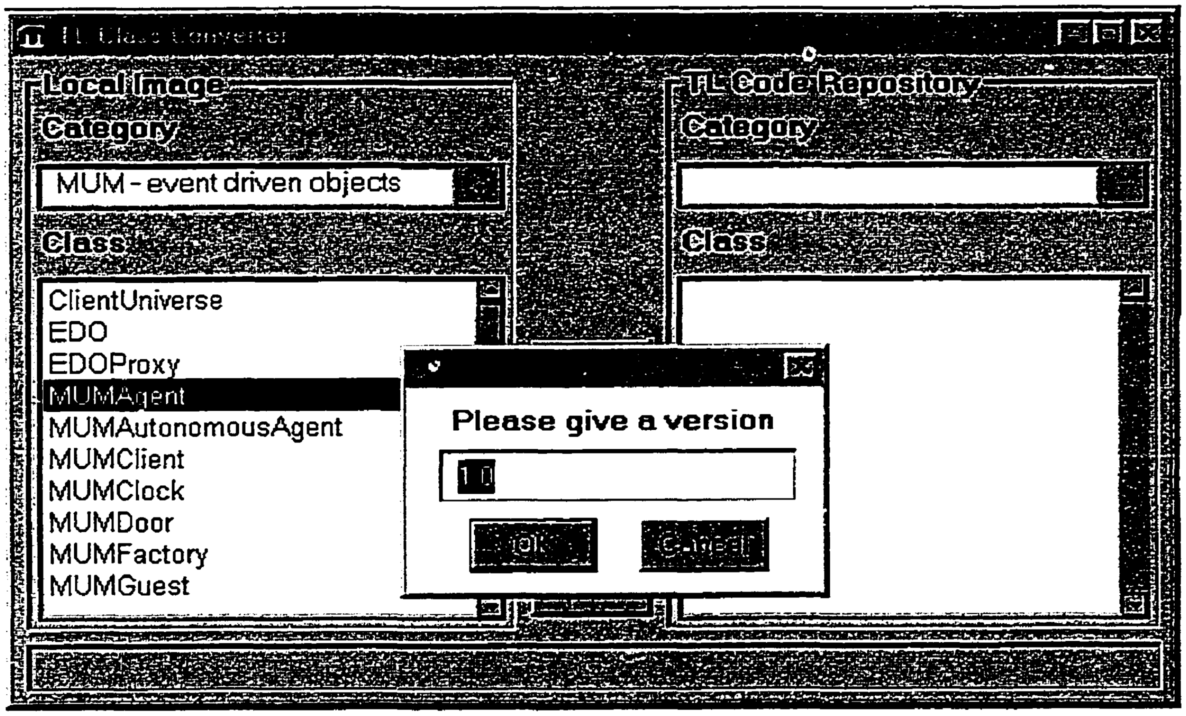


Figure 7.14 Version number request dialog

To protect some critical system classes in the server Smalltalk image, not all classes in the local image can be transferred to the code repository. The heuristic used to decide which classes can be safely transferred is as follows: If a class resides both in the server

Smalltalk image and in the local image but not in the code repository, it might be a system class. If such a class was transferred from the local image to the code repository, it could then be modified and loaded into the server Smalltalk image and overwrite the existing class. Because of this, Team Lab does not allow this class to be transferred.

To transfer a class from the code repository to the local image, the user must select a class in the class list on the right side and click "<<". If a class with the same name already exists in the local image, the transferred class will replace the existing one.

Chapter 8 Installation

This chapter describes where to get Team Lab and how to install it.

Team Lab is implemented in VisualWorks Smalltalk 3.1 and its parcels and installation guide can be downloaded from

http://ace.acadiau.ca/User/ivan/Research/CVE/download.html

To install Team Lab, users must have installed VisualWorks Smalltalk 3.1 and the ForkedUI parcel, which is a parcel from VisualWorks. Team Lab requires five parcels, namely Network, MUM – Core, MUM – Events, MUM – Tools and Team Lab. The first four parcels are MUM parcels and the last one is Team Lab parcel. To install Team Lab, load the parcels in following order:

Network

MUM – Core

MUM – Events

MUM – Tools

Team Lab

Since Team Lab is built on MUM, follow instructions about how to start a MUM universe that can be also found at above Web site.

Chapter 9 Conclusion and Future Work

Team Lab is an experimental collaborative virtual environment designed for teamwork. Because of its seamless integration with MUM environment, it is in some ways more powerful than existing code development tools. The combination of features of MUM and Team Lab gives the environment the potential of higher productivity and efficiency, while allowing more meaningful product documentation by capturing a broader development context as a basis for the management of organizational memory.

At present, Team Lab and MUM are still at an experimental stage of development. They have only been tested in a limited academic context with focus on operation, extendibility, and maintainability rather than CSCW (Computer Supported Cooperative Work) measures.

The main limitations of the present form of MUM are its purely event-driven operation and clumsy inter-universe navigation. Events present a high execution overhead and may slow down operation unacceptably for a large number of simultaneous users although scalability has not been tested. In the area of maintainability and extendibility, the drawback of MUM is that there are too many types of events, that events implement even operations that do not seem to require them, and that extension of existing EDOs and implementation of new EDOs is somewhat obscure.

The main limitation of Team Lab is its incomplete integration with the underlying VisualWorks environment. In particular, as team support became available as a part of VisualWorks in the form of StORE, a future version of Team Lab should take advantage of StORE facilities and build on them, integrating their functionality into MUM.

Future work[2] should include a new MUM architecture that removes complete reliance of events, integration of Team Lab and StORE and a variety of new tools and CSCW features. The new tools should include a more powerful remote browser in that users can select to browse classes in the local image, in the server Smalltalk image or in the code repository. The new browser should also provide all functions a Smalltalk System Browser has. A shared whiteboard is another useful tool that team members can use it for discussion as a real whiteboard. Once the design stabilizes and the implementation acquires sufficient functionality, it should be tested, initially in an academic environment and then in the "real world".

[2] A brief introduction to this thesis was accepted by CRIWG, 6th International Workshop on Groupware [18].

Glossary

CSCW	Computer-Supported Cooperative Work.
CVE	Collaborative Virtual Environment.
MUD	Multiple User Dungeons.
MOO	MUD, Object Oriented.
MUM	Multi-Universe MOO.
StORE	Smalltalk Open Repository Environment
UML	Unified Modeling Language.

Bibliography

[1] Object Technology International: ENVY/Developer/Developer, http://www.oti.com/briefs/ed/edbrief5i.htm.

[2] Cincom: VisualWorks Smalltalk 5i, http://www.cincom.com.

[3] Cincom: ObjectStudio, http://www.cincom.com/objectstudio/index.html.

[4] MUM, http://ace.acadiau.ca/User/ivan/Research/CVE/index.html

[5] Rémy E.: Collaborative Networked Communication: MUDs as Systems Tools, http://www.ccs.neu.edu/home/remy/documents/cncmast.html

[6] Mud FAQs, http://www.mudconnect.com/resources/Mud_Resources:Mud_FAQs.html

[7] Diversity University, http://arwen.marshall.edu/

[8] Bruckman A. and Resnick M.: The MediaMOO Project: Constructionism and Professional Community, http://www.cc.gatech.edu/fac/Amy.Bruckman/papers/convergence.html

[9] Haynes, C., Holmevik, J. R., High wired: On the design, use, and theory of educational MOOs, University of Michigan Press 1998.

[10] Churchill, E., Bly S.: Virtual Environments at Work: Ongoing use of MUDs in the Workplace, WACC 1999.

[11] Churchill, E., Bly S.: It's all in the words: Supporting work activities with lightweight tools, Group 1999.

[12] TeamWave, http://www.teamwave.com/

[13] Tomek I., Giles R.: Virtual Environments for Work, Study, and Leisure, Journal of the Virtual Reality Society, volume 4, number 1, 1999.

[14] Tomek I.: The Design and Implementation of a MOO, to be published in Journal of Network and Computer Applications.

[15] JerseyMOO, http://ace.acadiau.ca/User/ivan/Research/CVE/index.html

[16] Rheingold H.: The Virtual Community, http://www.rheingold.com/vc/book/

[17] Burka L.: The MUDline, http://www.opensite.com.br/random/servicos/muds/mudline.html

[18] CRIWG, 6th International Workshop on Groupware, http://criwg2000.di.fc.ul.pt/