DOOD: Distributed Object-Oriented Software Development Environment

Tae-Hoon Kim, Woo-Chang Shin, Geun-Duk Park, Tae-Heun Lee,
*Tae-Gyun Kim, Yeong Gil Shin, Chi-Su Wu
Department of Computer Science, Seoul National University
San 56-1, Shilim-Dong, Kwanak-Ku, Seoul, Korea
*Department of Computer Engineering, Pusan University of Foreign Studies
Uam-Dong 55-1, Nam-Gu, Pusan, Korea
kth@cs.snu.ac.kr, {smurf,dean,lth,wuchisu}@selab.snu.ac.kr,
ktg@taejo.pufs.ac.kr, yshin@cglab.snu.ac.kr

Abstract

The object oriented method becomes very popular in software development because of its efficiency of automating software development processes and reusing software modules.

To date, various object-oriented CASE tools have been used, however, they have concentrated on providing tools for developing a software system within a geographically localized area. Even though we can use a general purpose CSCW tool for the cooperative software development in distributed environment, current CSCW systems do not support any efficient object-oriented software development method.

This paper discusses the ways of combining CSCW features with an object-oriented CASE tool by introducing the experimental distributed object-oriented software development environment, DOOD (Distributed Object-Oriented Designer). DOOD supports three models (object model, dynamic model, functional model) of OMT (Object Modeling Technique) and provides consistent and transparent user interfaces for minimizing effects of geographically dispersed developing environment. Our system is implemented based on a client-server architecture. DOOD/Server manages the communication among clients and guarantees concurrent and consistent accesses to all the objects in the central repository. DOOD/Client provides modeling and collaboration schemes over TCP/IP network.

1. Introduction

As the software requirements are increased, the number of software developers has grown, and software development environment has adopted various features to support their demands. Software development environment has been evolved in the form of stand-alone type on the assumption that all those who are involved in a project work at one place. That is, software development environment is built on a powerful workstation and developers must log in the workstation to carry out their project.

However, the new demands of software functions from the current computing environment such as multimedia and virtual reality systems are so diverse, a centralized software development cannot provide a unified way of developing software systems.

Many people in software engineering field are interested in distributed software development environment by which the defects of stand-alone type has been improved. Since the system can be distributed over several computers, the cooperation among software developers located in different areas makes it possible to develop a software which requires diverse technology.

Recently, CSCW (Computer Supported Cooperative Work) are being used for the collaboration of geographically dispersed users. The toolkits supporting CSCW is also being developed in various fields. To implement a groupware, we should first consider the type of cooperative work. There are four types of cooperative work: [1]

- Synchronous mode: Participants work at the same time in the same location.
- Distributed synchronous mode: Participants work at the same time in different location.
- Asynchronous mode: Participants work at differ-
ent time in the same location.

- Distributed asynchronous mode: Participants work at different time in different location.

A typical groupware supporting distributed synchronous cooperative work is the desktop conferencing system. The desktop conferencing system is appropriate for general purpose of communication, but is not suitable for software development environment in which enormous documents are generated and participants communicate with one another mostly with diagram or program code. Therefore, to support distributed synchronous cooperative work in software development environment, it is desirable to add CSCW features in the existing software development environment. Although it is more difficult to implement such a system rather than using generic wrapper such as shared window system, we can get more flexibility of the system.[2]

The object-oriented paradigm becomes very popular because of its high development productivity. It can maximize software reusability and provide an easy way of automating the software development. Even though various object oriented CASE (Computer Aided Software Engineering) tools have been developed, they do not much consider the effective collaboration facilities.

This paper presents the design and implementation of DOOD (Distributed Object-Oriented Designer). DOOD is the distributed object-oriented software development environment which is an object-oriented CASE tool supporting OMT[3] (Object Modelling Technique) proposed by Rumbaugh et al. combined with CSCW features.

With DOOD, one can implement an object model with C++ language. By analyzing user interfaces of existing object-oriented CASE tools such as OOTher, Rational Rose, and WithClass, DOOD provides a consistent and effective user interface. Also, by using Object Modeller, Shared Scratchpad, and audio communication facility, DOOD supports distributed synchronous cooperative work between a manager and developers by allowing the transmission of multimedia data through the TCP/IP network. Change control, version management, and distributed asynchronous cooperative work facility are also provided by the repository based on a client-server model.

2. Related Works

2.1. Object-Oriented CASE Tools

There are many object-oriented CASE tools on various platforms such as PCs and workstations, in the form of commercial tools and shareware. We got demo versions and shareware versions from internet and evaluated the performance of each one. Since they are all stand-alone type and do not support CSCW features, we focus on comparing their user interface with that of our system.

2.1.1. WithClass

WithClass by MicroGold Software supports major object-oriented methodologies such as Rumbaugh OMT, Booch OOD, and Coad/Yourdon OOA and generates source codes using already developed scripts.

It supports direct and interactive input methods and shows the relationship between objects by automatic routing.

However, while a user inputs characters, he/she cannot correct the error. Since the automatic routing feature is incomplete, meaningless diagrams may be created. Popup menu, which is more convenient for editing graphic symbols, is not available.

2.1.2. OOTher

OOTher by Roman M. Zielinski supports Coad/Yourdon OOA. Diagrams are created only by dialog, which may be very tedious work.

It supports automatic routing to draw the relationship between objects. However, since user cannot correct the generated diagrams, there may exist meaningless diagrams.

2.1.3. Other Object-Oriented CASE Tools

The user interfaces of other object-oriented CASE tools such as Object Domain by Dirk Vermeersch, Show-CASE by MultiQuest and Rational Rose by Rational Software are also inconvenient because they have a dialog based input system.

2.2. Groupware

2.2.1. TeamRooms

TeamRooms[4][5] is a groupware system that fills the role of a team room for groups whose members can work both co-located and at a distance. Users run TeamRooms clients that connect over a network to a server which provides a number of rooms. Each room contains generic communication tools (a chat tool and a shared whiteboard).

It is implemented using groupware toolkit GroupKit[6][7], augmented to support centralized processes, user
authentication, a versioned persistence repository and embedded conferences.

GroupKit lets developers build applications for synchronous and distributed computer-based conferencing. GroupKit and its applications run on UNIX workstations under an X11 environment. It uses the interpreted Tcl language and Tk interface toolkit[8] and Tcl-DP socket extensions.[9]

However, GroupKit application may suffer a performance penalty if there is a high degree of parallel activity by users.[7] Since Tcl/Tk lacks the features of object-oriented paradigm such as an object-oriented widget set, developers using GroupKit have some difficulties in implementing object-oriented software applications. GroupKit does not directly support audio communication, therefore users should use other facilities, such as telephone, to transfer audio data.

TeamRooms is a prototype groupware system for general purposes, but lacks object-oriented modeling facility.

2.2.2. ConversationBuilder

ConversationBuilder[10] is a generic collaboration environment developed at the University of Illinois at Urbana-Champaign.

It consists of a conversation engine and multiple user interfaces. Tools are connected together by a message bus system.

ConversationBuilder provides the following facilities:

- The text server is a tool that allows users to display and edit textual components of nodes in the hypertext system.
- The graph browser allows users to display relations among objects.
- The shared whiteboard called SketchPad supports collaborative editing of sketches.

ConversationBuilder is also a generic collaboration environment which does not directly support the object-oriented software development.

3. Design and Implementation of DOOD

3.1. System Requirements and System Architecture

In developing DOOD, various requirements for a distributed object-oriented software environment are considered. From the analysis of previous works, we have identified the following set of requirements:
• Supporting three models (object model, dynamic model, functional model) of OMT.
• Real-time communication of plain texts, diagram information and graphic image data.
• Real-time communication of audio data.
• Distributed synchronous mode collaboration.
• Distributed asynchronous mode collaboration.

In the previous discussion, we know that object-oriented CASE tools cannot support CSCW features and general purpose groupwares are not appropriate for object-oriented software development due to the lack of a proper modeling tool.

For the implementing a distributed object-oriented software development environment, DOOD, we combine an object-oriented CASE tool with collaborative software engineering schemes.

Figure 1 illustrates the overall system architecture. DOOD/Server consists of Communication Server, Repository Server, Transaction Manager, and Central Repository. Transaction Manager controls concurrent repository accesses by synchronizing reads and writes.

DOOD/Client consists of Object Modeller, Dynamic Modeller, Functional Modeller, Shared Scratchpad, Communication Client, and Local Repository. Communication Client serves as the interface between DOOD/Client and external DOOD/Servers or DOOD/Clients.

DOOD users can not only develop a software product in object-oriented method but also participate in synchronous/asynchronous cooperative works using Object Modeller, Shared Scratchpad, and audio communication facility.

3.2. Communication Architecture

Communication Server receives the request message from a Communication Client process in DOOD/Client and retrieves the specified information from Central Repository.

Communication Client sends a special message to request channel establishment to Communication Server. Once the communication channel is established, DOOD/Client can get/put the information from/into Central Repository.

Communication Client can also send connection request message to another DOOD/Client. If the receiver sends back an acknowledgement message, a synchronous communication channel is established.

Figure 2 illustrates the communication architecture of DOOD. There is usually one Communication Server for a community of collaboration, and its address is "well-known" to other clients.

Both Communication Server and Communication Client have an unique access point. If a client wants to contact with a server, the client should know the access point of the server.

In Figure 2, there are two different communication channels between server-client and client-client connection. Control channel only transfers control messages, while data channel transfers actual application data such as plain texts, graphic images, and audio data.

3.3. Multimedia Data Transmission

It is possible to transfer multimedia data (graphic images and audio data) by using DOOD's communication facilities.

DOOD transfers graphic image in the form of pixmap data structure which is provided by X Window System. Because the amounts of graphic image data to be transferred are relatively small and not continuous, they can be transferred through data channels synchronously.

If a DOOD/Client sends a connection request message to another client and receives an acknowledgement message, both clients change their modes to synchronous collaboration mode. In synchronous collaboration mode, a data channel between them is established so that both clients can transfer multimedia data synchronously.

Participants in the synchronous collaboration can participate in the audio conferencing by using microphones attached to their workstations.
Since the amounts of generated audio data are relatively large and continuous, there should be an effective method to handle them. DOOD handles the audio data by using a temporary buffer pool not to lose any packet. The size of a buffer pool should be decided according to the analysis of network condition.

3.4. Object Modeller

The object modeller in DOOD supports object model of OMT and provides an efficient user interface. The OMT object model describes the structures of classes in a target system and the relationship among them in a static manner. It consists of classes, associations, generalizations, aggregations, qualifications, and link attributes.

Diagrams can be edited by using a dynamic popup menu. According to the type of a symbol under the mouse cursor, the corresponding popup menu is appeared. Therefore, a user can edit the symbol without moving mouse. We provide a direct text input method to input more quickly than a dialog-based method.

In all working windows including object modeller, the usage of mouse buttons are uniformly defined for consistent user interface.

Button 1 is used mainly in drawing a graphic object. It is also used as an indication of beginning or end of creating, moving, copying, magnifying, or reducing the size of a graphic object.

Button 2 is used to indicate the beginning of text input. Clicking this button at a position causes the cursor appeared and allows text input from that position. The user can terminate the text input state by just clicking another button or pressing Esc key. The user can also correct an existing text by moving the mouse pointer to the corresponding text area and clicking this button.

Button 3 is used to create and select a popup menu. DOOD event handler continuously keeps track of a mouse pointer and highlights the graphic object under current mouse pointer. It shows appropriate popup menu according to the object. In this way, we can reduce unnecessary mouse movement and allow the easy access to various services for each object.

Object modeller is designed using the object modelling method of OMT. We devide it into six modules[3] and model it using OOD (Object-Oriented Designer)[12]. The modelling results are converted to C++ codes by the OOD code generation facility. Figure 3 shows the dia-

![Figure 3. Main Module](image-url)
gram of the main module, which is printed using OOD diagram printing facility.

3.5. Dynamic Modeller and Functional Modeler

The aspects of a system concerning with time and changes are dynamic module[3] of OMT. The major dynamic modeling concepts are events, which represent external stimuli, and states, which represent values of objects. The state diagram is a standard concept that has been used in the literature.

The functional model[3] of OMT describes computations within a system. It specifies the meaning of the operations in the object model and the actions in the dynamic model. The functional model consists of multiple data flow diagrams which specify the meaning of operations and constraints. A data flow diagram contains processes that transform data, data flows that move data, actor objects that produce and consume data, and data store objects that store data passively.

DOOD/Client contains Dynamic Modeller and Functional Modeller. Both have the same user interface (for example, Button3 invokes object-specific dynamic popup menu) as that of Object Modeller.

3.6. Shared Scratchpad

After the data and control channel between clients have been established, users can exchange diagrams, texts and graphic images via Shared Scratchpad in real-time. All information written on Shared Scratchpad is transmitted to all connected user's Shared Scratchpad and synchronized.

3.6.1. Screen Dump

Assume that the communication channel between client A and client B has been established. If A wants to communicate with B, A selects a region and chooses “Screen Dump” at the popup menu. Then, the information of the selected screen is copied to A's Shared Scratchpad and transmitted to B's Shared Scratchpad.

3.6.2. Editing

Freehand sketch with mouse, text input, image input, and file i/o can be done in Shared Scratchpad. All editings on Shared Scratchpad are mirrored on another user's Shared Scratchpad. Using this Shared Scratchpad, the system provides a chat tool.

3.7. Repository

Central Repository consists of Project Information Area, Project Working Area and User Environment. Project Information Area stores miscellaneous project management information such as user id, project description, active user list. A DOOD/Server administrator is responsible for maintaining all the information stored in this area.

Since DOOD/Server also maintains each user's environment in Central Repository, each user can work in the same working environment regardless of his/her physical location. In order to reduce the communication overhead between the server and clients, DOOD/Client has User Environment Cache in its local repository.

![Figure 4. Repository Architecture](image)

DOOD/Client is an independent object-oriented CASE tool which support object-oriented software development life cycle. Even if it is not connected with a DOOD/Server, user still can use the whole feature of CASE tool excluding collaboration facility. In disconnected mode, user's working data are stored in Private Working Area. In connected mode, data stored in User Environment Cache should be synchronized with those in the User Environment of Central Repository by invoking flush/cache operation.

Project Working Area consists of Temporary File Pool and BaselineVersion Pool. If the owner of an object in User Environment invokes move_to_check operation, the object is moved to Temporary File Pool. Objects in Temporary File Pool should be verified to check its consistency and completeness. If the object is...

[1] If an user logs in DOOD/Server, the server and client change their mode to be connected. Then, the user becomes active and inserted into Server's active user list. Distributed synchronous collaboration is possible between active users.
consistent and complete, the project manager checks it in Baseline Version Pool.

Objects in Baseline Version Pool are grouped together as a unit for the purpose of versioning, called configuration. If the project manager invokes make version operation, the current configuration becomes a baselined version.

Workspaces provide contexts for both shared and private work. Whenever an user manipulates configurations and the objects in them, there must be a current workspace.

DOOD/Server’s version management facility allows an user to create alternative versions of configurations, which make it possible for each user to carry on his(her) work on the configuration even if it is already checked out by someone else. If different versions are checked out, these separate branches of development might be merged. The project manager is responsible for preserving the consistency among objects in Baseline Version Pool. However, in many cases, only the owner of the checked-out objects modifies them (others only view the objects), the merging process might not be complex.

For a large software development effort, uncontrolled changes rapidly lead to chaos. DOOD/Server supports change control[13] to provide a mechanism for the control of change.

Repository Server’s access control governs which users have the authority to access and modify a particular configuration object. Transaction Manager’s synchronization control ensures that parallel changes, performed by two different users, do not overwrite each other.

4. Discussions

The main objective of DOOD is to provide a distributed collaboration paradigm on an object-oriented software development environment. All the registered users in a DOOD/Server can participate in distributed cooperative work by using Object Modeler, Shared Scratchpad, and audio communication facilities.

![Figure 5. Synchronous Collaboration between DOOD/Clients](image)
In the software development life cycle, a diagram can be an effective method of the communication among participants of a project. In many cases, geographically dispersed participants of a project use telephones to communicate with each other. So, DOOD's distributed collaboration features which provide WYSIWIS view of a diagram and audio conferencing facility can be an useful method for cooperative software engineering.

In figure 5, DOOD/Client(A) and DOOD/Client (B) have WYSIWIS (what-you-see-is-what-I-see) views of an OMT diagram document. Audio communication channel is also established between them.

DOOD provides real-time multimedia data transmission (graphic image, audio data) features which cannot be provided by groupware toolkits described in 2.2.

The implementation platform of DOOD is UNIX, specifically SUN workstations running Solaris 2.5. Motif/X11 toolkits are used for graphical user interface implementation, TCP/IP based socket interface is used for communication between client-server and client-client. Since it is independent of any particular hardware, DOOD can be ported to run any UNIX environment quite easily.[15] Moreover, X window based GUI is faster than the interface based on a script language such as Tcl/Tk.

Repository Server maintains multi-version of an object, which prevents object inconsistency, deadlock, and lock waiting.[16] Maintaining multiple versions of an object requires more storage and causes heavier load to Repository Server. However, the repository only stores the delta from the previous version, storage requirement is not a serious problem. Multiple versions of an object can be used to back up the original object when system crash occurs.

5. Conclusion and Future Work

In this paper, we have presented the ways of combining CSCW features with an object-oriented CASE tool in UNIX environment.

Since DOOD provides distributed synchronous/ asynchronous collaboration schemes, DOOD users can do various cooperative works.

Some of the issues that need to be addressed for future DOOD versions are:

- Java, CORBA support
- DOOD/Client needs to be ported to run on various platform.

References