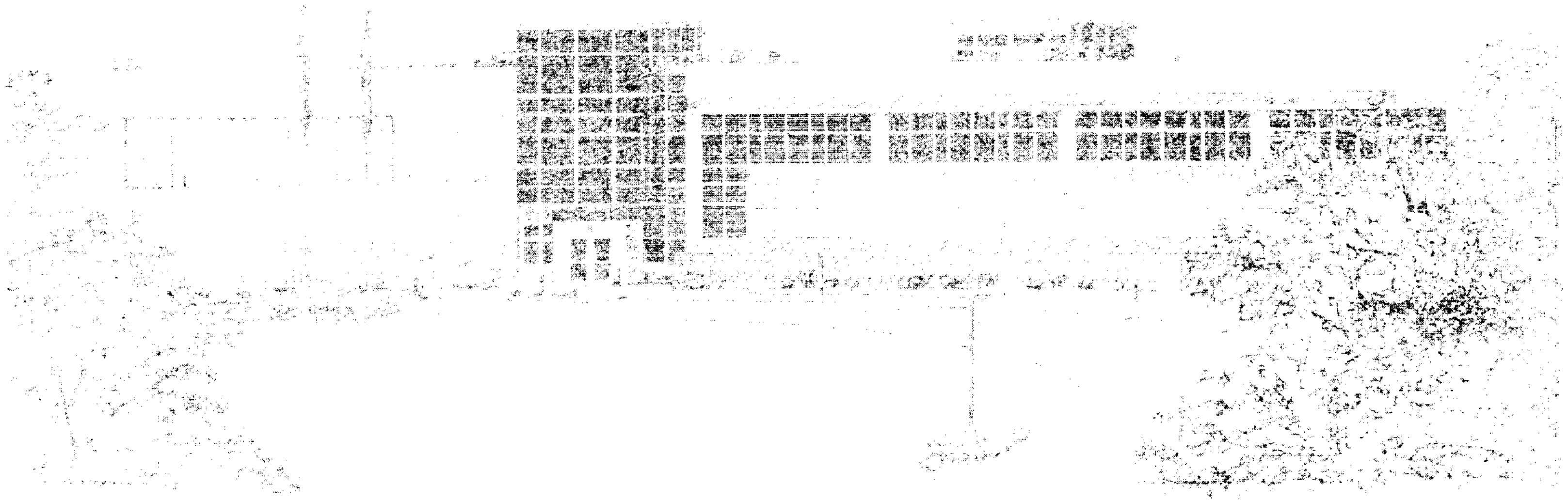
NASA/WVU Software IV & V Facility
Software Research Laboratory
Technical Report Series

NASA-IVV-97-018
WVU-IVV-97-018
WVU-CS-TR-97-020

Providing the Persistend Data Storage in a Software Engineering Environment Using Java/COBRA and a DBMS

Swarn S. Dhaliwal



National Aeronautics and Space Administration

West Virginia University

NASA-IVV-97-018

NASA IV&V Facility, Fairmont, West Virginia

Providing the Persistent Data Storage in a Software Engineering Environment Using Java/COBRA and a DBMS

Swarn S. Dhaliwal

December 5, 1997

This research was funded under cooperative Agreement #NCC 2-979 at the NASA/WVU Software Research Laboratory.

Providing the Persistent Data Storage in a Software Engineering Environment Using Java/CORBA and a DBMS

THESIS

Submitted to the Eberly College of Arts and Sciences
Of
West Virginia University
In Partial Fulfillment of the Requirements for
The Degree of Master of Science

By

Swarn S. Dhaliwal

Department of Computer Science
West Virginia University
Morgantown, WV 26506

December 1997

APPROVAL OF ADVISORY COMMITTEE

Steve M. Easterbrook, Ph. D. ____________________________

V. "Juggy" Jagannathan, Ph. D. ____________________________

John R. Callahan, Ph. D., Chair ____________________________

To my brother, my sisters,
my sister-in-law, and
my wife

Acknowledgements

I wish to express my deep sense of gratitude to my major Professor Dr. Jack Callahan for giving me an opportunity to be a part of the wonderful SRL (Software Research Laboratory). His ever-willingness to provide technical support and his ability to stimulate intellectual interest has had a profound influence on me and has gone a long way in the successful completion of this thesis.

I can never express enough appreciation to my supervisor Dr. Steve Easterbrook whose able guidance and foresight made the planning and completion of this thesis possible. His cheerful and friendly disposition, his ever-willingness to help, and his ability to provide constructive criticism and intellectual stimulation are exemplary. I consider myself fortunate having had an opportunity to work with him.

Thanks are also due to Dr. V. "Juggy" Jagannathan for agreeing to be on my committee and his willingness to provide technical help during planning the thesis research. I wish to thank everybody at SRL for always being ready to help and for providing an excellent work environment. I have really enjoyed being a part of such an outstanding organization. I express my gratitude to Todd Montgomery for providing technical help. Appreciation is also extended to my fellow graduate students : Mani (my office-mate), Zhong (my team-mate), Reshma, and Nicholay for their help and pleasant companionship. They have really made my stint as a computer science graduate student enjoyable.

I will always be grateful to my parents whose love and moral support have enabled me to come thus far. No words suffice to thank my elder brother, my sisters, my sister-in-law, and my wife for their love and moral support. The love of my nieces and nephews has always been a source of inspiration for me.

The financial support provided by NASA-IV&V Cooperative Research Project in the form of a Graduate Research Assistantship is gratefully acknowledged.

Table of Contents

List of Figures

ABSTRACT

An investigation was undertaken to build the software foundation for the WHERE (Web-based Hyper-text Environment for Requirements Engineering) project. The TCM (Toolkit for Conceptual Modeling) was chosen as the foundation software for the WHERE project which aims to provide an environment for facilitating collaboration among geographically distributed people involved in the Requirements Engineering process. The TCM is a collection of diagram and table editors and has been implemented in the C++ programming language. The C++ implementation of the TCM was translated into Java in order to allow the editors to be used for building various functionality of the WHERE project; the WHERE project intends to use the Web as its communication back-bone. One of the limitations of the translated software (TcmJava), which militated against its use in the WHERE project, was persistent data management mechanisms which it inherited from the original TCM; it was designed to be used in standalone applications. Before TcmJava editors could be used as a part of the multi-user, geographically distributed applications of the WHERE project, a persistent storage mechanism must be built which would allow data communication over the Internet, using the capabilities of the Web. An approach involving features of Java, CORBA (Common Object Request Broker), the Web, a middle-ware (Java Relational Binding (JRB)), and a database server was used to build the persistent data management infrastructure for the WHERE project. The developed infrastructure allows a TcmJava editor to be downloaded and run from a network host by using a JDK 1.1 (Java Developer's Kit) compatible Web-browser. The aforementioned editor establishes connection with a server by using the ORB (Object Request Broker) software and stores/retrieves data in/from the server. The server consists of a CORBA object or objects depending upon whether the data is to be made persistent on a single server or multiple servers. The CORBA object providing the persistent data server is implemented using the Java programming language. It uses the JRB to store/retrieve data in/from a relational database server. The persistent data management system provides transaction and user management facilities which allow multi-user, distributed access to the stored data in a secure manner.

CHAPTER 1: INTRODUCTION

Introduction

The importance of Requirements Engineering in the software development life cycle can never be over-emphasized. Requirements Engineering refers to the earliest phase of the software development cycle when requirements are elicited, defined, and specified. Requirements are the statements of need and are intended to convey understanding about a desired result independent of its actual realization (Kotonya and Sommerville, 1996). The requirements engineering process is aimed at providing a clear, consistent, and precise model and unambiguous statement of the problem to be solved by the software development process. Formulation of requirements may have a substantial impact on the success of a software development project and poorly formulated requirements are known to have resulted in partial success and, in some extreme cases, total abandoning of high budget software development projects (Boem, 1984; Boem, 1987; Kotonya and Sommerville, 1996). The problems of establishing an adequate set of requirements for a software system, often manifest in failure of the system to satisfy customer needs, are many and inadequate communication among requirements engineers is one of them.

Since the requirements engineering process is a human endeavor, the need for communication among the members of the engineering team is natural. This communication need may be tantamount to providing collaboration among geographically distributed people as it is not uncommon for the members of a requirements engineering team to be located at geographically distant locations. The emergence and astonishing success of the World Wide Web offers an opportunity for facilitating aforementioned collaboration among geographically distributed people. Although the Internet has revolutionized the information sharing, the kind of information that can be shared and the security with which it can be shared, is still limited. The requirements engineering process requires sharing of information which could be much more structured and in a complicated form than that could be provided with HTML (Hyper Text Markup Language). For instance, requirements documents may contain textual as well diagrammatic information that may be required to be viewed and modified on-line with restricted access. The plain HTML lacks the capabilities to accomplish this task. Therefore, use of a more powerful technology is required in order to facilitate collaboration among geographically distributed requirements engineers using the World Wide Web.

Java, a programming language designed for the Internet, offers a lot of promise for providing the kind of capabilities required for enabling collaboration among geographically distributed people who need to share and transact on complex information. The capability the java applets provide for sharing complex information over the internet is one of the major reasons it has been selected as the implementation language for the WHERE (Web-base Hypertext Environment for Requirements Engineering) project being undertaken by WVU/NASA SRL (Software Research Laboratory). The WHERE project is concerned with the communication and coordination problems faced on large, geographically distributed requirements engineering projects.

The ultimate goal of the WHERE project is to support the process of collaborative development of requirements specifications with tools to manage incremental changes to large specifications. The project builds upon the earlier work on ViewPoints (Finkelstein et al., 1992). The ViewPoints represent chunks of a specification and each of them has an owner and a representation style. The WHERE project aims to implement the ViewPoints framework and introduce the approach into a real project in order to collect information about the relationships between ViewPoints. At present, the WHERE project implementation is in the initial stage which involves building information representation infrastructure required for building the later parts of the project. The activities to be carried out during WHERE implementation require a set of editors and viewers for representing and modifying various kinds of information required for implementing ViewPoints framework.

A survey of the available software engineering tools revealed that TCM (Toolkit for Conceptual Modeling) project developed at Vrije Universteit fulfilled the information representation needs of the WHERE project. Therefore, TCM was adopted as the foundation for implementation of various tools to be used for building various functionality in the WHERE project. The C++ programming language implementation of the original TCM had to be translated into the Java programming language in order to Web-enable the toolkit. Also, the original TCM has been designed to be used by individual users who do not necessarily need to collaborate on-line from geographically distributed locations.

Therefore, a study is needed to adapt the Java version of TCM (TcmJava) in order to support distributed collaboration using the WWW and proven Web-browser technology. The issues involved include implementing various tools in TcmJava using applets downloadeable over the WWW. Also, the data generated by using these tools need to be stored persistently and securely. Since the Web-browsers impose strict restrictions on Java Applets when it comes to data storage, a mechanism is required to allow the applets save/load data from a persistent data store which may, possibly be, geographically distributed. The persistent data has to be available to the members of the requirements engineering team collaborating on a project. Access restrictions, transaction management, and concurrency control have also to be considered in order to provide meaningful collaboration in real-time. The present investigation was, therefore, undertaken with the following objectives:

- Explore ways to adapt TcmJava to the Internet.
- Investigate mechanisms to facilitate the transfer and persistent storage of data generated by applets implementing various tools in the TcmJava and downloaded over the Internet.
- Investigate various kinds of data storage mechanisms (Files Systems/Relational Databases/Object Databases) for meeting aforementioned data storage needs of the WHERE project.

The rest of this thesis is organized into four chapters which review (Chapter 2) the literature related to the present investigation; describe (Chapter 3) the methodology used during this investigation; describe and discuss (Chapter 4) the results of the investigation; and summarize, and conclude (Chapter 5) pointing out the needs for future work.

CHAPTER 2: LITERATURE REVIEW

Literature Review

This chapter presents a review of the literature related to the present investigation. The information presented here is organized into following topics.

- Requirements Engineering
- ViewPoints Framework
- Computer Supported Collaborative Work
- Persistent Data Storage in Software Development Environments

2.1 Requirements Engineering

Requirements engineering deals with the earliest phase of the software development process where the foundation for a software development project is laid down. It involves the elicitation, definition, and specification of the need the software project is being undertaken to fulfil. Many of the problems of software engineering have been attributed to the difficulties with the requirements specification (Kotonya and Sommerville, 1996). A greater proportion of the errors in a software system occurs during requirements and design phase (64%) rather than during coding phase (34%) (Boem, 1984; Boem, 1987). Moreover, it is more expensive to fix an error made at earlier stages if they are discovered during final stages. A requirements error found at the requirements stage costs only about one-fifth compared with the cost for fixing the same error if it were found after the system is in use. Discrepancies between the capabilities of a delivered system and the needs it intended to fulfil are common and may incur very high costs (Roman, 1985). Findings of a survey on nine software development projects (US Government Accounting Office, 1979) showed that 47 % of the money was spent on the software that was never used. Another 29 % was spent on the software that was never delivered and 19% of the money resulted in software that was either reworked extensively or abandoned after delivery. According to the aforementioned study only 2% of the total money spent resulted in software that completely met its requirements. Therefore, an improvement in methods used in requirements engineering has a potential for tremendously curtailing the software cost.

The requirements engineering process is fraught with difficulties which are often manifest in the failure of software to satisfy the real needs of the customer. Several of these problems are listed by Kotonya and Sommerville in their paper on Requirements Engineering with ViewPoints (Kotonya and Sommerville, 1996). Some of these problems stem from the lack of appropriate tools for supporting the requirements engineering process. There is a need for tools to help the requirements engineers to collect, structure, and formulate requirements in an efficient and consistent manner. Since the requirements process is a human endeavor, the occurrence of communication problems during the process is natural. In large projects, a group of individuals must collaborate in the requirements engineering process that leads to the production of requirements specification, the documentation of the outcome of the requirements elicitation and definition. The requirements are never stable and so the requirements specifications are apt to evolve. Managing evolving requirements specifications is a significant problem because a small change to one part of a specification may impact the whole system specification documentation. These impacts are often hard to reason about and hence it is hard to know that all implications of a change have been taken into account.

Research on Requirements Traceablility (Gotel and Finkelstein, 1994; Gotel and Finkelstein, 1995) have tried to address the problems arising out of evolution of specifications and need for recording information about individual who must communicate in order to carry out the process of software specification. Requirements traceability tools help to alleviate the problems of change management in evolving specifications by recording links between requirements at different levels, between requirements and test cases, design objects, and so on. However, existing traceability tools only record links without any other information about the relationship expressed by the link. Such tools encode a simple process model based on flow down of requirements through different levels. They do not capture any knowledge about the method and notations being used, and hence fail to provide any active support for the development and evolution of specifications. Therefore, a framework is required that will provide tool support for recording more comprehensive information about chunks of evolving specifications and relationships among them. In addition, the framework must support collaborative development among geographically distributed engineers. The following section briefly reviews the literature related to the ViewPoints framework which the WHERE project intends to use as its foundation for providing tool support to facilitate change management in evolving requirements specifications.

2.2 ViewPoints Framework

The ViewPoints framework supports distributed software engineering in which multiple perspectives are maintained separately as distributable objects (Finkelstein et al., 1992). A ViewPoint can be thought of as a combination of the idea of an actor, knowledge source, role, or agent in the development process, and the idea of a view or perspective, which an actor maintains. ViewPoints are loosely coupled, locally managed, coarse-grained objects which encapsulate partial knowledge about the system and domain, and the process of development. The system and domain may have been specified in a particular, suitable representation scheme. The knowledge contained in a ViewPoint is assigned to five different parts of the ViewPoint called slots. A ViewPoint has the following five slots (Nuseibeh et al., 1993; Nuseibeh et al., 1994):

- A representation style which is the scheme and notation used by the ViewPoint to express the knowledge it possesses.
- A domain describing the area of concern addressed by the ViewPoint, with respect to the overall system under development.
- A work plan comprising the set of actions that will be used to build the specification, and a process model to guide application of these actions.
- A specification describing the ViewPoint domain using the notation described in the ViewPoint style and developed using the strategy described in the work plan.
- A work record containing an annotated history of actions performed on the ViewPoint.

Each ViewPoint has an owner who is the development participant associated with the ViewPoint. It is the responsibility of the ViewPoint owner to develop a specification for the ViewPoint using the notation defined in the style slot, following the strategy defined by the work plan, and for a particular problem domain. Various actions and events involved in the ViewPoint are recorded in the work record. The ViewPoints framework deliberately encourages multiple representations and departs from attempts to develop monolithic specification languages. The framework does not make a commitment to a particular software development method. In general, a software development method is composed of various techniques. Each technique has its own notation and associated rules governing when and how to use that notation. The ViewPoints framework presents an opportunity to implement a particular software development method by defining a set of ViewPoint templates. These templates, as a group, describe the set of notations provided by the method, and the rules governing their use as a group or independently of each other.

The ViewPoints framework provides for inconsistency toleration without any requirement for changes to one ViewPoint to be consistent with others (Finkelstein et al., 1994). A set of inter-ViewPoint rules can be defined depending upon the method being used. These rules express the relationships that should hold between particular ViewPoints and are used to perform consistency checking. The consistency may be checked incrementally between ViewPoints at particular stages rather than being enforced at all times. The application of consistency checks is governed by a protocol where the checking process is initiated by either ViewPoint owner. The resolution of inconsistencies is guided by a fine-grained process model in each ViewPoint (Nuseibeh et al., 1993).

Tools support for the ViewPoints framework has been built in the form of a prototype computer-based environment (Nuseibeh and Finkelstein, 1992). The prototype environment provides a ViewPoint Viewer which has two distinct modes of use: 1) method design; 2) method use. Method design involves the creation of ViewPoint templates which are the ViewPoints for which only the representation style and work plan slots have been filled. In method use, ViewPoints are instantiated from the templates created in method design and are used to represent various perspectives. Each ViewPoint instantiated from a particular template inherits the knowledge necessary for building and manipulating a specification in the chosen notation, and cross-checking consistency with other ViewPoints. Therefore, each ViewPoint serves as a self-contained specification development tool.

The ViewPoints framework offers a coherent approach to the management of multiple perspectives. The approach supports multi-language specification, without the requirement for a common data model or language. The framework, therefore, facilitates method integration as well as distributed development. The framework has been used to implement software engineering methods such as CORE (Nuseibeh et al., 1993) and the CDA (Kramer and Finkelstein, 1991). This use of the framework has demonstrated its ability to express relationships between different representation schemes.

Since the ViewPoints framework is designed to support distributed software engineering, an implementation of the framework will involve providing tool support for geographically distributed collaboration. A number of architectures have been developed which aim to provide tool support for collaborative work. The following section presents a brief review of these architectures pointing out their suitability for use in a system intended to provide tool support for facilitating collaboration among individuals involved in requirements specification.

2.3 Computer Supported Collaborative Work

The magnitude of software development projects demands that a team consisting of more than one software engineers work together on a project. This requires collaboration among individuals, in the form of being able to access, view, and modify common information. The need for providing computer support to facilitate effective collaboration has spawned a large volume of research into development of tools aimed at making the collaborative activity less costly and less time consuming (Bentley et al., 1997; Callahan and Ramakrishnan, 1996; Dix, 1996; Johnson, 1996; Toye et al., 1994). The World Wide Web (WWW) has become a potent platform for collaborative work. The Collaborative Software Development Laboratory (CSDL) has been doing research in development of tools for facilitating collaboration during various phases of software development process. The World Wide Web Consortium (w3c) has also organized a number of symposia since 1995 aimed at identifying extensions to web technology which would facilitate wide-area asynchronous collaboration. Research has also been done on developing tools for providing collaboration during design phase of the software development process (Emmerich and Schafer, 1996). Although the aforementioned research has resulted in the development of tools/technology that claim to facilitate collaboration during various phases of software development, these tools either lack the capability to provide persistent storage of and controlled access to complex information or fail to support collaboration among geographically distributed individuals. The following section will briefly review some of these technologies.

The WWW offers a globally accessible, platform independent infrastructure and being increasingly looked upon as a potential platform for richer cooperative work (Dix, 1996). However, the web was designed principally as a mechanism for information access and its use for richer forms of collaborative activity may not be obvious. There are architectural issues involved in the use of WWW for cooperation and the most obvious one is the possible extensions and/or modifications to the parts of web to adapt it for cooperative work. There are three parts of the Web which may be extended or modified to infuse cooperative work capability into it; server, client, and protocol. A number of systems have tried to use sever-end extensions to facilitate cooperation. These systems have typically used CGI scripts and independently running servers. Most notable among these aforementioned systems are BSCW (Bentley et al., 1996) and futplex (Holtman 1996). Another possible extension is the use of client helpers and applets. The incorporation of Java and Java-script into web-browsers has emphasized the value of client-end computing, especially for rapid user interface feedback. Some systems have also made use of downloaded helper applications and modified clients to run Tcl/Tk as a client-side script language (van Welie and Elins, 1996). There are other proposed extensions/modifications which are not particularly relevant to the present investigation but are discussed in an excellent report by Dix (1996).

In the Web, the information is usually represented as web pages or electronic documents without any facilities for direct communication, as such. Therefore, a number of applications intended to facilitate collaborative work supply direct communication facilities which could be either synchronous or asynchronous. The synchronous communication facilities include applications such as HushTalk (van Welie et al., 1996) supplying talk–style facilities. Asynchronous communication is primarily supplied by transforming communication into information structure which can be accessed and replied to by multiple users. Providing asynchronous communication requires less deviation from the Web model than providing synchronous communication which effectively bypasses the web protocols entirely. Therefore, the use of asynchronous communication facilities is likely to be easier to provide and present a greater support for using the exiting capabilities of the Web.

The WWW has a number of distinct advantages as the basis for tools to support collaborative information sharing, the most important being the availability of proven technology in the form of Web-browsers (Bentley et al., 1997). The Web-browsers are available for all popular computing platforms and operating systems and provide access to information in a platform independent manner. They offer a simple user interface and consistent information presentation across platforms. Although WWW is an excellent platform for geographically distributed collaborative work, it is limited by its inability to store state information, represent complex information, support multiple authoring, and provide concurrency control. Providing tool support for geographically distributed collaboration in a system like requirements engineering requires the support system to be very flexible in the kind of information that may be represented, persistently stored, and concurrently accessed in a controlled manner. This may require making client as well as server side extensions to the existing WWW infrastructure if the well developed and proven Web technology has to be taken advantage of in providing geographically distributed collaboration. The following section will further discuss the attempts at extending the WWW in order to facilitate a richer collaboration.

The 'Egret' system developed by the Collaborative Software Development Laboratory (CSDL) at University of Hawaii implements a multi-client, multi-server, and multi-agent architecture (Johnson, 1995). Egret provides both low and high level storage and communication facilities for the development of cooperative work applications. Data ranging from unstructured binary storage, to schema-based and structured storage records, to HTML-compatible hypertext may be represented. The architecture uses indexing and local replication mechanisms to enable efficient "relational-style" queries over the underlying network database. Inter-process communication is implemented via

TCP/IP sockets, and provides a variety of programmatic and interactive client communication facilities. Password mechanisms are provided to facilitate secure collaboration in groups dispersed across the internet. The architecture has been used to develop applications providing tool support for software review and quality improvement (Johnson, 1994), collaborative authoring and learning (Johnson and Moore, 1995), and collaborative learning and review (Wan and Johnson, 1994). The architecture, however, supports collaboration involving only textual information and fails to make use of the proven technology in the form of the capabilities of the WWW. Therefore, this system is not suitable, as such, for providing geographically distributed collaboration in a system like requirements engineering where cooperation involving textual as well as non-textual information must be supported.

The BSCW (Basic Support for Cooperative Work) system developed at German National Research Center for Information Technology provides basic features for cooperation in an integrated service, accessible from different computing platforms (Bentley et al., 1997). This system makes no demands on users to adopt new word processing, spreadsheet, or other application software. Moreover, the system does utilize the capabilities of the Web and in fact is an extension of a standard Web server through the server CGI Application Programming Interface. A 'BSCW server' (Web server with the BSCW extension) manages a number of shared workspaces. These workspaces are repositories for shared information, accessible to members of a group using a simple user name and password scheme. A shared workspace can contain different kinds of information such as documents, pictures, URL links to other Web pages or FTP sites, threaded discussions, member contact information etc. Facilities are provided for saving information from client machines and also loading information to client machines from the BSCW server. The BSCW system supports upload of multiple types of documents, automatically detecting the document type and providing full feedback on the progress of the document transmission to the BSCW server. The event service that is built into the system provides users with information on the activities of other users, with respect to the objects with in a shared workspace. The system also provides controlled sharing and member administration capabilities. The system has been designed to provide basic features for supporting cooperative work for widely-dispersed working groups, independent of their computing, network, and application infrastructures. The system supports collaboration involving documents with textual as well as pictorial contents. The system is, however, not suitable for use in a requirements engineering environment which generally requires the capabilities to generate, persistently store, and modify a wide variety of information. Moreover, the requirements for information representation may change overtime requiring the capability to be able to generate and manage new kinds of information which is possible only with support of full-fledged programming language.

The SHARE project being undertaken at Stanford University intends to provide a methodology and environment for collaborative product development (Toye et al., 1995). Their domain is to facilitate collaboration among engineers involved in design and production. The SHARE architecture consists of a set of agents interacting as peers over the Internet. Each agent can represent one or more of the following: a designer and his personal CAD tools, a database or other information service, a computational service that supports engineering or the engineering process. The agents exchange information and services using a simple command language (Finin et al., 1992) and representation of multimedia information (Bornstein and Freed, 1992). The messages are sent using standard e-mail and TCP/IP transport services. The architecture uses e-mail as the primary medium for both human communication and tool integration. The rationale behind using e-mail in this project is pervasiveness of e-mail and its familiarity to large number of designers. This project is geared specifically towards providing tool support for collaboration among engineers and is not suitable for use in a software engineering environment where the requirements for collaboration are different.

The WISE (Web Integrated Software Environment) system developed by SRL (Software Research Laboratory) at West Virginia University, makes use of existing Web technology to support measurement of change activity as an implicit part of the software process (Callahan and Ramakrishnan, 1996). The WISE provides a forms based, work-flow management system that helps members of a software development team overcome geographical barriers to collaboration. Development of the WISE system is an excellent example of using the existing proven technology to provide tool support for collaboration in software engineering process. The WISE system has been designed to provide tool support for software project management and process measurement and can be used in conjunction with tool support for collaboration in various phases of software development.

Various architectures have been developed to provide tool support for collaborative work. Some of them make use of existing and proven technologies that minimizes the effort to provide the initial infrastructure on which to build the more specialized frameworks geared towards supporting the collaborative work in specific domains. Various systems that are currently available to provide tool support for collaborative work are either limited in the kind of information that can be represented or are designed to serve the collaborative needs of the people engaged in work in specific domains. None of the currently available systems provide the kind of tool support required to facilitate collaborative work in software requirements specification which requires representation and sharing of information more complex than textual and statically pictorial information contained in documents supported by currently available systems. The WHERE project which aims to use the ViewPoints framework for managing evolving requirements specifications requires an infrastructure providing tool support to represent various kinds of diagrammatic and textual information encountered during requirements specification process.

As pointed out in the discussion above, one of the limitations of the current architectures for collaborative work is the kind of information that can be maintained persistently, accessed concurrently in a consistent and controlled manner, and modified while maintaining the integrity of the persistent store. Tool support for a collaborative work environment, being built on a framework like ViewPoints designed to support distributed collaboration, needs to provide persistent storage of data with aforementioned constraints. Therefore, providing persistent storage facilities for the WHERE project is important and formative part of the implementation. Before the implementation could proceed further, availability of a persistent storage infrastructure is exigent. The mechanisms used to manage the persistent data may have a significant impact on the way some of the tools supporting the subsequent functionality of the project are built. The following section reviews the issues involved in persistent data storage in software engineering environments and its relationship to the persistent data storage needs of the WHERE project.

2.4 Persistent Data Storage in Software Development Environments

Software Development Environments (SDEs) include tools intended to support one or more of the software life-cycle phases (Emmerich et al., 1993). This often involves construction and analysis of documents and document interdependencies. The value of an SDE is judged by its ability to enable incremental, intertwined, and syntax-directed development of documents. Good SDEs also provide for maintenance of these documents, tracing back of errors through different documents, and change propagation through document boundaries to correct errors (Engels et al., 1992; Habermann and Notkin, 1986). These environments are also expected to provide multi-user and often geographically distributed support. They should have flexible and adaptable mechanisms to facilitate controlled sharing of information by a number of users. These environments usually require the storage/retrieval of large number of objects and relations among them at different levels of granularity. Moreover, these objects must be manipulated under the control of an advanced transaction mechanism. These considerations emphasize the importance of storage/retrieval mechanism underlying an SDE.

It has been argued that dedicated database systems that are specialized with respect to functionality and implementation are necessary for use in software engineering (Bernstein, 1987). The functionality and efficiency of purely relational database management systems is considered inadequate to satisfy the needs of software engineering tools and environments (Taylor et al., 1988). The computer science community has seen the development of a number of systems which radically differ from standard relational technology. Despite the substantial number of these new database systems, a suitable database system for SDEs still does not exist (Emmerich et al., 1993).

A process-centered environment (PSDE) is a software development environment in which providing multi-user support is based on a well-defined development process. A database for software engineering should provide: 1) efficient manipulation of the document representation defined by the software development process; 2) advanced transaction mechanisms on the stored structures to enable sophisticated collaborative support. In many of the existing collaborative development environments the documents are handled as monolithic blocks. This representation militates against the attempts for providing inter-document consistency checking and preservation. Therefore, the need of support for incremental, intertwined development and maintenance of software is not served. The lack of appropriate persistent data storage and retrieval mechanisms is thought to be responsible for the lack of appropriate functionality in the currently available SDEs.

Architecturally, PSDE consists of three main components:

- A well-defined process engine to coordinate the work of developers involved in a project.
- A set of integrated, syntax-directed tools for allowing the developers to conveniently manipulate and analyze documents without compromising consistency between related documents of different types.
- An underlying database for software engineering (DBSE) which is capable of storing project information and documents.

The first two of the requirements outlined above are being addressed in separate investigations being undertaken as parts of the WHERE project. Since the present investigation intends to address the third requirement, review here will concentrate on the storage/retrieval mechanism.

The common internal representation for syntax-directed tools such as syntax directed editors, analyzers, pretty-printers and compilers is a syntax-tree of some form (Emmerich et al., 1993). Usually this abstract syntax-tree representation of documents is generalized to an abstract syntax-graph representation for reasons such as efficient execution of documents, consistency preservation by tools, and user-defined relations within documents. Use of this approach alone may be inefficient for operations such as consistency. Therefore, the researchers have developed techniques based on the introduction of additional, non-syntactic paths for more direct attribute propagation (Hoover, 1987; Johnson and Fisher, 1982). These non-syntactic paths are examples of context-sensitive relationships which connect syntactically disjoint parts of a document and may be used in both consistency checking and change propagation.

Requirements of persistence and integrity necessitate that a persistent representation of each document under manipulation must be updated as user-action is finished. Usually a user-action affects only a very small portion of the document concerned. The updates resulting from these user-actions may become inefficient if a complex transformation between the active and persistent representations of a document is required; and the update process involves unnecessary rewriting of parts of the document not being modified. To avoid such inefficiency, the underlying storage/retrieval mechanism must support the definition, access, and incremental update of the stored structures with facilities for efficient traversal. To preserve the integrity of stored structures, support for atomic transactions is necessary.

Context-sensitive and user-defined relations between document components (ViewPoints) necessitate incorporation of some kind of structure in the persistent stored documents. The aforementioned relations may not be confined to within individual documents and may exist between components of distinct documents. Consistent handling of these inter-document relationships requires that the set of documents making up a project must be represented in the form of a single structure. Therefore, the underlying storage/retrieval mechanism must be amenable to the kind of representations discussed above.

The kind of structures required to represent a project and attribute information associated with it cannot be determined by the storage/retrieval mechanism. However, once these structures have been well-defined, the storage/retrieval mechanism must be able to define and control the internal storage for those structures. The underlying database system, therefore, should have capabilities to store and control the kind of structures (possibly object-oriented) used in software engineering projects. Incremental changes to these structures should be supported by the underlying system. The underlying system should provide the facilities for implementing the operations performed by tools in terms of modifying the overall structure stored in the database system. This is necessary because of two reasons:

- The structure used to represent the whole project should be encapsulated with operations that preserve the structure's integrity and provide a well-defined interface for accessing and modifying it.
- If the access and modification operations are performed within the storage/retrieval mechanism, they are more efficient than performing them within tools as need for transferring unnecessary data over the network is greatly reduced in the former case.

In order to be able to perform the modification operations within the storage/retrieval system, it must be powerful enough to express various kinds of relationship which are to be manipulated in a modification operation. In addition, the process defined for a development project may specify a reasoning component enabling the users to perform various kinds queries on the stored data. The system must be able to provide support for performing those queries. A typical query may be to show a list of all the documents owned by a particular developer. In addition, the queries may be designed to assess the overall state of the project. The state assessing criteria, of course, will have to be defined by the process. The storage/retrieval mechanism must be able to support it transparently without any need to transfer large amounts of data over the Internet; the latter can be very inefficient. It may be desirable for the storage/retrieval system to have the capability to support queries which are not known a priori and may become necessary as the development of the project progresses.

Given the evolutionary nature of the requirements specification process, the process governing the development of a particular project may need to be changed as increasing amount of knowledge is obtained about the system. For instance, the stored data will represent various entities in the system under development and there will be relationship defined among those entities. Those relationships may change or some entirely new relationships may have to be defined during the evolution of the project. This means that the system must be capable of allowing the modification of existing relationships and definition of new relationships among the stored data. In addition, it is desirable to have the capability of being able to modify or extend the structures representing entities in the system as the need for adding additional attributes to an entity may arise during project evolution.

Since the storage/retrieval system is intended to maintain information about the entire project and different components being developed concurrently but independently may be at different stages of development, support of revisions and versioning is highly desirable and must be provided in a good database system for software engineering. The system must provide facilities for maintaining version histories of various components and sub-components of the project.

Providing multi-user support in an SDE necessitates the definition of access rights for particular documents and their components. Also, the transaction mechanisms are required to control and enable concurrent multi-user access to shared information. The storage/retrieval mechanism must provide mechanisms to identify individual users as well as user groups. The capabilities should be provided to define and modify the ownership of stored objects representing components of the project and their further sub-components. The information about a particular component of the project may need to be accessed by members of more than one groups; the storage/retrieval mechanism must support definition of multi-group access rights. The access rights may need to be modified at any time and such a capability must be supported by the underlying mechanism. In addition, the definition or modification of access rights does not mean anything unless enforced by the system.

The PSDEs require storage/retrieval systems having transaction mechanisms which are much more sophisticated than the conventional transaction mechanisms (Emmerich et al., 1993). The conventional mechanisms could result in rollback which deletes the effect of a possibly long-lasting developer effort, or they could block the execution of certain activity for days or even weeks. Such properties are too restrictive for a PSDE and result in situations which are intolerable. These shortcomings of the conventional mechanisms have been realized and advanced transaction mechanisms such as split/join transactions (Pu et al., 1989) and cooperating transactions (Nodine et al., 1991) have been developed. These mechanisms have tried to achieve the desired result by relaxing one or more properties of atomicity, consistency preservation, isolation, and durability which characterize the conventional mechanisms. A detailed overview and critical evaluation of those advanced mechanisms is given in Barghouti and Kaiser (1991).

Emmerich et al. (1993) argue that none of these advanced mechanisms is powerful enough to serve as **the** transaction mechanism for a database of a PSDE. They quote Peuschel et al. (1992) to point out that only the process engine, which knows the current state of an ongoing project, can decide whether and when to request a lock for a particular sub-graph and how to react in case of inability to acquire the lock. The process engine also defines whether a transaction is executed in isolation or in a non-serializable mode.

The preceding sections have briefly pointed out the expectations of a storage/retrieval mechanism underlying an SDE. In this section, we will briefly look at some of the available technologies. The relational DBMS, as such, are inappropriate for meeting the persistent data storage needs of SDEs (Emmerich et al., 1993) because of three reasons: 1) The data model of RDBMSs cannot appropriately express the structures required to store project information of an SDE; 2) They do not support versioning at a level that may be required in an SDE; 3) They do not allow the implementations of customized transaction schemes. Emmerich et al. (1992) gives a more detailed discussion and reasoning about the unsuitability of RDBMSs for use in SDEs. The ooDBMSs (Object-Oriented DBMSs) provide a natural way of meeting the requirements of client/server systems and systems whose data is more complex than that can be lined up in relational tables (Orfali et al., 1996). The ooDBMSs have an advantage over RDBMSs in that they know the overall structure of complex objects and sometimes their behavior as well. However, the ooDBMSs are still under development with respect to functionality as well as standardization. Pure ooDBMSs still lack functionality in areas of complex search, query optimizers, and server scalability. Orfali et al. (1996) predict that with the efforts of standardization (ODMG-93 is an example) going on for ooDBMSs and the promotion of standardization of these systems with in the CORBA ORB committee, these systems will be the successor to RDBMSs.

The emergence of new software technologies such as the Java Programming Environment and CORBA implementations and ever-increasing popularity and usefulness of the WWW may have changed the way people think of multi-user distributed applications which may involve extensive database access. In addition, a combination of these technologies along with emerging database technology may have bridged the gap between the functionality provideable by RDBMs and the persistent storage/retrieval needs of an SDE. The present investigation intends to explore the use of these software technologies to provide the persistent data storage needs of an SDE (WHERE project).

CHAPTER 3: METHODOLOGY

METHODOLOGY

This chapter describes the methods and technologies used and evaluated during this investigation. The chapter also gives details of the implementation done to achieve the objectives of the proposed research.

This investigation was conducted as a part of the WHERE project being undertaken by SRL. Since the WHERE project intends to provide tool support for collaborative development of requirements specifications, a foundation was required on which to build the tools required to meet the specific needs of the project. There were two options available to the WHERE project team: 1) build all the tools from scratch; 2) adopt some already existing toolset as foundation and build on top of it. Since building tools is a time-consuming process and a lot of effort may be expended on building tools from scratch, the WHERE team decided in the favor of the second option provided a suitable toolset already existed. A survey of various tools intended to provide tool support for software engineering environments was done and the TCM (Toolkit for Conceptual Modeling) project being undertaken at vrije Universteit seemed to be a good foundation for the tools to be built during the WHERE project. The following section gives a brief description of the TCM software which will be followed a description of the functionality that was required in the WHERE project but was not provided by the TCM software.

3.1 TCM Software

The TCM project was undertaken with an aim to produce software support for software requirements and design engineering. The software delivered by the project can be used to represent various kinds of information during requirements and design phases of software development process. The functionality to be provided by TCM includes: various graphical editors to provide visual representation of different, mutually consistent views of product requirements and product designs; tool support for graphical simulation of the specified product; and support for generation of prototype code for the product. All representations of the product produced using TCM software are conceptual, meaning the representations are meant to externalize conceptualizations of the software product. Various requirements and design engineering methods supported by the TCM are described in Wieringa (1996).

The current version of the TCM is a collection of graphical editors for a range of graphical notation systems used in requirements and design engineering methods. The TCM runs on Unix systems with X-Windows. The graphical editors constituting the TCM can be used for editing several kinds of documents including diagrams, tables, and trees. The editors are available for following kinds of documents:

- *Diagrams*: Generic Graph Diagrams, Entity-Relationship Diagrams, Class Relationship Diagrams, State Transition Diagrams, Recursive Process Graphs, Data and Event Flow Diagrams, and JSD Process Structure and System Network diagrams.
- *Tables*: Generic Tables, Transaction Decomposition Tables, Transaction Use Tables, and Function Entity Type Tables.
- *Trees*: Generic Textual Trees, and Function Refinement Trees.

Various editors share a mostly common user-interface which has been designed to be user-friendly and usable without any further help. Limited on-line help is provided. The current version supports constraint checking for single documents (e.g. name duplication, cycles in an is-a relationship). The TCM distinguishes built-in constraints (of which a violation cannot even be attempted) from immediate constraints (of which an attempted violation is immediately prevented) and soft constraints (for which the editor issues a warning if a violation occurs during drawing). The current version of TCM does not yet support constraint checking across documents which is required for integrated conceptual modeling. The implementers of the TCM are planning to enhance it with cross-diagram checking functions. More information about TCM, its detailed design, source code, a running version with all the necessary documentation can be obtained by contacting the TCM developers at tcm@cs.vu.nl.

The aforementioned features make TCM a suitable foundation on which to build various kinds of functionality required to implement the ViewPoints framework, the conceptual framework underlying the WHERE project. Since the WHERE project aims to support collaboration among geographically distributed people and TCM is not designed for that purpose, the TCM could not be used as such to meet the needs of the project. In addition, we wanted to take advantage of the Web as the communication back-bone because of various kinds of proven technology it offers in the form of communication protocols and Web-browsers. But to be able take advantage of Web technologies, a complementary technology was required which would allow using the power of Web technology to serve the communication needs of our environment. The Java Programming Environment seemed to fit the profile of that complementary technology we were looking for. Therefore, we decided to translate the TCM software into Java Programming Language. The translation process was started in August of 1996 and the first version of the TcmJava (The Java version of the TCM software) was released in the summer of 1997. The following section briefly describes the Java Programming Environment followed by a brief description of the C++ to Java translation of the TCM software.

3.2 Java Programming Environment

Java is an object-oriented programming language which is relatively new and have been gaining increasing popularity among the software developers especially those involved in using the Web. Java has many interesting features, two of which are of particular importance for use in distributed applications that want to take advantage of the proven Web technology. First, Java source code can be compiled into a format which is independent of any particular machine architecture. This format consists of virtual machine instructions and symbolic data and is called byte-code format. Execution of this bytecode requires Java Runtime Environment (JRE) which contains a special program called Java Interpreter. The Java Interpreter knows the meaning of the bytecodes and can execute Java bytecode irrespective of the underlying machine architecture. The Interpreter itself, however, needs to be ported to a particular platform on which Java programs have to run. The Interpreter knows how to convert the bytecodes to the memory addresses and machine-instructions of the underlying architecture. The Java approach is a trade-off between speed and portability. Execution of the bytecode is slower that of the compiled code but the bytecodes are completely independent (at least in theory) of the architecture of the underlying system. Consequently, if the JRE is available for a platform, the Java bytecode will run on it irrespective of the architecture. Execution of the Java bytecode is faster than the fully interpreted code. Therefore, Java approach is a judicious compromise between portability and speed. The second feature is not a feature of the Java Language itself but allows the Java applets to complement and enhance the capabilities of the Web. It is the APPLET tag of the HTML (Hyper-Text Markup Language). The APPLET tag provides the information which enables a Web-browser to find and execute the applets. An applet is a Java program which requires a Java-enabled Web-browser or an appletviewer to run. In order to be able to execute a Java applet the Web-browser must have been extended to incorporate the JRE. If the Web-browser has this capability, it can then automatically download an applet to the user's host machine and execute it there. This feature provides a powerful mechanism for transferring the executable code over the Internet which can be used to extend client-side capabilities of a Web-browser greatly. The executable code in the form of downloadable Java applets can be provided to the user when it is actually needed which obviates the need for installing it at the client side ahead of time. Therefore, a combination of Web and Java technology provides a powerful infrastructure that may greatly simplify the communication complexities involved in developing and deploying a client server environment intended to provide collaboration among geographically distributed people. More details about Java and its relationship to Web can be obtained from Hamilton (1996) and Yourden (1996). These considerations motivated the translation of C++ code of TCM into Java. The following section describes the translation process.

3.3 C++ to Java Translation of TCM

Since both C++ and Java are object-oriented languages, translation process for most part was straightforward. However, there are some features where the two languages markedly differ and there is no direct mapping from one language to the other. For most part, the translation process involved making some syntactic changes to the C++ code in order to convert it into Java code. The parts of the translation process involving significant changes between C++ and Java code are described below.

3.3.1 Templates

The templates in C++ has no direct equivalent in Java. The TCM code has a *List* class which implements a ordered collection of a generic type and provides methods for performing various operations on the collection. This class is extensively used throughout the TCM code and is implemented using templates. In TcmJava, this class has been implemented by subclassing the *Vector* class included in the standard Java library provided with JDK (Java Developer's Kit). All the operations of the C++ *List* class could be mapped to the Java version with variations in some cases. Because Java language did not support defining operators at the time of translation, copy (==) and other operators defined in the class, could not be directly mapped. However, equivalent operations could be provided. The generic parameter in the template was mapped to Java *Object* which is the type stored in the Java *Vector* class. The translated *List* class could be used naturally during the rest of the translation process.

3.3.2 Strings

The C++ programming language does not have a standard *String* class. The TCM code has defined a *String* class which provides a convenient way to manipulate a collection of characters. This class is also extensively used throughout the TCM code. Although the functionality provided by this class can be derived from the Java *String* class, the mapping between the two is not natural. Therefore, we defined a new *string* class in TcmJava. This class uses the standard *String* class to implement various operations declared by the TCM string class. Implementation of this new *string* class simplified the translation of the code involving the use of the *string* (C++) class. Although we could have used the standard Java *String* class as such, it would have made the translation process more difficult and error prone because it would require keeping track of the mapping which was convoluted in some cases. Although implementing a new *string* class involved work and some performance overhead, it simplified the translation process making it straightforward.

3.3.3 Parameter Passing

The C++ allows passing of primitive data types by reference. This can be done by using pointers or by using the C++ reference operator. Java did not have a built-in mechanism for passing primitive by reference at the time of translation. Therefore, we had to implement wrapper classes for various primitive types (int, float, long etc.) in order to allow them to be passed by reference.

In Java, the reference types are also passed by value whereas in C++ these may be passed by reference by using double pointers. Some of the TCM code uses passing by reference semantics while passing reference type parameter. The double pointers in the Java code were simulated by passing arrays of reference types which behave exactly like double pointers in C++. The passing of the *String* variables by reference was achieved by using the *StringBuffer* class. In C++ the strings can be passed by reference by passing a pointer to a *char*. Since Java does not have pointers and Java strings are static, one must use *StringBuffer* in order to allow the *String* variables to be manipulated in the called methods.

3.3.4 GUI Code

The C++ and Java versions of the TCM code have very little similarity in the Graphical User Interface (GUI) code. The C++ code uses X/Motif libraries and callback mechanism to implement various GUI components. There is no facility in Java to use these mechanisms. Therefore, a large portion of the GUI have to be redesigned and re-implemented in the TcmJava. Java's event model markedly differs from C++'s callback mechanism. This is the part of the TCM which required redesigning in order to implement it in Java. However, we were able to duplicate most of the X/Motif functionality of the original TCM by using various GUI components provided in the standard Java AWT package. A significant amount of effort was, however, expended on implementing an image button which is not a part of the standard Java. The standard

Java Button class included in the AWT package provides a very limited functionality in terms of the kind of information it can present to the user. In fact, only textual information can be presented. The GUI code of TCM makes extensive use of image buttons to present the information about various drawing functions to the user in an intuitive manner. To duplicate this in Java, we had to design and implement (in some cases using freely available existing Java code) a number of classes that handle displaying of image buttons and handling of events resulting from user interaction with those images buttons.

Another part of the user interface where standard Java functionality was really limited to serve the needs of TcmJava is the drawing capabilities provided by the *AWT* in the form of standard *Graphics* class. This class provides no direct methods for drawing of a number of shapes that the TCM uses to represent various kinds of information. Most notable ones are drawing of various kinds of line patterns (dashed, stippled etc.) while drawing various shapes. We had to implement new Java classes to duplicate this functionality of the TCM in TcmJava.

Although the event model used in TCM and the Java event model are fundamentally different, translation of the event handling code did not present many problems. This is because the event model used in Java is more systematic and simplified than using X/Motif callbacks which is messy and error-prone. Although the parts of the TCM implementation responsible for event handling had be completely redesigned and re-implemented, duplicating the functionality of the original code probably took lesser time and effort than it would if it were translated. This is because of better and more systematic event handling mechanisms provided by the Java language.

3.3.5 Storage/Retrieval

The mechanisms used for persistent data storage/retrieval in the original TCM are probably the only parts of the TCM implementation which are less than impressive. This is a direct result of the limitations of the C++ programming language to provide suitable mechanisms on which to build. This is where the perfectly object-oriented design of the TCM falls apart a little bit. The TCM maintains the persistent data by storing it in Unix files which contain textual information in a predefined file format defined by the implementers of the TCM. The details about the TCM file format are given in the TCM design document. The object references are converted into ASCII characters and stored in the files. While reading the file, the reference information stored in ASCII format is converted to an object reference by converting it into an integral type (primitive type 'long' in C++) and casting it to an object pointer. Storing of other information also involves disassembling of the information represented in an object and storing the resulting pieces of information in ASCII format. These pieces of information stored in ASCII format must be put back together and the objects reconstructed during retrieval of a stored document. After reconstructing the objects and obtaining the reference information, the object references are reset to construct the original structure of the document in memory as it were before it was stored. This approach has many limitations. First, it destroys the modularity of the design and implementation because the code responsible for storage/retrieval must be scattered among various classes which otherwise have nothing to do with persistent data storage. Second, a lot of extra information must be written to files in order to allow the system to interpret the information correctly at retrieval time. This information has no relevance to the conceptual information being stored but has to be there to overcome the limitations of the approach used. Since there is extra stored information, reading this information and using it to interpret the actual information involves extra work which is an unwanted overhead. Third, the file format defined to store the information is apt to change over time as the project evolves to incorporate increasing functionality because of the need to store additional information. This introduces the burden of keeping the newer versions of the TCM software compatible with the documents generated and persistently stored using the older version. The TCM software already have at least two different file formats which will multiply further simply to meet the evolving persistent storage needs of the software as it itself evolves. Although the evolution of the file format is natural, the mechanism used to manage it in current versions of the TCM software is cumbersome. This is because there is lot of overhead involved in ensuring compatibility among various formats each of which has extra stored information which must be interpreted differently depending upon the format. These limitations of the storage/retrieval mechanism used in the TCM forced us to look for alternative mechanisms which will be described in the following sections.

One of our goals during the translation process was to stay as close to the original implementation as possible in order to be able to incorporate the enhancements to the original software into the translated software. Therefore, we tried to adapt the storage/retrieval mechanism of the TCM to Java. It simply did not work. There is no way in Java to cast an object reference to an integral type so that it can be stored in ASCII format as it is done in the TCM. Java does allow converting the object references into ASCII format. However, if we used it that way, it would mess-up the C++ to Java mapping which will be difficult to manage especially because the storage/retrieval code is scattered and involves lot of checking. Therefore, we decided to use the Java language mechanisms which are more systematic, object-oriented, and intuitive. To do so, we had to drift away from the storage/retrieval philosophy of the original TCM and use the mechanisms built in the Java language to do the checking while loading the stored information.

The Java Programming Environment introduced the concept of Object Serialization into the second major version the language (JDK 1.1). The object Serialization extends the core Java Input/Output classes with support for objects. Object Serialization supports the encoding of objects and the objects transitively reachable from them, into a stream of bytes; and it supports the complementary reconstruction of the object graph from the stream. Serialization can be used for lightweight persistence and for communication via sockets or Remote Method Invocation (RMI). The default encoding of objects protects private and transient data, and supports the evolution of the classes. A class may implement its own external encoding and is then solely responsible for the external format. Originally, we tried to complement the TCM approach with the Java Object Serialization by storing the objects as such, instead of first converting them into ASCII and them storing them. The design of the storage/retrieval mechanism was, however, kept the same as that of the TCM. This approach worked well for stand-alone editors maintaining storing the persistent data in the local Unix or DOS files. However, it involved extra and unnecessary work which could later be dispensed with by using a pure Java and totally different approach. Moreover, since our major objective in the implementation of the WHERE project is to provide tool support for collaboration among distributed people, this hybrid approach using a combination of original TCM mechanism and Java Object Serialization did not meet our needs. Devising a suitable mechanism that would facilitate the persistent data storage by TcmJava editors to a remote server was one of the objectives of this investigation. A detailed description of the mechanism developed to achieve this objective will be given in a later section of this chapter.

3.3.6 Other Miscellaneous Issues

This section describes some other issues involved in the translation process, none of which warrants a category of its own. The virtual methods in C++ mapped naturally to normal non-static, non-final Java methods because by definition these methods in Java are virtual. The concept of "friendliness" in C++ is handled by the Java language by using the *package* concept and various access modifiers. The enumerations in C++ were mapped to final classes in Java. Such a Java class has a final field corresponding to each member of the enumeration. The class also provides methods for constructing the enumerations, and accessing and setting the values of the their members, in order to prevent the assignment of illegal values by the user. The C++ global variables and constants are mapped to *static* fields and *static final* fields of the corresponding classes. In cases where C++ code defines globals outside any class definition (for instance in header files), new *final* Java classes were implemented to map those variables.

The preceding sections have briefly described the translation process pinpointing the parts of the process involving significant effort. One of these parts is the code responsible for persistent storage/retrieval of data generated by various TcmJava editors. The section on storage/retrieval pointed out the limitations of the TCM approach and described the hybrid approach used to provide a storage/retrieval mechanism for the TcmJava version using local Unix/DOS file system for persistent data storage. The following sections will describe the approach developed in this investigation to allow persistent storage/retrieval of the data generated by TcmJava editors downloaded over the Internet and storing/retrieving data in/from a remote server.

3.4 Java, CORBA, Web, and Database Server Approach

Since the WHERE project aims to provide tool support for collaborative development, we adopted Java Programming Environment for implementing the functionality. Reasons for selecting Java were discussed in an earlier section. Having selected Java, we needed a mechanism which would be used for persistent storage of data generated by *applets* launched from distributed locations using a Web-browser. The 1.0 release of Java, the one available at the time of planning of this investigation, did not provide any mechanism for using remote methods by distributed application components. Therefore, we needed some mechanism for allowing the TcmJava editors, launched as applets from distributed locations, to store/retrieve data in/from TcmJava database server/servers. The features of CORBA seemed to complement Web-based TcmJava applets well for serving the persistent data management needs of our environment. Hence, we decided to use CORBA ORB (Object Request Broker) technology (and possibly other CORBA services) to implement a persistent data storage infrastructure. This infrastructure would serve as a base for building additional functionality required in the WHERE project as the process underlying the project is defined. For instance, one of the functionality could be to build the inter-document consistency checking mechanism into the storage/retrieval mechanism. The following section will give a brief overview of the CORBA before proceeding further with the details of the approach.

3.4.1 Overview of CORBA

The CORBA (Common Object Request Broker Architecture) is an industry standard developed and promoted by the Object Management Group (OMG), an industry standards organization. The CORBA specifies rules for communication among object-based, distributed applications in a platform independent manner. One of the core functionality specified by CORBA is the ORB (Object Request Broker) which is a standard mechanism enabling the distributed software objects and their clients to interact.

An ORB is the hub of the communication facilities provided by a CORBA implementation. The ORB provides the communication mechanisms needed by objects and their clients to communicate with each other. Using an ORB, an object and its clients may be present in the same process, or in different processes. The processes may be executing on different hosts connected by a network. The operations that a client may invoke upon an object are specified using a declarative language. This language is a part of the CORBA specification and is called Interface Definition Language (IDL). The clients invoke the services of an object by invoking the operations specified in IDL and objects provide these services by implementing those operations. The objects and their clients may be implemented independently of each other and in different programming languages. The operation requests specified in IDL are conveyed from client to object by the ORB software in a transparent manner. The ORB software is also responsible for conveying the responses from the objects back to their clients. Usually, the implementers of the ORB software provide an IDL compiler which generates the source code for some of the software necessary for allowing the objects and their clients to communicate. The IDL compiler takes operation definitions specified in IDL as input and generates the necessary source code. The generated source code consists of two parts: 1) the part which is compiled and linked with the code providing implementation for an operation; 2) the part which is compiled and linked with the code which intends to use an operation in order to use a service provided by an object implementing that operation. The IDL mappings are available for various programming languages such as C, C++, Smalltalk, and Java. These mappings provide language mechanisms which can be used by the programs written in the language of a particular mapping to invoke the CORBA services specified in IDL and converted to that language by the IDL-to-language compiler.

Since the details involved in passing of information between the objects and their clients are implemented by the ORB software and are transparent to the object as well as its clients, the objects and clients do not have to know various details about each other. These details about objects and their clients may include the specific locations, host machine and data representations, languages used for implementation, operating system underlying the host, or communication protocols used for information transfer.

Therefore, the use of the ORB software allows developing distributed applications which may comprise of the programs written in different source languages which are executable on different host machines and operating system platforms. The flexibility provided by the CORBA approach allows the distributed applications to be composed of legacy and third-party software as well as newly developed software.

The CORBA approach as it is today, has some limitations. The CORBA does not solve the problem of deploying the components of a distributed application (Evans and Rogers, 1997). The programs comprising a distributed application must be installed on the hosts where they will execute. This usually represents a problem for the client software in a multi-user distributed application. The deployment problems are faced in both the initial setup and in maintenance because software upgrades may require replacing of older components with the new ones. The deployment problem is further complicated by platform heterogeneity because the same component software may be required to execute properly on a range of different host architectures and operating systems. This is where the use of proven Web-technology consisting of Web-browsers and revolutionary language Java complements the CORBA in providing a very powerful and flexible approach to developing distributed client servers applications. Another limitation of CORBA is the power of the IDL. Although IDL allows specifying the operations containing parameters which may consist of any of the primitive data types and common constructed types in common programming languages (C, C++, Smalltalk, Java), the power of IDL is limited in that it does not allow the passing of arbitrarily complex types definable in the object-oriented languages (for instance, Java). The problem can be circumvented by disassembling the information contained in the aforementioned types into pieces and defining IDL interfaces for transfer of individual pieces. This, however, may involve making unreasonably higher number of remote invocations and callbacks from server to clients. Some of the providers of the ORB software have realized this limitation of the IDL and have defined upwardly compatible extensions to the IDL which enable transfer of complex data. One such technology will be described in a later section following a description of the ORB implementation chosen for this investigation.

3.4.2 Visibroker for Java ORB

The ORB software used in this investigation was the trial version of 'Visigenic Software's implementation of the ORB. This software is called Visibroker for Java (VBJ) and is available, for free evaluation for a period of three months, from 'Visigenic Software'. The version of the software used in the present investigation was VBJ3.0. The decision about using the VBJ was made after evaluating other rival technologies (for instance ORBIX WEB from the Iona Technologies). The major reason for choosing VBJ was its strict adherence to the industry standards which promises long-term success and viability of the technology. The following section will briefly describe the VBJ.

The VBJ is a complete implementation of CORBA 2.0 (OMG, 1995) ORB and supports a development environment for building, deploying, and managing distributed object applications (Visigenic, 1997). These applications are interoperable across platforms. Objects built with VBJ are easily accessible by Web-based applications that communicate using the OMG's Internet Inter-ORB Protocol (IIOP). The IIOP is the standard for communication between and among distributed objects running on the Internet, intranets, and in enterprise computing environments.

The VBJ connects (Fig. 1) a client program (an applet or an application), running

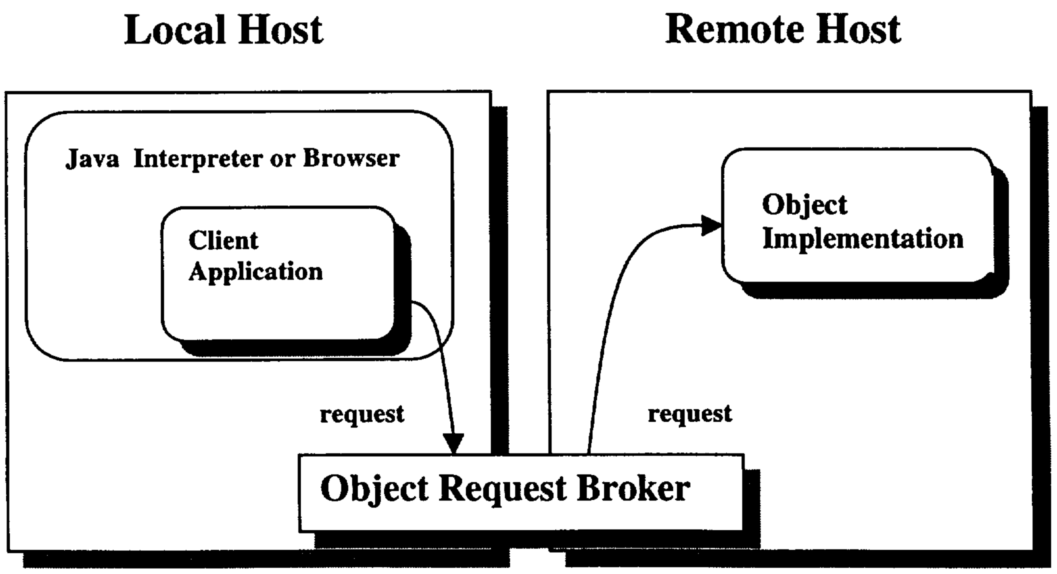


Figure 1. A client application invoking operations on an object through ORB

in a Java Virtual Machine (JVM) or in a Java-enabled browser, with the objects providing services the client program wishes to use. The execution of the object and its way of providing a particular service are transparent to the client program. The object may reside on the same host computer as the object or may be located on a remote computer somewhere on the network. The only things client program needs to know are the name of the object or the object reference and the way to use the object's interface. The ORB software takes care of the details of locating the object, routing the request, and returning the result. It is important to note that the VBJ ORB is not a separate process but is a collection of Java objects and network resources that integrates within end-user applications allowing the client applications to locate and use objects. The VBJ has several key features which are described in Visibroker Programmer's Guide (Visigenic, 1997). Various applications need not use all of the features and typically only use a few of them. This discussion will only briefly describe the ones relevant to this investigation.

- *Interface Repository*: The Interface Repository (IR) is an online database of meta information about ORB object types. Meta information stored for objects includes information about modules (an IDL namespace), interfaces, operations, attributes, and exceptions, all of which must have been defined using IDL.
- *Smart Binding*: This is a VBJ enhancement to the CORBA specification which improves performance by choosing the optimum transport mechanism whenever a client binds to a server object.
    - If the object is local to the client process, local method calls are used to communicate.
    - If the object resides in a different process, IIOP is used to communicate.
- *Smart Agents*: A VBJ smart agent, which is also an extension to the CORBA specification, facilitates obtaining object references. A Smart Agent can automatically reconnect a client application to an appropriate object server if the server currently being used becomes unavailable due to a failure. Furthermore, the Smart Agents can use Visibroker's Object Activation Daemon (OAD) to launch instances of a server process on demand.
- *Object Activation Daemon*: This is a facility to allow an object server to be launched automatically when a client expresses a desire to use the services provided by the

objects contained in the server. In order to allow the servers to be launched automatically, they must be registered with the OAD which includes command-line utilities for registering, unregistering, and listing objects.

- *GateKeeper*: The GateKeeper is an optional feature in Visigenic's implementation of the CORBA specification. The gatekeeper implements a mechanism which allows the client programs to make calls over the Internet to objects that do not reside on the Web server (Fig. 2). The client programs may also receive callbacks from the

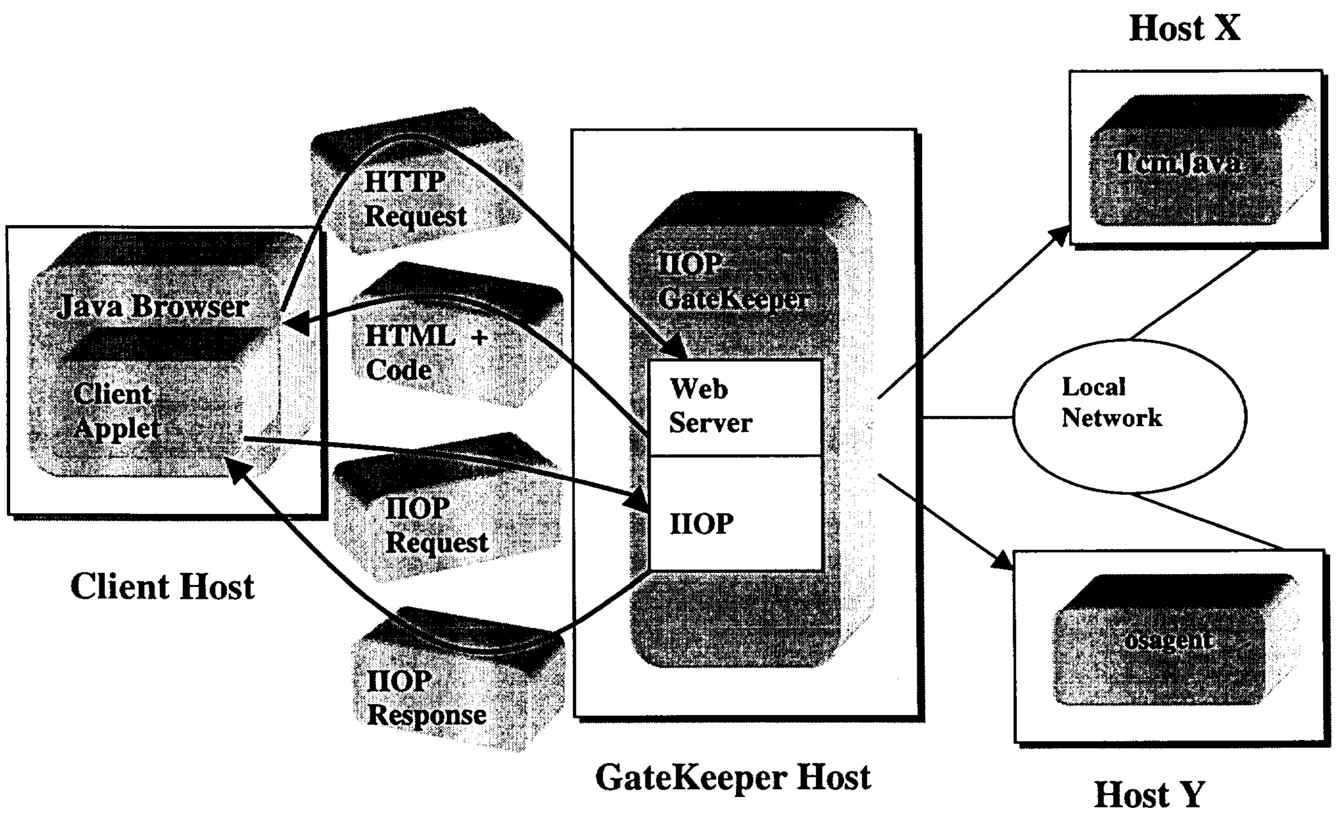


Figure 2. Interaction of an applet with IIOP GateKeeper in a distributed setting

aforementioned objects. The Gatekeeper runs on a Web server and uses mechanisms which fully conform to the security restrictions imposed by the Web-browsers. In addition, the GateKeeper handles communication through firewalls. The GateKeeper can also be used as an HTTP daemon, thereby eliminating the requirement for a separate HTTP server during the application development phase.

- *ORB*: The ORB supports the functionality specified by CORBA 2.0 (OMG, 1995) specification. The ORB includes:

- *Runtime*: The runtime supports the execution of the client or server programs.
- *Utilities*: Basic utilities used by the system administrator or the developer to obtain information about the state of the ORB environment.
- *Server Components*: Include Interface Repository, Smart Agent, and the OAD.

As pointed out earlier, IDL has limitations with regard the type of parameters that can be passed around. In Java, one can define arbitrary data types some of which may be analogous to IDL structures (an IDL data type struct). If a Java class is defined in a way such that it conforms to certain requirements, then it can be mapped to an IDL struct. If a Java class has to be mapped to an IDL struct, it must satisfy the following requirements:

- It must be a final class.
- It must be a public class.
- The class implementation does use inheritance.
- All the data members of the class are public.

If a Java class does not meet all of the above requirements, it cannot be mapped to a standard IDL *struct* type. This limits the kind of information that may be passed around as parameters using standard CORBA specification. The implementations of CORBA have tried to get around this limitation by defining extensions which are upwardly compatible with standard CORBA. One such extension is Visibroker's extensible structs which are described below.

3.4.3 Extensible Structs

The extensible structs implement pass by value semantics and allow parameters of an operation to be of an arbitrary Java class type. This mechanism allows passing of an object state from client Java program to server program using ORB brokered communication provided that a class definition of the Java object is present on the server side. The Java programs on the server side may invoke methods on the passed object which is a clone of the original object and has the same state as the original object. This mechanism uses Java Serialization to pass classes in the form of extensible structs. Java Serialization compresses a Java object's state into a serial stream of octets that can be communicated as a part of the remote requests. The extensible struct mechanism allows the data using pointer semantics to be passed successfully. A group of interrelated objects appears to be the same after its transport across a network. The use of this mechanism, therefore, greatly simplifies developing distributed Java applications that use CORBA and Web as their communication back-bone.

The preceding sections have described the Java, CORBA, and Web components of the approach used in this investigation to develop the infrastructure for building persistent data storage capabilities into the WHERE project. Use of this approach allows Java applets running at remote clients to store/retrieve data in/from a server which is free to choose any mechanism for managing the data. The methods employed for managing the data at the server are totally transparent to the clients. Therefore, the server implementation is free to choose from various mechanisms such as file systems, relational database management systems, or object database management systems. The persistent data management needs of the software engineering environments involving distributed collaboration are best served by using an object database management system (Emmerich et al., 1993; Orfali et al., 1996). However, the object database management systems are still under development and have not reached the popularity enjoyed by the relational systems. The ever-increasing popularity of Java has resulted in development of technologies which integrate the Java Programming Environment with the relational and/or object database management systems and allow Java objects to be stored transparently. These technologies take advantage of various features of proven database management technology while providing access to stored data using Java language features. One of such technologies is the Java Relational Binding (JRB) developed by the 'O2 technology'. This technology seemed to fit well into our approach for bridging the gap between a standard relational database management system (Sybase, Oracle, etc.) and an object-oriented language (Java). Therefore, we decided to explore the use of JRB in our attempt to build persistent data storage infrastructure. The following section gives a brief description of the JRB.

3.5 Java Relational Binding

The JRB (O2 technology, 1997) is a middle-ware product that bridges the gap between Java applications and relational databases. The JRB consists of a development environment and a runtime environment (Fig. 3):

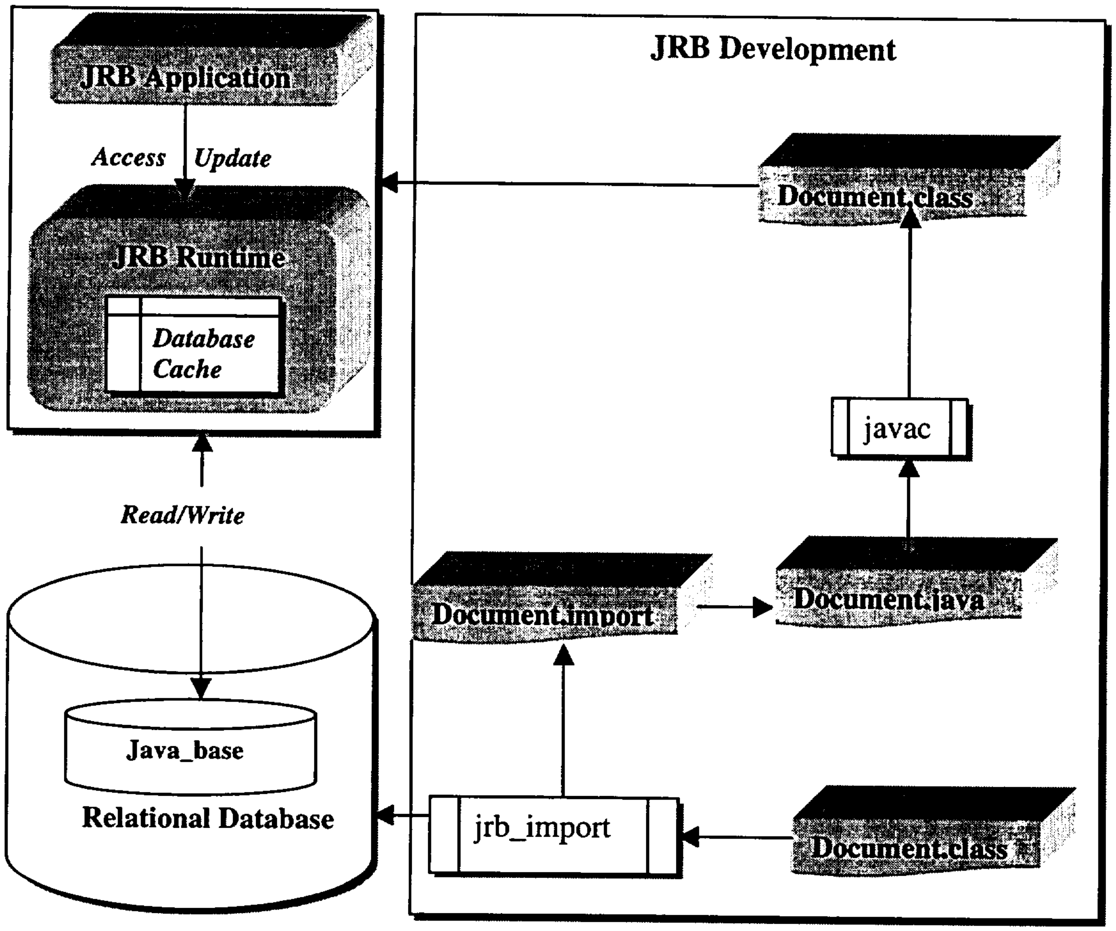


Figure 3. Architectural overview of a JRB application.

- The development environment consists of a set of tools (e.g. jrb_import) which, given the description of a set of Java classes, generate an equivalent relational schema and associated methods to read and write objects in the database. The generated methods map Java objects to their representation in the relational schema.
- The runtime may consist of a totally platform independent set of Java classes. However, particular RDBMS and platform specific versions are also available which

improve performance by using proprietary API of the RDBMS. The runtime runs on top of any JDBC compliant driver and manages an object cache to improve performance.

The JRB provides an alternative to the Java developers who want to store data in a relational database. The other alternative available which may be used by a Java developer for this purpose is the JDBC interface. However, the JDBC interface provides very limited functionality and leaves the burden of mapping a Java object to a relational schema on the developer. This involves writing Java code to map the Java object to the corresponding rows of the corresponding relations (tables). The same process must also be performed in the reverse direction in order to read the stored data into a Java program. Therefore, using JDBC interface involves lot of extra and unnecessary effort on the part of the Java developer. The JRB enables the developer to get rid of the drudgery of writing and debugging the same code over and over. The JRB attempts to overcome the limitations of the JDBC interface by providing the following capabilities:

- The developer only deals with Java class description and does not need to know the details of the relational model.
- The JRB development environment automatically generates the relational schema from the Java class description.
- The environment also generates the code mapping objects to the relations and in the reverse direction.
- Facilities are provided to allow limited evolution of class definitions.
- The JRB utilizes the native mechanisms of the RDBMS to provide transaction management and referential integrity among the objects stored in the database.

The interface, provided by JRB to access the database functionality, is very simple. The API of the JRB uses standard Java classes and builds on and reinforces the style and virtues of the existing core Java classes. The developers of the JRB have plans to make the binding available on the O_2 Engine (O_2 object database management system) in near future. This further enhances the suitability of JRB for an approach like ours because the interface will be same whether the underlying database management system is an O_2 database engine or a relational system.

Having described individual components of the approach used in this investigation, it is now time to put them together and describe how the overall system will work. The architecture of the infrastructure developed during this investigation is given in Figure 4. The bytecodes for the TcmJava classes and other code necessary for the TcmJava editors to run is located on the Web server which is under the control of the person responsible for administering the persistent data management system. A TcmJava editor may be launched by an authorized person by using a JDK 1.1 compatible Web-server. Whether a person is authorized is determined by the persistent data management system based on login and password information furnished by the user intending to the launch a TcmJava editor. A user will request access to a TcmJava editor by specifying the URL of the Web page containing the applet that will control the subsequent operation providing the user various kinds of information. Once the authenticity of the user is

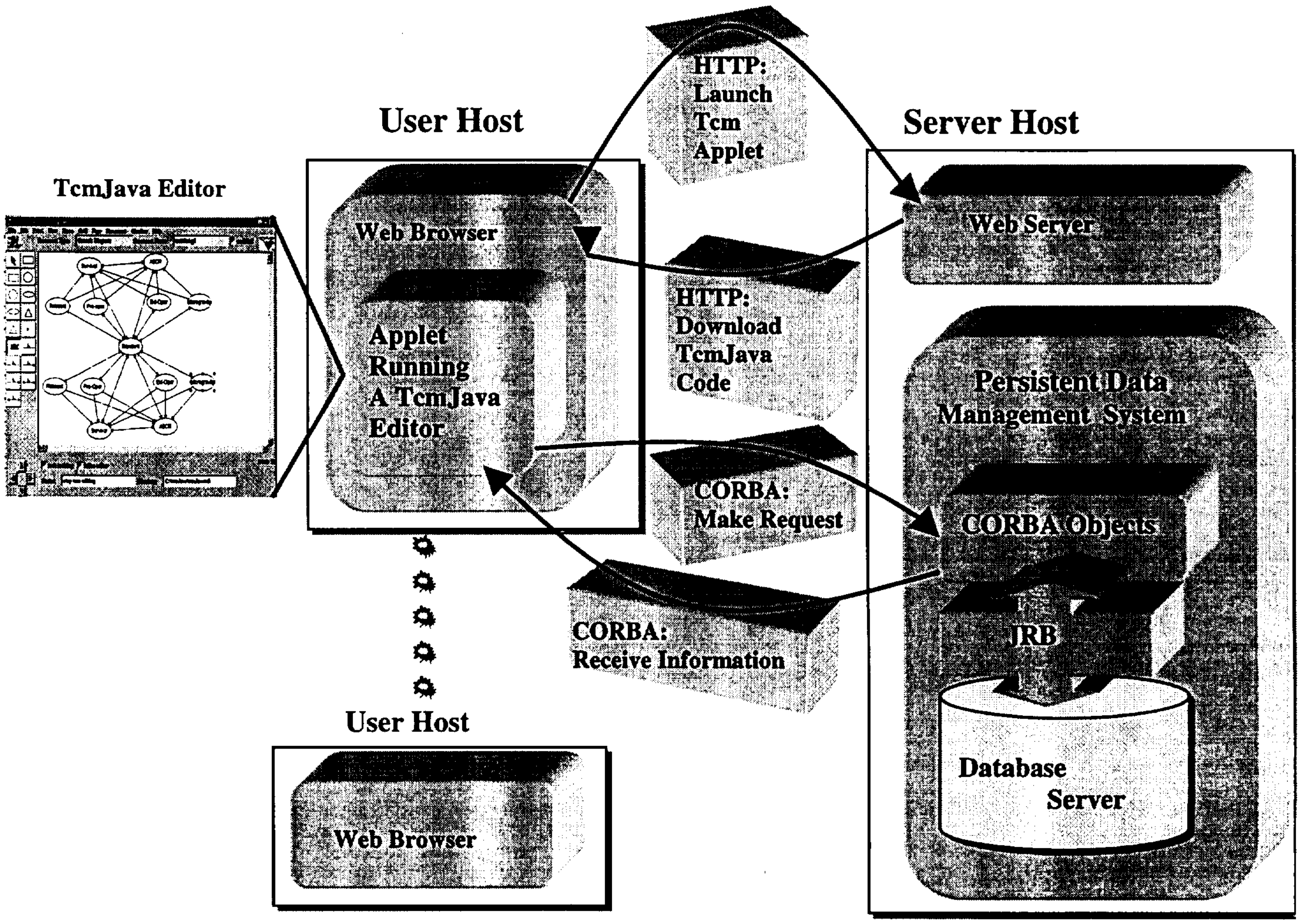


Figure 4. Architecture of the Java, CORBA, Web, and Database Server approach

verified, the system will try to initialize the CORBA environment displaying error messages in case of failures. However, the initialization process is carried out transparently and the user does not need to know what kind of initialization occurs. The system would have done necessary initializations and established connections for remote invocation before the user runs an editor. The user can then read the documents, edit them, and save them back to the server. Similarly new documents may be created and saved. The system keeps track of the owner of a document who may change the access rights for the other users. The TcmJava editors used by the user execute on the client machines and make remote requests through CORBA calls which are conveyed by the ORB to the appropriate object. Rest of the management of persistent data is transparent to the clients and is done by the object implementations. In our approach the objects are implemented in Java and use JRB API to store and retrieve data from a relational database server. The details about the system will be given in the next chapter which will discuss the results of this investigation. The purpose of this chapter was to give an overview of various methods and technologies used in the approach developed during this investigation.

CHAPTER 4: RESULTS AND DISCUSSION

Results and Discussion

This chapter discusses the results of the present investigation. The previous chapter (Methodology) described various methods and technologies used to implement the approach developed during this investigation for building persistent data storage infrastructure for the WHERE project. This chapter discusses various components of the infrastructure providing implementation information wherever appropriate. There are four components of the "Java, CORBA, Web, and Database server" approach used for building the persistent storage infrastructure. These components will be described and discussed in order of their implementation during this investigation. The order of description is as follows:

- Java Component: Storage/Retrieval Mechanism of the TcmJava
- CORBA Component: CORBA compliant Java code
- Web Component: Integration of TcmJava and CORBA components with Web
- Database Server Component: JRB and Database Server

4.1 Java Component: Storage/Retrieval Mechanism of TcmJava

The previous chapter described the translation of C++ code (TCM) into Java code (TcmJava). One of the major problems with the translated code was mapping of the storage/retrieval mechanisms used in TCM to those in TcmJava. In TCM the storage/retrieval process starts in the class *Document* and subsequently ripples through subclasses of the *Document* and other two classes (*Shape* and *Subject*) and their subclasses. The conceptual view of the process of storing the simplest document in the TCM (a Generic Diagram) is given in Fig. 5. The process starts with the *Save* method in

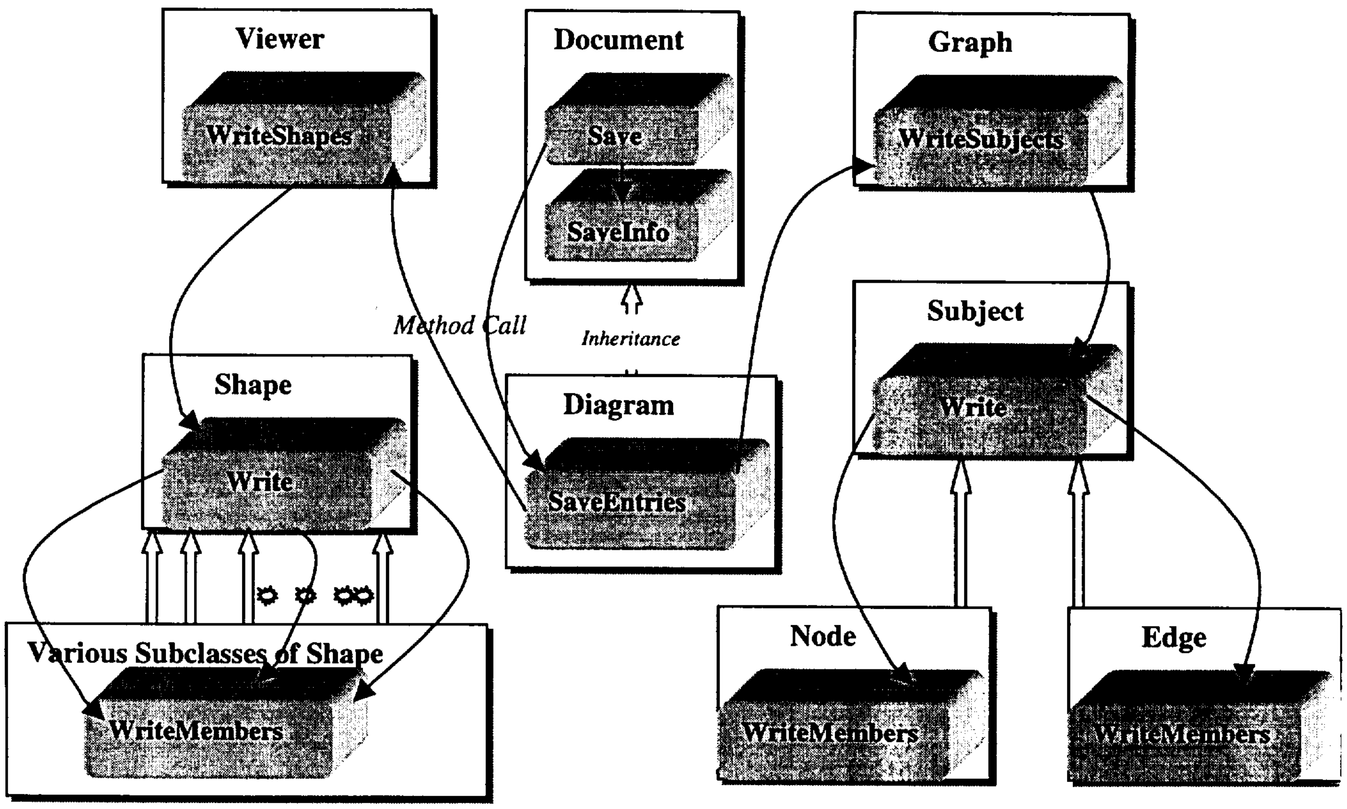
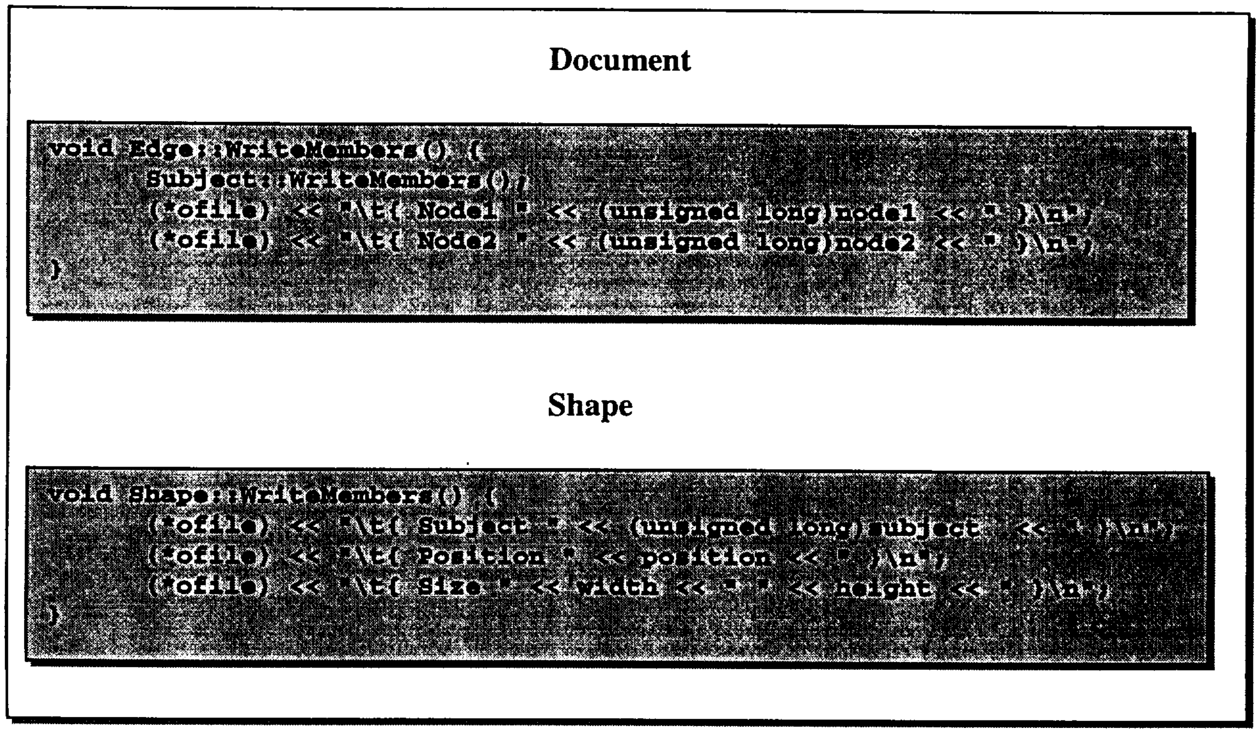


Figure 5. Conceptual view of the process of storing a document generated by TCM

the *Document* class. This method writes information, which is common to all documents generated by TCM, to a file by calling the *SaveInfo* method in the same class. The *Save* method also calls the *SaveEntries* method of a sub-class of the *Document* (*Diagram*, *Table*). The *SaveEntries* method calls *WriteSubjects* and *WriteShapes* methods in classes *Graph* and *Viewer*, respectively. The *WriteSubjects* and *WriteShapes* methods call the *Write* method in classes *Subject* and *Shape*, respectively. The *Write* methods of both the *Subject* and *Shape* classes write some information to the file and then call *WriteMembers* methods in their sub-classes which usually involve more than one level down the class hierarchy depending upon the kind of document being saved. These *WriteMembers* methods are where the information specific to a particular part of the document is written. The implementation of a typical WriteMembers method from both a *Subject* and *Shape* classes is given in Fig. 6. Problem with translating this code into Java was the storing of



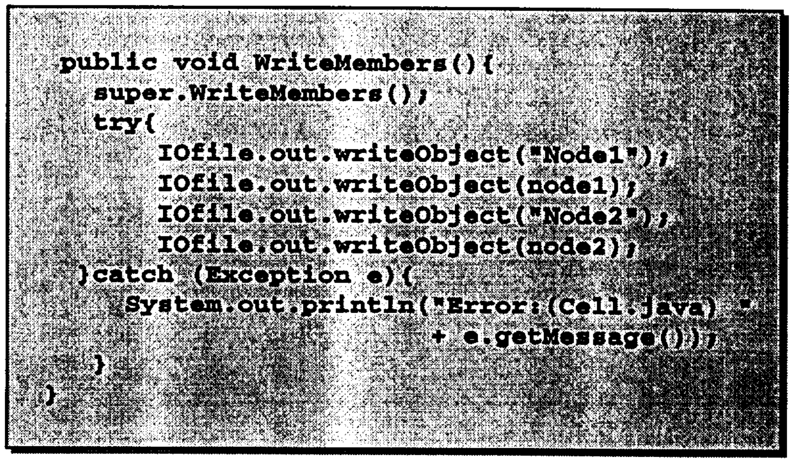
```
void Edge::WriteMembers() {
        Subject::WriteMembers();
        (*ofile) << "\t{ Node1 " << (unsigned long)node1 << " }\n";
        (*ofile) << "\t{ Node2 " << (unsigned long)node2 << " }\n";
}
```

```
void Shape::WriteMembers() {
        (*ofile) << "\t{ Subject " << (unsigned long)subject  << " }\n";
        (*ofile) << "\t{ Position " << position << " }\n";
        (*ofile) << "\t{ Size " << width << " " << height << " }\n";
}
```

Figure 6. Implementation of typical WriteMembers methods for Subject and Shape classes

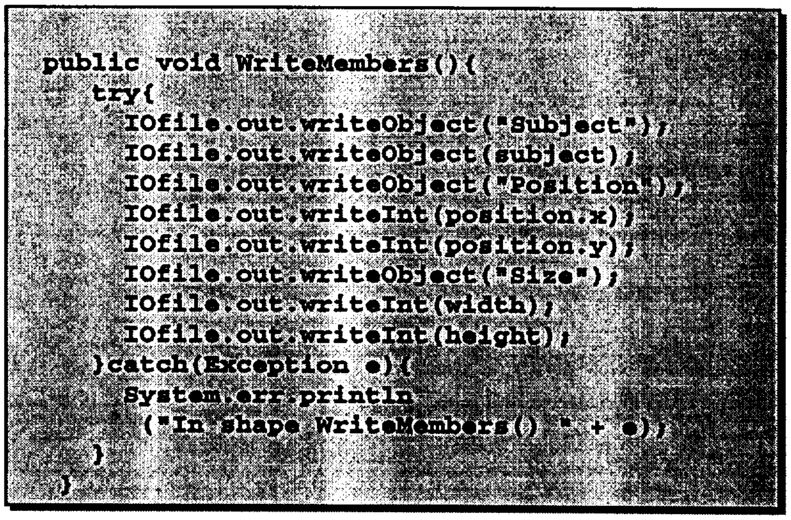
object references (casting of a pointer to *unsigned long)* as *long* values into a file. Fortunately, by the time we reached that stage of translation Java Serialization was introduced as a part of the JDK 1.1. Use of Java Serialization made the storing of objects much simpler. It allows the object to be written as such along with any other objects transitively reachable from the object being written. Therefore, we used Java Serialization's *WriteObject* method to translate the TCM code which used pointer to *unsigned long* conversion before storing it into an ASCII file. The C++ code shown in Fig. 6 was translated into the Java code given in Fig. 7. This approach worked for the

Document



```
public void WriteMembers(){
   super.WriteMembers();
   try{
       IOfile.out.writeObject("Node1");
       IOfile.out.writeObject(node1);
       IOfile.out.writeObject("Node2");
       IOfile.out.writeObject(node2);
  }catch (Exception e){
    System.out.println("Error:(Cell.java)"
                          + e.getMessage());
  }
}
```

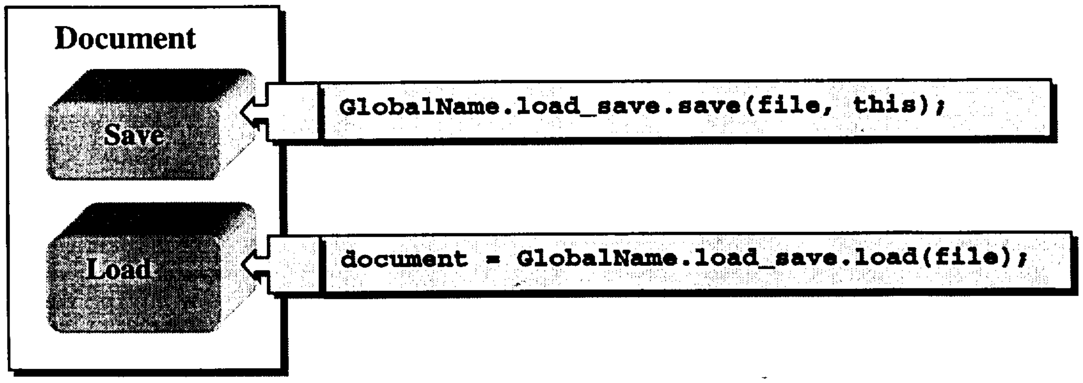
Shape



```
public void WriteMembers(){
   try{
     IOfile.out.writeObject("Subject");
     IOfile.out.writeObject(subject);
     IOfile.out.writeObject("Position");
     IOfile.out.writeInt(position.x);
     IOfile.out.writeInt(position.y);
     IOfile.out.writeObject("Size");
     IOfile.out.writeInt(width);
     IOfile.out.writeInt(height);
   }catch(Exception e){
     System.err.println
      ("In shape WriteMembers() " + e);
   }
}
```

Figure 7. Translated code of WriteMembers methods for Subject and Shape classes

standalone TcmJava application which stored/retrieved data in/from a local Unix/DOS file. However, it involved making a very large number of input/output (IO) calls and wrote unnecessary information to the file. Moreover, this approach would not work for a distributed TcmJava application involving remote method invocations to store/retrieve data in/from a remote server. As such this approach was grossly inefficient and would become impractical if data was to be stored/retrieved over a network. Since our objective in this investigation was to build persistent data management infrastructure for a project aiming to build tool support for distributed collaboration, we must devise an approach which would involve making the least number of invocations while storing/retrieving data. In addition, we wanted to leave the original architecture of TCM intact in the translated version. After analyzing the storage/retrieval mechanism of the original TCM, we were able to devise an approach which involves storing/retrieving lesser information than the original approach and is more elegant. Our approach involves writing the *Document* object itself to the persistent storage. The Java Serialization takes care of Saving/Loading of the information which is a part of the *Document* Object (i.e referenced by it). The writing of information derivable from the stored information is prevented by making the fields representing that information *transient*. The reading/writing activity now involves making a single invocation of an operation irrespective of whether it is a local or remote invocation. Moreover, the reading/writing occurs entirely in the *Load/Save* methods of the *Document* class. This approach does not require to explicitly store extra information needed to interpret the stored information when it is retrieved; Java serialization takes care of that. A comparison between Figs. 5 and 8 illustrates the features of our approach vis-à-vis the approach used in TCM. The document



```
GlobalName.load_save.save(file, this);
document = GlobalName.load_save.load(file);
```

Figure 8. Conceptual view of the storage/retrieval process in TcmJava.

storage/retrieval process in our approach involves making a single call to the *Save/Load* method of the *load_save* class which implements the persistent storage mechanism. The details of how and where the persistent storage occurs are transparent to the TcmJava application performing the storage/retrieval. The *load_save* class may implement a storage mechanism which saves the *Document* object to a local file or may be a mechanism which transports data over a network and makes it persistent on a remote server. Again, the details of how the data is made persistent locally/remotely are transparent and may be changed without any effect on the working of the TcmJava application using them. The development of this approach was the first step in building the infrastructure. With this first step in place, if the data were to be persistently managed locally, implementing the *load_save* class meant using *WriteObject* and *ReadObject* methods of the Java interface *java.io.Serialization*. However, our aim was to develop an infrastructure which will be used by geograhically distributed multi-client TcmJava application. To accomplish this, we needed a mechanism which will allow storage/retrieval of Java objects to a remote server transparently. This leads to the second component of our approach which will be discussed in the following sections.

4.2 CORBA Component: CORBA Compliant Java Code

While using the Web as a communication back-bone for building distributed Java applications, one faces the problem of applet authentication. The Java language and run time have built-in mechanisms to ensure that the applets downloaded from the Web via a Web-browser do not compromise the security of the local system. Java runtime has a built-in authentication mechanism which checks to determine whether an applet is allowed to perform a particular action. Whenever, an applet tries to perform a security-sensitive operation, the runtime throws security exception and the operation is prevented. Examples of the security-sensitive actions include accessing the file system of a Web-browser's host, or opening a network connection to another host. Therefore, a mechanism is required such that the security-sensitive actions could be delegated to that mechanism which will perform them transparently in a secure manner. Implementations of CORBA represent one of such mechanisms which can be used in developing distributed Java applications involving data transport and using Web as the communication back-bone. The Remote Method Invocation interface released as a part of the JDK 1.1 is another such mechanism. Two mechanisms are capable of serving the kind of functionality required for implementing the infrastruture we are building. However, we decided to use the CORBA approach because: it allows a greater flexibility in implementing the server functionality; the software required to enable the remote invocations is available in the form of ORB implementations; CORBA offers additional functionality (CORBA services) which can be taken advantage of during building subsequent parts of the infrastructure. An overview of CORBA and one of its implementations (Visibroker for Java) used in this investigations was given in the previous chapter. Here we will discuss the details of using the CORBA technology to provide the functionality required during the present investigation.

The use of ORB technology for transporting information between TcmJava client applications and the persistent data management server, required defining CORBA interfaces which will then be transparently used by the clients for sending and receiving data from the server. Normally, one would define the interfaces using the IDL which makes them usable by any CORBA compliant application irrespective of the platform and implementation language. However, the data transfer capabilities of the standard IDL did not meet our needs. Fortunately, the CORBA implementation, we are using, uses an extended version of standard IDL which allows transferring of information represented by an arbitrary Java class. This version of IDL is upwardly compatible with standard CORBA and the interfaces making use of extended features can be used in conjunction with the standard IDL interfaces which will still be completely platform and implementation independent. Since our client applications will be Java anyway, it did not matter for client. However, it affords the capability of being able to provide the part of the server implementation by using environments other than Java if such a need should arise during building various functionality into the infrastructure. Our prototype of the infrastructure defines a CORBA interface (Fig. 9) specifying the operations available to the client TcmJava applications for storing and retrieving the persistent data. A TcmJava

```
  public interface TcmLoadSave extends org.omg.CORBA.Object {
  //
  public String save(String fileName, Document document);
  public Document load(String fileName);
  public boolean fileExists(String fileName);
  public boolean isDirectory(String fileName);
  public boolean isFile(String fileName);
  public String[] listFiles(String dirName);
}
```

Figure 9. Interface definition for the CORBA object providing persistent storage.

Client application uses this interface without worrying about how the data is made persistent. The CORBA object implementing this interface determines how and where to store the data. An implementation of TcmLoadSave interface may be a CORBA object which manages the persistent data in a file system of some operating system running on a host machine somewhere over the network. One such implementation is given in Fig.10.

```
public class TcmLoadSaveServer extends LoadSave._sk_TcmLoadSave {
  public static final String LOAD_SAVE_DIR =
  "/projects/nasa/data/tcmJava/root";
  public TcmLoadSaveServer(String name){
    super(name);
  }
  //
  public String save(String fileName, Document document){
    fileName = LOAD_SAVE_DIR + fileName;
    //
    if (ReadWriteUtility.writeObjectFile(fileName, document))
      return "Succeeded";
    else
      return "Failed";
  }
  . . . . . . Implementation of the remaining methods . . . . . . .
}
```

Figure 10. An implementation of the TcmLoadSave CORBA interface .

The class in this implementation (TcmLoadSaveSever) sub-classes (*extends*) the *LoadSave._sk_TcmLoadSave* class which is a class that is automatically generated by the IDL compiler and used by the ORB software for delegating the requests to the actual implementation. The Java source code for *the LoadSave._sk_TcmLoadSave* class is given in Fig. 11. This class is a sub-class of another automatically generated class

In our prototype implementation, the Java classes implementing the TcmJava editors are located on a host running in the domain 129.164.10.x (x is a specific host address) and can be accessed via a Web-server (Fig. 13). Any user (for the demonstration version) having access to the Web and having a JDK 1.1 compatible Web-browser could download and run the TcmJava editors. When an editor is launched, it establishes Internet connection with a CORBA object managing the persistent data that is generated by the editor and communicated to the CORBA object. The communication between the TcmJava and the CORBA objects is mediated by the combination of the IIOP GateKeeper and the OSAGENT. In our prototype implementation, the CORBA objects run on two hosts in the domains 157.182.114.x and 157.182.112.x. One of the hosts is an NT workstation and the other one is Sun SPARC station. The CORBA

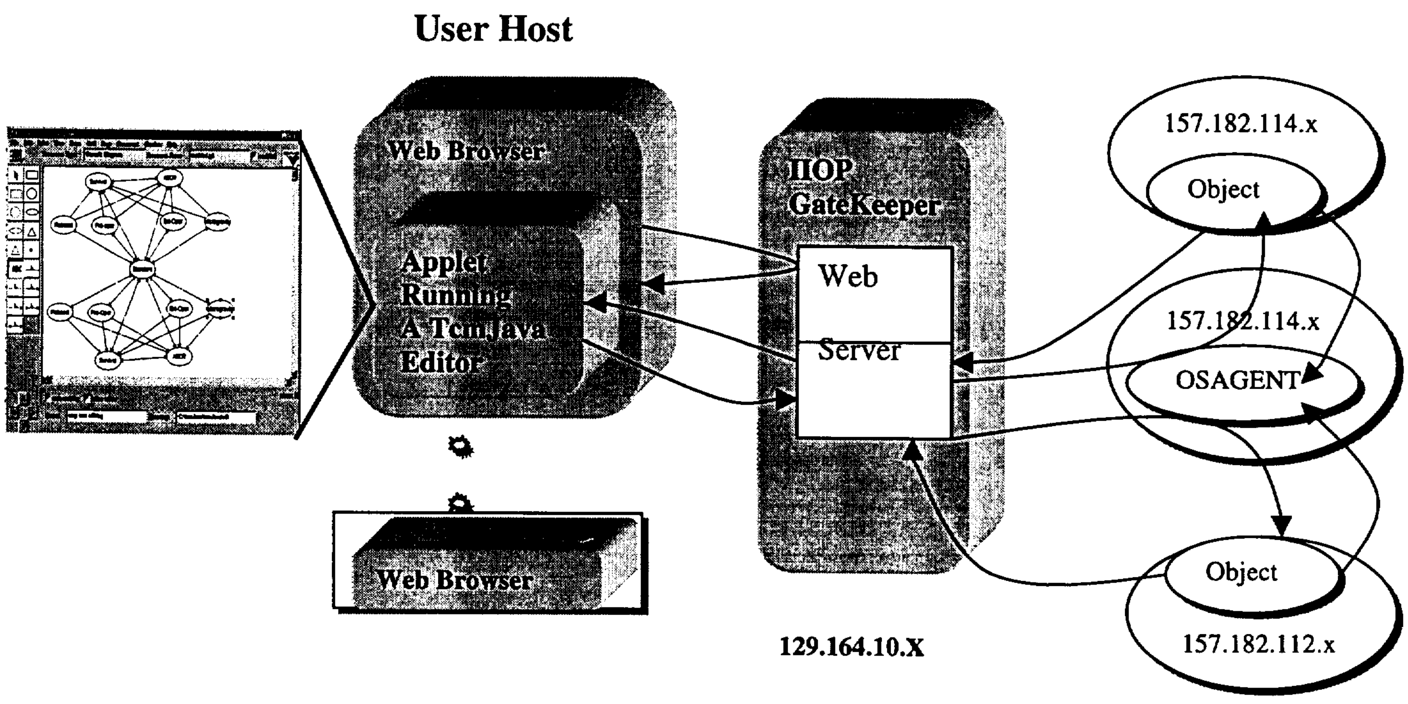


Figure 13. Description of the operation of the prototype implementation

objects running on these hosts store the data communicated to them on the local DOS/UNIX files, respectively. A user of the TcmJava may specify on which host to store the data. However, the data does not have be stored in a file system. Instead, a more powerful mechanism such as a relational or object-oriented database server may be used for managing the data. These considerations will be discussed in a later section following the next section which discusses the third component of our approach.

4.3 Web Component: Integration of TcmJava and CORBA components with Web

This component deals with using the well-developed Web-technology as a communication back-bone in our approach. Our decision to translate the C++ code of TCM into Java allowed us to use the capabilities of the Web. Furthermore, the security issues involved in the use of the Web technology were circumvented by using ORB technology. Although our approach works with any Web-server implementing the WWW protocol, it allows the use of a specialized Web-server. In our prototype implementation, we used the JavaWebServer from Sun Microsystems running on an NT Workstation as well as general purpose Web-servers running on Sun SPARC stations. The JavaWebServer provides Java specific capabilities which may be taken advantage of during subsequent developments of the persistent storage infrastructure. The present investigation, however, did not explore the use of these capabilities. An important feature of our approach is that it benefits from the development of browser technology; the Netscape Communicator has in-build Java and Visigenic-ORB Run-time environments which greatly reduce the Internet traffic involved in launching TcmJava applications using the Web.

In this and two previous sections, we have discussed the components of our approach involved in running a TcmJava editor from multiple Web-clients and communicating the data generated by the editor to CORBA objects responsible persistently storing the managing the data. The following section discusses the mechanism to be used for making the data persistent.

4.4 Database Server Component: JRB and Database Server

Various issues involved in meeting the persistent data storage needs of a Software Development Environment (SDE) were reviewed in the chapter on Literature Review (Chapter 2). It was pointed out that the relational database management systems as such are not suitable for use in SDEs because of lack of ability to specify and manage the complex data generated during various phases of an SDE. The object-oriented databases although very suitable for use in an SDE have not reached a development stage which warrants their exclusive use in providing the persistent data storage needs of an SDE. Therefore, we explored the use of a hybrid approach which uses the object-oriented methods to persistently manage data in a relational database management system. This section will describe the details of the approach.

This hybrid approach uses a middle-ware product from 'O_2 Technology' called Java Relational Binding (JRB) which provides a high level interface to an underlying database, where Java objects and class information are stored. An overview of the JRB was given in the chapter on Methodology (Chapter 3). In this section, we will discuss the details involved in its use in our approach. The JRB API consists of a set of Java classes which are used by the Java applications intending to store/retrieve Java objects from the database. The methods used for Storage/Retrieval of Java objects in/from the database are specified in Java Interface called *PersistentObject*. Any class whose objects are to be stored in the database must *implement* this interface. The implementations of the methods declared in the *PersistentObject* interface are generated by a tool provided with the JRB which takes the Java class (whose objects are to be made persistent) as input. Therefore, any classes which represent the information to be made persistent (stored in the database) must be *imported* into the database by using the import tool (*jrb_import*). In TcmJava there are a large number of classes which represent the information generated by various TcmJava editors. If the approach is to be used as such, all these classes must be imported. Also, if any modification is made to any of them, the modified class must be re-imported. In addition to the need for importing unnecessary classes, the approach would make the storage mechanism dependent on the classes used to generate the information. This necessitated developing another mechanism which would involve importing of a smaller number of classes and would make the storage mechanism ıdependent of the TcmJava classes. Therefore, a set of new classes were implemented › circumvent the difficulties involved with using the TcmJava classes as such. These lasses are packaged in a Java Package (tcmJavaServer) and are described in the ›llowing section.

.4.1 The tcmJavaServer Package

his package contains the classes used to represent the information to be made persistent nd will be imported into the database. The rationale for implementation of these classes ame from the fact that in TcmJava, the information to be made persistent, was epresented by only a small number of classes. However, these classes were either very ›w or very high in the class hierarchy, thereby, necessitating the storage of intermediate lasses which actually did not represent any information but must be stored because of eing part of the class hierarchy. One of such class hierarchies of the tcmJava is given in igure 14. The Figure also shows the corresponding hierarchy in the new package nplemented for making the information persistent. In this case, if an object of class ripleBox as it occurs in the tcmJava package has to be made persistent, all the classes hown in the hierarchy in the left-hand side of the Figure must be imported into the

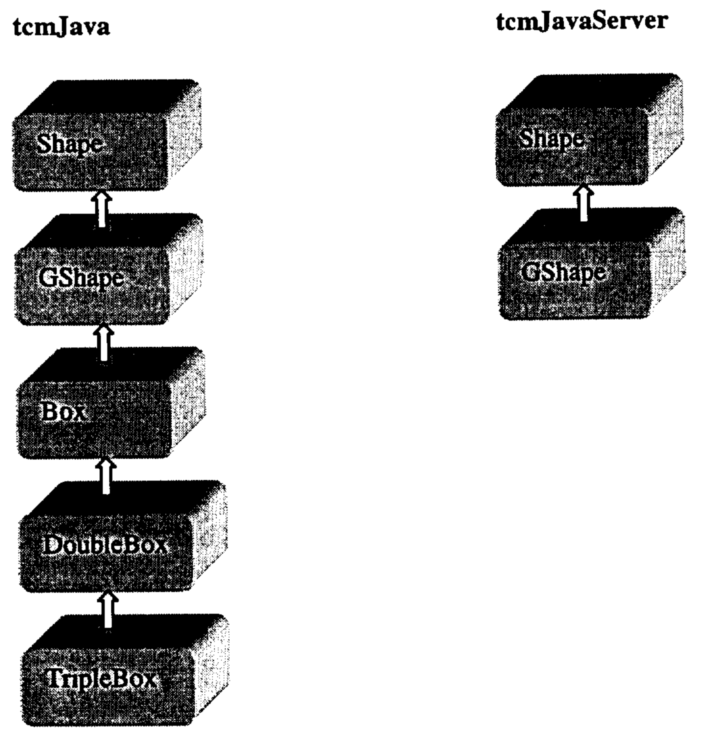


Figure 14. Corresponding class-hierarchies in the tcmJava and tcmJavaServer packages.

The classes of the tcmJavaServer package will be made persistent by importing them into the underlying database via the *jrb_import* tool which is a part of the JRB. After importation into the database, each class will implement the *PeristentObject* interface and will contain methods required for performing various operations required for reading, writing, and managing the data. The following section discusses each of these operations by using one of the classes in the tcmJavaServer package as an example.

4.4.3 Persistent Objects

Figure 15 shows the implementation of one of the classes in tcmJavaServer package before being persistence capable (being imported into the database). This class will be

```
public abstract class Subject extends Object{
  protected String name;
  protected Graph graph;
  protected initialize(SubjectData data){
    this.name = data.name.getstr();
    if (data.graph instanceof tcmJava.sd.DFGraph)
      this.graph = new DFGraph((tcmJava.sd.DFGraph)data.graph);
    else
      this.graph = new Graph(data.graph);
  }
  //
  protected void initSubject(tcmJava.dg.Subject subject, SubjectData
data){

................................... more methods ...............................
```

Figure 15. A typical class from tcmJavaServer package before being persistence capable.

used as an example to illustrate the process of making tcmJava objects persistent, using the JRB and a relational database server. In order to store/retrieve data represented by this class in/from the relational database, the class must be made persistence capable by importing it into the database and subsequently changing its class-definition followed by recompilation. The persistence capable version of this class is shown in Figure 15.

```
public abstract class Subject extends Object implements PersistentObject{
  protected String name;
  protected Graph graph;
  protected initialize(SubjectData data){
    this.name = data.name.getstr();
    if (data.graph instanceof tcmJava.sd.DFGraph)
      this.graph = new DFGraph((tcmJava.sd.DFGraph)data.graph);
    else
      this.graph = new Graph(data.graph);
  }
  //
  protected void initSubject(tcmJava.dg.Subject subject, SubjectData data){
..................................... more methods ..............................

  bodies of methods declared in PersistenObject interface
```

Figure 16. A typical class from tcmJavaServer package after being persistence capable.

After importing the class into the database, its definition must be modified such that it *implements* the *PersistentObject* interface of JRB API. Before this modified class implementation can be recompiled, the definitions of the methods declared in the *PersistentObject interface* must be inserted into the class body; these method definitions are generated by the jrb_import tool used for importing the classes into the database. After this modified definition of the class is compiled by using the Java Compiler, objects of this class may be stored/retrieved in/from the underlying database.

Once a class is made persistence capable, the application manipulating objects of that class is given full control over the persistent data through the methods declared in the *PersistentObject* interface. In addition, JRB provides various other functionality through static methods of some utility classes which are part of the JRB API. The following sections will discuss management of the persistent data represented by tcmJavaServer classes which had been made persistence capable by using JRB and an underlying relational database server.

4.4.4 Transaction Management

The transaction management capability available to the persistent tcmJavaServer objects can be very useful in the view of being able to provide the persistent data to multiple-users using tcmJava editors from remote sites. The transaction management is conveniently provided through the methods of a class (*Transaction*), part of the JRB API. The methods of this class give control to the application to manage transactions. This allows the process (to be defined to control the operation of the WHERE environment) to specify the transaction management policies but yet providing a convenient way to enforce them. For instance, the process may specify that certain documents may be viewed by a group of users but may be modified by a subset of those users. The transaction mechanism together with the access control mechanism (to be discussed later) provides a convenient way of enforcing such a policy. The users allowed only to view a document can be restricted to open the document in read-only mode. Regarding the users with update rights, concurrent updates can be easily supported, of course, according to a policy specified by the process. The system provides convenient ways and leaves the control to the programmer who may enforce various policies.

4.4.5 Access Control

The access control is provided by first defining users and then assigning them various access rights. The user management is done through a Java class which allows adding users, defining login information, changing the login information, and deleting the users. The user management can only be performed by the person who created the database. A user can be allowed/refused following six kinds of rights:

- *Import* : The user is enabled/disabled to import Java classes.
- *Access* : The user is enabled/disabled to read persistent objects.
- *Update* : The user is enabled/disabled to write persistent objects.
- *Delete* : The users is enabled/disabled to delete persistent objects.
- *All* : The user is enabled/disabled to read, write, and delete persistent objects.
- *Grant* : The user is enabled/disabled to perform all the previous operations and to give grant permission.

The user who imports the Java classes into the database has all the rights on the imported classes. This user may grant or revoke access rights to other users on classes imported by him. This prevents the unauthorized users from accessing the persistent data managed while making it accessible to multiple users running tcmJava editors from distributed locations. Also the security control lies with a single user which is a very desirable feature in distributed applications making persistent data accessible to multiple users.

4.4.6 Creating Persistent tcmJavaServer Objects

All the tcmJavaServer classes that have been imported into the database are provided with the methods for writing them into the database. The CORBA object which is connected to a tcmJava editor running in a remote host implements the persistence mechanism. This object receives a copy the tcmJava object, whose data is to made persistent, through a CORBA call made by the applet running the editor. The CORBA object creates a corresponding tcmJavaServer object and copies the data from the tcmJava object to this object. The tcmJavaServer object and all the objects pointed to by it are then written to the database through methods in the *PersistentObject* interface. The writing occurs inside a transaction where the objects being written are locked to preserve consistency. Mechanisms are provided to prevent deadlocks among the concurrent updates to a given object. The enforcement of concurrency is left to the programmer and may be dictated by the process governing the environment (WHERE), the persistence mechanism is part of. References among the stored objects may be created according to some specified policy and may be used to implement a process model. For instance, the consistency among stored documents may be checked and enforced by creating references. In addition, the checking may be done at the server side without the need for transporting lot of data over the network which may be needed if consistency checks are to be performed at the client side. This approach, therefore, tackles a number of issues involved in providing persistent data storage in an SDE; these issues were pointed out in chapter on Literature Review (Chapter 2).

4.4.7 Retrieving Persistent tcmJavaServer Objects

The tcmJavaServer objects stored in previous sessions can be accessed in a current session though data entry points defined in the database. The object access occurs through *class extents*. A class extent contains all instances of a class that had been previously written to the database. The system defines two types of class extents which provide access to instances of a class only or to instances of a class and all its sub-classes. A class extent can be filtered through a predicate (very similar to the *where* clause of a *select –from-where* **SQL** query) to obtain a particular instance. For instance, a tcmJava document is represented by an instance of the *Document* class defined in the tcmJavaServer package. One of the fields of the *Document* class is a *String* which identifies a particular document. Therefore, a particular document may be retrieved from the database, inside a Java program (Fig. 16). When an object is retrieved from the

```
Document document;   // declare a variable of document type.
Extent documentExtent ; // declare a variable of type extent.
documentExtent = Extent.all("Document"); // obtain a reference to all instances of Document and its
// subclasses
document = documentExtent.where ("id = 'some_identifier'").element();
```

Figure 17. Retrieval of a previously stored document from the database.

database, all of its fields of primitive data types are read. However, the reference types must be explicitly read using methods provided in the *PersistentObject* interface. The system also provides query capabilities using primitive as well as reference-type fields of the stored object. For instance, one of the queries could be: retrieve all documents created by user X. Since an object of class Document contains a field representing name of the document creator, the aforementioned query can be easily made. The results of a query can be read into an object of Java *Enumeration* type and used in anyway the programmer deems appropriate. These capabilities can be very useful when building subsequent components of the WHERE project.

4.4.8 Deleting Persistent Objects

The *PersistentObject* interface contains a delete method which facilitates deleting of a stored object. Before an object can be deleted, the system checks to see that the object is not referenced by any other stored object. If the references to the object being deleted exist, the deletion is disallowed and an exception is raised to inform the application attempting to perform the deletion. The deleted objects remain in the memory of the application and may be written back to the database. This feature can be very useful for enforcing consistency among the stored documents according to some process governing the operation of the SDE (WHERE).

This chapter described and discussed the four components of the approach developed during this investigation for building persistent data storage infrastructure of the WHERE project. The approach discussed here will serve as a foundation on which the subsequent functionality will be built. This approach provides for communication among geographically distributed people who are part of a software engineering team. The approach uses the Web as the communication back-bone and CORBA for allowing multiple users to read/write data from/to multiple servers. The data storage mechanisms are transparent to the tcmJava applications using them. The approach provides for performing a large portion of the computing on the server-side, thereby, cutting-down on the amount of data that must be communicated among the geographically distributed locations over the Network. A relational database server with Java language access is used to store the data with flexibility to switch over to an Object-Oriented Database Server in future. Our system has tried to handle a number of issues involved in providing persistent data storage for a distributed software engineering environment. The following chapter (Chapter 5) will summarize and conclude the thesis pointing out needs for future work.

CHAPTER 5: SUMMARY, CONCLUSIONS, AND FUTURE WORK

Summary, Conclusions, and Future Work

This chapter summarizes and concludes the thesis, pointing out the need for future work. This thesis describes and discusses the results of an investigation undertaken to build an infrastructure for providing persistent data storage for a Software Development Environment (WHERE) aiming to provide collaboration among the members of a geographically distributed team. The aim of the investigation presented in this thesis was to build the foundation software for the WHERE project. A software implementing a set of Diagram and Table editors (TCM) was adopted as the foundation software. The C++ code of the TCM software was translated into the Java programming language in order to enable the software to be used in a distributed environment taking advantage of the Web technology. The persistent storage mechanism of the TCM software had been designed to be used in a single user environment and, therefore, had to be redesigned to use it in a distributed environment. An implementation of CORBA (Common Object Request Broker Architecture) from Visigenic Software (VisiBroker for Java) was used to facilitate the communication between the tcmJava editors running as Java applets on client hosts and the persistent data server managing the data generated by the editors. The persistent data server was implemented as CORBA objects running on a Network host and receiving data from and sending data to the tcmJava editors. The data could be transparently stored on a single server or multiple servers each of which would be implemented by a CORBA object. A CORBA object implementing the persistent data server for tcmJava uses a middle-ware called Java Relational Binding (JRB) to store/retrieve persistent tcmJava objects in/from a relational database server. The Transaction and User management facilities provided by JRB could be used by CORBA objects implementing the server to provide multi-user concurrent access to store tcmJava objects in a secure fashion.

The results of this investigation have shown that a combination of Java Programming Environment, World Wide Web, a middle-ware bridging the gap between Java programs and a data base, and a database server can be used to provide the persistent data storage for a distributed Software Development Environment. Using proven technology, in the form of existing software as a foundation for building software aimed at achieving specific objectives, is advantageous as opposed to developing the software from scratch. The former avoids expending time and energy on the software which is not of direct use to a particular project but must be developed in order to serve as a foundation on which the software, to be used in the project, would be built. The complex data which is often generated in a Software Engineering Environment and must be persistently stored and managed could be successfully handled by using the approach developed during this investigation. By using this approach we have successfully built the basic infrastructure required for building various functionality required for making the data, to be generated in WHERE project, persistent and managing it. The following section points out the future work that will be required to build various functionality on top of the infrastructure developed during this investigation.

The first step in the future work, aimed at building functionality on the top of the infrastructure developed during this investigation, will be to define a process governing the operation of the WHERE project. With the basic infrastructure in place, any further development should be governed by the process. This process will specify various policies to be used in the WHERE project and these policies will govern the decisions which will have to be made to carry out the further development. For instance, it must be decided whether the inter-document consistency checks are to be performed on the client or the server side and further development carried out accordingly. Once these decisions are made, further implementation then can be carried out using the basic mechanisms developed during this investigation. For instance, mechanisms are already in place to enforce the security restrictions required for allowing controlled multi-user access to the persistent tcmJava documents. However, the security mechanism to be actually used in the real project can only be implemented after the policy for it is specified which will be done as a part of defining the process governing the over all operation of the project.

REFERENCES

Barghouti, N.S. and G.E. Kaiser. 1991. Concurrency control in advanced database applications. ACM Computing Surveys 23(3): 269-317.

Bentley, R., W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkel, J. Trevor, and G. Woetzel. 1997. Basic Support for Collaborative Work on World Wide Web. Int. J. Human Computer Studies, Academic Press, Cambridge. In Press. 20 pp.

Bernstein, P.A. 1987. Database system support for software engineering. *In* Proc. 9th Int. Conf. Softw. Engg., Monterey, Cal. pp. 166-178.

Boem, B. 1984. Model and metrics for software management and engineering. IEEE Comp. Soc. Press, pp. 4-9.

Boem, B. 1987. Industrial software metrics top 10 list. IEEE Softw. 4(5): 84-85.

Borenstein, N. and N. Freed. 1992. MIME (Multipurpose Internet Mail Extensions): Mechanisms for specifying and describing the format of Internet message bodies. RFC 1341, USC/Information Sciences Institute.

Callahan, J. and S. Ramakrishnan. 1996. Software project management and measurement on the World-Wide-Web (WWW). Proc. WET ICE '96, Stanford, California, USA. IEEE Comp. Soc. Press. 156-161.

Dix, A. 1996. Challenges and Perspectives for cooperative work on the Web. Proc. ERCIM Workshop on CSCW and the Web, Sankt Augustin, Germany.

Emmerich, W. and W. Schafer. 1996. Environments for group-oriented software design - The Groupie experience.

Emmerich, W., W. Schafer, and J. Welsh. 1993. Databases for Software Engineering Environments – The Goal has not yet been attained. pp. 145-162. *In* I. Sommerville and M. Paul (eds.), Software Engineering ESEC '93. Proc. 4th European Software Engineering Conference, Garmisch-Partenkirchen, Germany, volume 717 of Lecture Notes in Computer Science, Springer.

Engels, G., C. Lewerentz, M. Nagl, W. Schafer, and A. Schurr. 1992. Building integrated software development environments. - Part 1: Tool specification. ACM Trans. Software Engineering and Methodology 1(2): 135-167.

Evans, E. and D. Rogers. 1997. Using Java applets and CORBA for multi-user distributed applications. IEEE Internet Computing, IEEE Computer Society. pp. 43-55.

Finin, T. et al. 1992. Specification of the QKML agent-communication language. Tech. Report EIT TR 92-04, Enterprise Integration Technologies, Palo Alto, CA.

Finkelstein, A., J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. 1992. Viewpoints: A framework for integrating multiple perspectives in system development. Int. J. Softw. Eng. Knowl. Eng. 2(1): 31-57.

Finkelstein, A., D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. 1994. Inconsistency handling in multi-perspective specifications. Trans. Softw. Eng. 20(8): 569-568.

Gotel, O. and A. Finkelstein. 1994. An analysis of the requirements traceability problem. IEEE Int. Conference on Requirements Engineering, Colorado Springs, IEEE Comp. Soc. Press. 94-101.

Gotel, O. and A. Finkelstein. 1995. Contribution Structures. IEEE International Conference on Requirements Engineering, York, UK, IEEE Comp. Soc. Press. 100-107.

Habermann, A.N. and D. Notkin. 1986. Gandalf: Software development environments. IEEE Trans. Softw. Engg. 12(12): 1117-1127.

Hamilton, M. 1996. Java and the shift to Net-centric computing. Computer 29(8): 31-39.

Holtman, K. 1996. The futplex system. *In* U. Busbach, D. Kerr, and K. Sikkel (eds), ERCIM Workshop on CSCW and the Web, Sankt Augustin, Germany, GMD/FIT.

Hoover, R. 1987. Incremental graph evaluation. Ph. D. thesis, Cornell University, Dept. Computer Science, Ithaca, NY, USA. Technical Report No. 87-836.

Johnson, G.F. and C.N. Fisher. 1982. Non-syntactic attribute flow in language based editors. pp. 185-195. *In* Proc. 9th Annual ACM Symp. on Principles of Programming Languages. ACM Press.

Johnson, P. 1996. Egret: A framework for advanced CSCW applications. ACM Softw. Eng. Notes 21(2).

Johnson, P. 1994. Supporting technology transfer of formal technical review through a computer supported collaborative review system. *In* Proc. Fourth Int. Conference on Software Quality, Reston, VA, USA.

Johnson, P. and C. Moore. 1995. AEN Home Page. WWW HREF = "http://www.ics.hawaii.edu/csdl/aen.

Kotonya, G. and I. Sommerville. 1996. Requirements engineering with viewpoints. Softw. Eng. J. 11(1): 5-18.

Kramer, J. and A. Finkelstein. 1991. A configurable framework for method and tool integration. Proc. of European Symp. on Softw. Development Environments and CASE Technology, Konigswinter, Germany. Springer-Verlag.

Nodine, M.H., A.H. Sakarra, and S.D. Zdonik. 1991. Synchronization and recovery in cooperative transactions. *In* Implementing Persistent Object Bases – Principles and Practice – Proc. 4th Int. Workshop on Persistent Object Systems.

Nuseibeh, B. and A. Finkelstein. 1992. Viewpoints: A vehicle for method and tool integration. Proc. Int. Workshop on Computer-Aided Software Engineering (CASE '92), Montreal, Canada. IEEE Comp. Soc. Press. 50-60.

Nuseibeh, B., A. Finkelstein, J. Kramer. 1993. Fine-grain process modeling . pp. 42-46. *In* Proc. 7th Int. Workshop on Software Specification and Design (IWSSD-7), Redondo Beach, California, USA. 42-46.

Nuseibeh, B., J. Kramer, and A. Finkelstein. 1993. Expressing the relationships between multiple views in requirements specification. Proc. 15th International Conference on Software Engineering (ICSE-93), Baltimore, USA, IEEE Comp. Soc. Press. 187-200.

Nuseibeh, B., J. Kramer, and A. Finkelstein. 1994. A framework for expressing the relationships between multiple views in requirements specification. Trans. Softw. Eng. 20(10): 760-773.

O2 Technology. 1997. Java Relational Binding User Manual. 3600 West Bayshore Road – suite106, Palo Alto, CA, USA.

Orfali, R., D. Harkey, and J. Edwards. 1996. CORBA Services: Persistence and object databases. pp. 140-159. *In* The Essential Distributed Objects Survival Guide, Johnson Wiley & Sons, Inc., New York, USA.

Peuschel, B., W. Schafer, and S. Wolf. 1992. A knowledge-based software development environment supporting cooperative work. Int. J. Softw. Engg. Knowl. Engg. 2(1): 79-106.

Pu, C., G. Kaiser, and N. Hutchinson. 1989. Split transactions for open-ended activities. pp. 26-37. *In* Proc. 14th Int. Conf. Very Large Databases, Morgan Kaufman.

Roman, G.C.R. 1985. A taxonomy of current issues in requirements engineering. IEEE Computer 18(4): 14-21.

Taylor, R.N., R.W. Selby, M. Young, F.C. Belz, L.A. Clarce, J.C. Wileden, L. Osterweil, and A.L. Wolf. 1988. Foundations of the arcadia environment architecture. ACM SIGSOFT Softw. Engg. Notes 13(5): 1-13. *In* Proc. 4th ACM SIGSOFT Symp. Software Development Environments, Irvine, CA, USA.

Toye, G., J. Tenenbaum, M. Cutkosky, J. Glicksman, and L. Leifer. 1994. SHARE: A methodology and environment for collaborative product development. Post-Proc. IEEE Infrastructure for Collaborative Enterprises (CDR-TR) #19930507. 16 pp.

U.S. Government Accounting Office. 1979. Contracting for Computer Software development: Serious problems require management attention to avoid wasting additional millions. FGMSD-80-4.

van Welie, M. and A. Elins. 1996. Chatting on the Web. *In* U. Busbach, D. Kerr, and K. Sikkel (eds), ERCIM Workshop on CSCW and the Web, Sankt Augustin, Germany, GMD/FIT.

Visigenic Software, 1997. Visibroker for Java: Programmer's Guide Versions 3.0. HREF http://www.visigenic.com.

Wan, D. and P. Johnson. 1994. Experiences with CLARE: A computer-supported collaborative learning environment. Int. J. Human-Computer Systems.

Wieringa, R. 1996. Requirements Engineering: Frameworks for Understanding, John Wiley & Sons, Chichester, UK.

Yourden, E. 1996. Java, and the Web, and the software development. Computer 29(8): 25-30.