RICE UNIVERSITY

Integrating Document Control in Design Drawings

by

Arun Kumar Srinivasan

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

Master of Science (M.S.)

APPROVED, THESIS COMMITTEE:

Dr. Michael Terk, Chair, Asst. Professor

[Signature]

Dr. Satish Nagarajaiah, Assoc. Professor

[Signature]

Dr. Willy Zwaenepoel, Professor

HOUSTON, TEXAS

JANUARY, 2003
JANUARY, 2003

ABSTRACT

Integrating Document Control in Design Drawings

by

Arun Kumar Srinivasan

In order to manage large construction projects efficiently, the contractor generally splits a project into a group of inter-related design tasks and assigns them to different organizations through sub-contracts. Managing the information about how the components of the design change with time, making sure that none of the constraints created by the inter-dependency of design components are violated in the final design is a challenging task. As a result, intensive research has been conducted in the area of Document Control to ensure the integrity of the design throughout the duration of the project.

Most of the present efforts in Document Control require active participation of the user in informing a server about the design drawings that he uses. This research provides a Version Control mechanism embedded into design drawings and shows how a similar mechanism can improve communication between designers in a project.
ACKNOWLEDGMENTS

The author wishes to thank his project advisor, Dr. Michael Terk for his constant guidance and encouragement. Without his support, it would have been impossible to carry out all the work mentioned in the project.

I would also like to express my gratitude to Dr. Satish Nagarajaiah and Dr. Willy Zwaenepaul for agreeing to be on my thesis committee. I would also like to thank Dr. Veletsos for his support and interest during the course of the project.

Last but not the least, I would like to thank my family and friends for their constant support and help that they have lent me in times of need. Prasanna Parthasarathy, Mukund Srinivasan, Prakash Kripakaran and Bhanu Nagendra were very helpful even though I had to disturb them with questions and doubts. I am very grateful to them for their help and support.
TABLE OF CONTENTS

Table of Contents ........................................................................................................ iv
List of figures .................................................................................................................. vi
CHAPTER 1: INTRODUCTION ......................................................................................... 1
  1.1: OBJECTIVE OF THE RESEARCH .......................................................................... 4
  1.2: OUTLINE OF THESIS ........................................................................................... 5
CHAPTER 2: BACKGROUND ............................................................................................ 7
  2.1: DOCUMENT CONTROL ......................................................................................... 7
  2.1.1: Introduction to Intellectual Property ................................................................. 8
  2.1.2: Introduction to Version Control (Terminology) .................................................. 10
  2.1.3: Introduction to Configuration Management (Terminology) ............................... 13
  2.2: PREVIOUS RESEARCH IN CM AND VC .............................................................. 17
  2.3: ISSUES WITH EXISTING DOCUMENT CONTROL SYSTEMS ............................ 23
  2.4: MOTIVATION ....................................................................................................... 24
  2.4.1: Support for floating copies ............................................................................... 24
  2.4.2: Reduce designer’s interaction with the Document Control server ..................... 24
  2.4.3: Enhanced communication and collaboration ...................................................... 24
  2.4.4: Support for Intellectual Property ...................................................................... 25
CHAPTER 3: PROPOSED APPROACH ......................................................................... 26
  3.1: FLOATING COPIES ............................................................................................. 26
  3.2: RECENT TECHNOLOGY THAT PROVIDE SOLUTIONS ..................................... 27
  3.3: PHILOSOPHY ....................................................................................................... 31
  3.3.1: Supporting Floating Copies ............................................................................. 32
  3.3.2: Automating Client-side Document Control Interaction ................................... 34
  3.3.3: Communication and Collaboration .................................................................. 34
  3.3.4: Intellectual Property Control .......................................................................... 35
  3.4: REVIEW OF APPROACH ....................................................................................... 37
CHAPTER 4: IMPLEMENTATION ................................................................................ 40
  4.1: IMPLEMENTING CLIENT-SIDE AUTOMATION OF VERSION CONTROL .............. 40
  4.2: VERSION CONTROL SERVER ............................................................................. 44
  4.3: IMPLEMENTING ENHANCED COMMUNICATION & COLLABORATION ........... 48
CHAPTER 5: CONCLUSION .......................................................................................... 55
  5.1: CONTRIBUTION OF THIS RESEARCH ............................................................... 55
  5.2: ISSUES & LIMITATIONS ..................................................................................... 56
  5.2.1: Security issues when using macros ................................................................. 56
  5.2.2: Issues with the approach ................................................................................. 58
  5.2.3: Issues with the implementation ........................................................................ 60
  5.3: AUTOCAD WISHLIST ......................................................................................... 61
  5.4: SCOPE FOR IMPROVEMENT .............................................................................. 62
5.5: Conclusion ...........................................................................................................64
BIBLIOGRAPHY .........................................................................................................66
APPENDIX A: SAMPLE CLIENT-SIDE CODE (VBA) .........................................................69
APPENDIX B: NOTE TO ADMINISTRATORS ..................................................................72
APPENDIX C: SUPPORTING EXISTING DOCUMENT CONTROL SYSTEMS ..............75
<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 2.1</td>
<td>Version History Graph, Currency (From Katz, 1986)</td>
<td>19</td>
</tr>
<tr>
<td>Fig. 2.2</td>
<td>Object and version planes (From Westfechtel, 1996)</td>
<td>22</td>
</tr>
<tr>
<td>Fig. 4.1</td>
<td>Server Menu Page displaying list of existing Projects</td>
<td>45</td>
</tr>
<tr>
<td>Fig. 4.2</td>
<td>Form to create a new project and specify its components</td>
<td>47</td>
</tr>
<tr>
<td>Fig. 4.3</td>
<td>Server displays information about a project and providing links to download template drawings and current versions of its components</td>
<td>48</td>
</tr>
<tr>
<td>Fig. 4.4</td>
<td>Menu created by macro to provide communication functionality</td>
<td>50</td>
</tr>
<tr>
<td>Fig. 4.5</td>
<td>Text box to send commands to execute through chat interface</td>
<td>52</td>
</tr>
<tr>
<td>Fig. 4.6</td>
<td>Request to execute command received through chat interface</td>
<td>52</td>
</tr>
<tr>
<td>Fig. 4.7</td>
<td>Posting message in message board - Specifying receiver list</td>
<td>54</td>
</tr>
</tbody>
</table>
CHAPTER 1: INTRODUCTION

Most large construction projects are too complex to be managed and developed by a single person or organization. In order to develop such a project efficiently, the developer hires a contractor who organizes the project into a group of inter-related design components. These design components represent the functional decomposition of the project based on the various disciplines such as architecture, structural engineering, and electrical engineering involved in the project.

The contractor assigns the task of developing each design component to sub-contractors, who in turn assigns a group of his engineers to work on various aspects of the design component. The design of each component is further decomposed into smaller units of design and each member of the group of engineers is allocated to the design of one or more aspects of the design component. The group of engineers comes up with a solution that is represented as a set of design drawings. The sub-contractor ensures that the design drawings submitted by the engineers satisfy the requirements of the design and sends the drawings to the contractor.

Due to the inter-related nature of the project decomposition, updating any component of the design can have an impact on the constraints governing the design of other components, not only within that discipline but also across disciplines. Therefore, over the various stages of the project, the design drawings are constantly modified and updated.

* English does not contain a convenient unisex pronoun: we shall use 'he', 'him' and 'his', but this should not be taken to mean that we think roles are constrained by gender. We used 'he/she' in our early drafts, but it is a very tedious business both for writing and reading.
When a designer has finalized his design drawing as an improvement over the previous design, he submits the drawing, through the sub-contractor, to the contractor, who has to ensure the integrity of the complete design before accepting the drawing. Once it has been verified that the new drawing is consistent with the rest of the design, it is necessary to make the improved design available for the other designers and to notify all designers working on the affected components about the availability of the updated design. This notification also ensures that all designers working on components that are related to the updated component are made aware of the changes that may affect their design. In cases where the new design has potential to impact the entire project design, it might be necessary to notify designers working on all components of the project.

The continuous creation and modification of design drawings results in a cyclic design process, resulting in the generation and modification of a large number of design drawings. Managing the information about how the overall design changes with time, making sure that none of the constraints created by the inter-dependency of designs from various disciplines are violated, and notifying the involved designers about updates or errors is a challenging task. As a result, intensive research has been conducted over the past two decades in the field of Document Control aimed at ensuring the integrity of the design throughout the duration of the project and providing better communication between designers.

The technological advances in the field of electronics and computers have provided new possibilities and solutions to handling Document Control. However, technology has also contributed to the rise of new issues that also need to be addressed by Document Control systems.
Over the last couple of decades, the advent of computers has provided designers with increased computational power. This computational power has allowed designers to work on complicated projects that, without the help of computers, would be impossible. Computers help in the analysis and design of the projects. Further, computers are also useful in managing projects and supporting the various tasks involved in managing and tracking the documents generated as part of the development process.

Simultaneously, the increased use of the Internet through networking technology in computers has added to the complexity of the Document Control process by enabling the project managers to reach a more widespread audience. This meant that a project could have engineers from all over the world working on the various components involved in the project. However, the benefits of this approach are sometimes negated by the added potential for delays in communication between the various members participating in the design of a project. In addition, designers often lose a lot of time searching for specific documents and information. Even within the realm of an individual organization, the ease with which designers can transfer design drawings between one another creates additional challenges in keeping track of changes to the design and of the people who are working on the various versions of the design drawings.

Therefore, as the functional and geographical distance between the participating companies increases, the task of providing Document Control becomes even more complicated. In this respect, technological advances have, in addition to providing more powerful solutions, created new challenges in the realm of Document Control that need to be addressed.
1.1: **OBJECTIVE OF THE RESEARCH**

The most critical aspect of Document Control is keeping track of all the design drawings generated and used as part of the development process. The concept of autonomous agents [26] presents a possible solution to address this issue. Autonomous agents refer to programs that act on behalf of another program or the user by monitoring the situation and reacting to various changes in the environment around it. The research aims to enable design drawings to act as autonomous agents that can perform Document Control operations on behalf of the designer.

In order for design drawings to act as autonomous agents, the design drawings should contain embedded knowledge that allows it to monitor its state and the environment in which it is being accessed. Then, the design drawings can make decisions based on its knowledge and provide a mechanism to track the generation and evolution of the documents. By achieving this, the perception of a drawing can be changed from a passive representation of a design solution to that of an active design representation that would be responsible for automating the tasks of Document Control on behalf of the designer.

In order to have a drawing act as an autonomous agent, one possible approach is to take advantage of macro programming capabilities in application software. Macros represent pieces of program code that can be used to automate tasks within an application. Many modern applications allow users to embed macros within a document. Macros that are stored within a document are called embedded macros or document macros. By using embedded macros, the document can contain both the design data as well as some program code that can provide
behavioral aspect to the document. As a result, these macros can act as autonomous agents and provide a mechanism to track the generation and evolution of the design drawings.

The objective of the research is to develop a new approach to provide the project management team and the team of designers with a mechanism by which they can achieve Document Control for their projects with minimum manual participation. This approach uses embedded macros to provide client-side functionality that automates communication with a Document Control server. Further, this objective of the research is also to develop a sample implementation that will highlight the benefits of the proposed approach.

Finally, this research explores the additional possibilities created by the proposed approach to Document Control in enhancing communication and collaboration between designers working on the same project component.

1.2: OUTLINE OF THESIS

The remainder of the thesis is organized as follows. In the next chapter, the background information (terminology and previous research) is discussed. Further, the issues with the existing approaches and the motivation behind the present research is also discussed in detail.

Chapter 3 presents a new approach to address some of the issues that are not addressed by previous research approaches, discussing the philosophy and methodology involved. Chapter 4 describes a sample implementation that highlights the benefits of the proposed approach.

Finally, chapter 5 discusses the issues that remain as a significant barrier to the wide adoption of this approach and also identifies some of the limitations of this approach and
implementation. Further, it discusses the contribution of the research and proposes directions for future work.
CHAPTER 2: BACKGROUND

This chapter provides background on the terminology used by researchers in the field of Document Control. Further, it describes the previous research approaches in the field of Document Control, discusses their shortcomings and states the motivation for this research.

2.1: DOCUMENT CONTROL

In many present-day non-integrated systems environment [32], technical personnel spend a significant amount of time looking for data, drawings and technical specifications required to do their work. This often results in reduced work quality and extensions of deadlines. As the world markets get more competitive, high quality, reduced cost and rapid delivery times are essential to success of a design organization.

*Document Control* refers to the set of mechanisms used to track and support inquiries about the status of various design drawings and notify people working on affected project components when a design change is initiated. Document Control allows organizations to manage the creation, evolution and the archiving of a project and its components. While the term Document Control is relatively new, it can be considered to be a combination of three well-defined concepts: Intellectual Property, Version Control and Configuration Management.

Intellectual Property refers to something that is the result of an individual’s creativity that he can “own”. The owner of an Intellectual Property can therefore control and be rewarded for its use, and this encourages further innovation and creativity to the benefit of everybody. In the digital realm, Intellectual Property applies to either an application software or a data file. When it comes to the protection of software, the owner of an Intellectual Property often
obtains a copyright. Further, the owner can protect the distribution of the software by providing serial number checks and/or hardware locks. However, there is no clear mechanism to support the concept of Intellectual Property for data.

The other two concepts in Document Control, namely Version Control and Configuration Management, are inter-related. Version Control refers to the concept of tracking the different designs of a project component as they are created, modified and copied by and between designers. Configuration Management refers to the mechanism used to ensure the integrity of the overall design by tracking interdependencies between the various components of the final design.

2.1.1: Introduction to Intellectual Property

The concept of Intellectual Property allows individuals and companies to protect the products of their creativity by enabling the owner to specify restrictions on accessing their product. Intellectual property allows people to own their creativity and innovation in the same way that they can own physical property. Intellectual property refers to something that is the result of an individual's creativity that, thereby, allows the individual to be able to obtain a profit by providing it as a service to the others, simultaneously preventing the others from legally modifying it and marketing it without the consent of the individual. The owner of an Intellectual Property can therefore control and be rewarded for its use, and this encourages further innovation and creativity to the benefit of everybody. The four major types of Intellectual Property are:

- Patents (for inventions),
• Trademarks (for branded goods),

• Copyrights (for material), and

• Designs (for product appearance).

While the concept of Intellectual Property is not new, the advent of computers has created additional issues that need to be addressed. In the digital realm, Intellectual Property refers to application software and data files. The concept of Intellectual Property is covered for programs/applications through copyrights. Further, the owner of the Intellectual Property can control its distribution by providing hardware locks and serial number checks.

In the case of data files, the owner of the files can specify reading and writing restrictions for other users. Therefore, for data files, Intellectual Property control might refer to one of two possible situations. By specifying that the user can read the file but not write to it, the owner can allow others to view the file and even make a copy of it, while preventing the file from being modified by the others. On the other hand, by specifying that the user can neither read the file nor write to it, the owner can prevent other users from even accessing the file.

The issue of Intellectual Property for data files is very important in design organizations. For instance, a company might want to have certain documents available only to those people who work with them, or perhaps to only one division of their company. In most existing Document Control systems, the access to documents is protected either by specifying access rights or by providing a password. However, once a document has been accessed and transferred as a local copy onto a designer’s computer, the task of controlling access to the
document moves from the owner of the document to the designer who obtained it.
Currently, there is no mechanism to provide control over the access of the local copies of any
document.

2.1.2: Introduction to Version Control (Terminology)

Having discussed the concepts involved with maintaining the security of accessing the
design drawings, this section discusses the concepts involved in Version Control, which are
used to manage the generation and evolution of the design drawings involved in a project. The
definitions for the terms introduced in this section are obtained from references [2], [14], [16],
[20] and [25].

As mentioned earlier, each project is divided into a group of components, which are
assigned to various companies through sub-contractors. Within each company, a group of
engineers work to come up with designs for components assigned to them. In this process,
they aim to provide the best designs that conform to the requirements of the project and
maintain the integrity of the design. Work begins based on a set of design requirements and
assumptions. As the design progresses, some of these assumptions may become invalidated. At
each such occurrence, contractor must review the affected parts of the design to determine if
they should be redesigned. If the contractor determines that certain components of the design
need to be redesigned, he notifies the designers working on the affected components about the
new requirements and restrictions. This results in a series of design cycles as the designers of
each component come up with designs that both conform to the requirements of the project
and represent an improvement over the previous design. Therefore, as the project progresses,
each design team produces many variations in the design of each of their components. Each instance of a component created in this process is referred as a *Version* of that component.

There are two differing philosophies about what constitutes a new version. One school of thought defines a version uniquely by its location. This means that each individual physical representation of a particular component is considered a version. The second school of thought defines a version uniquely by its location and timestamp. In this approach, a mechanism is provided to keep track of the changes made as the design of each version progresses. While the second approach encompasses the first, it is debatable as to how much advantage is obtained by adding a time component. While tracking the changes that are made can occasionally provide designers with the ability to revert back to some stage, it often leads to storage space issues. The main cause of the increased need for storage space is that no difference is made between major and minor changes. During the design progress, there is unquestionably going to be numerous changes but many of them are trivial. Storing all this information can consume a lot of space, while providing limited benefits.

When a version A is created based on version B, the version A is considered the *child* of version B, while version B is considered the *parent* of version A. While the version A might be created based on more than one version (say versions B, C and D), it starts off by obtaining a copy of one of those versions (version B) and making modifications to it based on the other versions (versions C and D). In such situations, only B is considered as the parent of A. It is clear therefore that while a version can have more than one child, it can have only one parent. When a designer makes some changes in the design of a component, he may want to inform the designers who have obtained the drawing from him, and the designer from whom he
obtained it. However, keeping track of the designers who have obtained the drawing from him is tedious. Therefore, it is necessary to have a mechanism that can keep track of versions, the designers using them and how the versions are related to each other.

The term Version Control (VC) refers to the set of concepts and mechanisms used to keep track of versions of a project component. Version Control consists of two major tasks: to model the structure of a component's evolution, and to store information about the history, state and contents of each version of the component. In order to model the evolution of a component in terms of its versions, most systems use a graphical representation in which the versions are connected together by lines that depict the parent-child relationship between versions. This graphical representation of the design history of any project component is called the version derivation graph of that component. The version derivation graph makes it possible to quickly determine the history of a particular version of a design component and identify all the versions that have been derived from it.

One of the basic concepts in VC is that of version status or version state. The status of a version can be broadly classified as either temporary or permanent. A temporary (transient) version is a version of a design that is either incomplete or waiting for verification/approval from the contractor, and therefore may be modified. A permanent (working) version is a version of design that has been determined by the contractor to be consistent with other design drawings. Thus, a permanent version is considered stable and cannot be modified. However, a permanent version may become obsolete due to newer permanent versions, and if the available storage space becomes depleted, it may be deleted to make space for newer permanent versions.
A permanent version can be further sub-classified as released, frozen and current. Once a permanent version is made available to all, it becomes a released version. A released version may be frozen if it is important that it should not be deleted. A frozen version can neither be modified nor be deleted, until its status is changed. This is useful when the contractor decides that he needs to keep that version around for some time. When a designer starts working on the design of a project component, he can either work on one of his own previous versions, or he can obtain a version from one of his co-workers, or he may obtain a released version. Often, the contractor prefers to have the designers start off from the most up-to-date released version. In such a situation, he specifies such a version as the current version of the component. Among the released versions, only the current version is then made available to the designers. This helps prevent excessive branching of the version derivation history.

2.1.3: Introduction to Configuration Management (Terminology)

This section discusses the concepts that are used to ensure the integrity of the design as the project proceeds. The definitions for the terms introduced in this section are obtained from references [2], [3], [5] and [25].

Given the complexity of a typical construction project, the task of producing a design is divided into various sub-tasks for each of the different disciplines and assigned to different project groups. Because the final design consists of a combination of smaller components, it is referred to as a composite object. These sub-tasks are thereby referred to as its component objects. Each component object can, in turn, be a composite object itself if it is further decomposable. The lowest elements in the decomposition tree of a project represent components that cannot be decomposed any further and are called primitive objects.
The information regarding which version of each of the components is used to construct a complete design is called a Configuration of the design. As the design of a project component progresses, existing constraints get modified and additional constraints are created on the designs of the other project components. When a designer updates a component of the design, he submits the design to the contractor to verify that it is consistent with the design of the rest of the components. The mechanisms that are used to ensure that none of the constraints imposed by any of the components are violated by the other components of the project while creating new configurations is called as Configuration Management (CM).

In CM, the terms workspaces, check-ins and check-outs define important concepts that support controlled access to components and configuration data within the project environment. The terms check-in and check-out represent the operational concepts of copying to- and from- a particular storage space, respectively. In order to be able to differentiate between the personal efforts of an individual designer, the collective work of a group of designers and the completed designs that have been approved, it is necessary to define separate storage spaces, called workspaces, with different access restrictions for checking in and checking out versions/designs. Katz defines a workspace as a named repository of design objects (drawing files, etc.) from which users and design applications can access the objects. Three kinds of workspaces are defined – Private, Archive and Group.

The Private workspace is created for individual users to store their personal efforts. Only that particular user, who is called the owner of that workspace, has access to the versions of the objects in here. This means that only the owner can define the levels of security for checking-in and checking-out version to and from his workspace. Often, he might allow a specific group
of co-workers to check-in or check-out versions. This workspace allows a designer to experiment with the design of a component before sending it to the contractor for verification. The drawings in these workspaces are usually temporary versions as the drawings are yet to be verified. Once the owner is confident that his version of the design has some advantages over the previous design, he can submit it to the contractor for verification.

When a designer submits a version of a component design to the contractor, the contractor has to verify that using the new version will not compromise the integrity of the final design. Once the contractor has determined that a particular design drawing is consistent with the rest of the configuration, and that it is an improvement over the previous design, he has to make that drawing available to the other designers. The Archive workspace represents the public repository where released versions are stored. The versions stored here can be accessed by all but cannot be modified. Unless they are frozen, the versions stored here may be deleted by the contractor in case of storage requirements. All designers can check-out any version stored here. However, only contractors can check-in a new version after carefully ensuring that the version is consistent with the latest designs from the other project groups.

Occasionally, a group of designers might decide to work together on a drawing. By posting comments about the changes made, one can inform the other members of the group as to the current status of the drawing. The Group workspace is a storage space for designers to share incomplete or partially verified versions, so that a group of designers can work on it together and can use versions of other components within the same workspace to verify the consistency of the design. The characteristics of a group workspace are similar to that of a private workspace except for the fact that the number of owners is no longer restricted to one.
The versions stored here are usually classified as temporary. Only the members of the group can check-in new versions into this workspace. Similarly, only the members of this group can check-out any version here and submit it to the contractor for verification of the integrity of the design.

*Change Notification* is another important concept in Configuration Management and Version Control. Change Notification refers to a mechanism by which designers would be informed when some changes have been made to a component they are working on. Occasionally, the contractor might want to report either errors in existing design versions or might want to report updates in the design of a particular component. This is achieved through Change Notification.

Another important concept in CM is that of *equivalences*. The various disciplines involved in a construction project construct different "views" of the same design. For example, while the architectural division comes up with the layout of space within a building, the structural division provides the details that support the building. The various representations of the same object in the various disciplines involved in the project are called equivalences. Usually, the design of one object results in creating constraints on the design of its equivalents.

A useful concept in Configuration Management especially in the preliminary stages of design is that of *dynamic configuration binding*. Configuration binding refers to the concept of specifying which version of each component object is to be combined to obtain the configuration of a composite object. In the early stages of design, it might be useful to leave the actual version of each component unspecified until the actual process of building the
composite object. When a new configuration is being built, a series of rules are specified that allow the system to identify which version of each component is to be chosen for that particular configuration. This idea is called dynamic configuration binding. The advantage of this method is that the contractor can generate and compare multiple configurations at a time easily by tinkering with the set of rules.

The concept of dynamic configuration binding can be achieved through layering. The basic idea in layering is to place portions of the version histories for multiple related objects into layers (grouping versions of various components) and to provide contexts (the set of rules) that order the layers. If two versions of different objects must be used together, they are placed in the same layer. The context defines the search order over the layers, thereby defining the versions to be chosen at run-time.

2.2: PREVIOUS RESEARCH IN CM AND VC

A number of researchers have conducted research in Document Control over the past couple of decades, especially in the areas of software configuration management and Computer Aided Design (CAD). Several guidelines and frameworks have been defined for the implementation of Configuration Management and Version Control.

Katz was one of the earliest and most influential researchers who helped define the various aspects that need to be considered while building a model to support CM and/or VC. As early as 1986, Katz [2] proposed the concept of a centralized repository that would store information about versions and provide a mechanism to track these versions. He complemented this concept by building a Version Server to support CM and VC concepts. Using the Version Server, Katz organized the design into an archival hierarchical description
across representations and maintained the versions of the various design portions. Further, the Version Server supported CM concepts such as workspaces to permit individual changes to be shared in a controlled way and implemented the “careful” update of the archive workspace. Configurations, versions and equivalences were represented in three planes. Configurations and versions were represented in two orthogonal planes – *hierarchical composition* and *version history*, respectively. An *Equivalence plane* allowed multiple representations for the same object. The composite object and the components in the configuration plane were represented by “*is-a-part-of*” relationships between objects, for e.g., a component is-a-part-of a composite object. The parent and child versions in the version history were represented by “*is-a-kind-of*” and “*is-a-child-of*” relationships, i.e., the child is-a-child-of the parent version. The different representations for the same object in the equivalence plane were represented by “*is-equivalent-to*” relationships, for e.g., the structural design of a building is-equivalent-to the architectural design of the same building. Further, a version derivation graph was implemented to represent the inter-relation of versions of the same object. The version histories were represented by acyclic graphs, using arrows to indicate is-a-child-of relationship. Support for current versions and for dynamic configuration binding through layering were also provided. The equivalences presented themselves in the form of constraints that must be validated before the design could be considered correct. Depicted below (Fig. 2.1) is a simplified look at the version derivation graph with currency control. Initially, $v[0]$ is the current version. After currency moves to $v[2]$, no further versions can be derived from $v[0]$, $v[1]$, $v[3]$, or $v[4]$ without repositioning the currency. As a result, $v[5]$, $v[6]$ and $v[7]$ are all derived from $v[2]$ either directly or indirectly. However, the contractor might still prefer to retain $v[2]$ as the current version for that component.
Hong-Tai Chou and Won Kim [25] (1986) attempted to provide a comprehensive framework for version control in an integrated CAD environment. They proposed the use of 3 databases - public, private and project, to separate information logically, and that versions be classified as transient, working or released. Transient versions would exist in private databases while released versions would exist in public databases. Working versions represented an intermediary stage and could exist in either the private database or the project database. In order to incorporate the distributed nature of a CAD environment and the complex configuration of CAD objects, they proposed different set of capabilities between the databases, especially in sharing files. Further, they proposed two mechanisms for Change
Notification. They suggested that the notification of change/uploads could be either message-based or flag-based. In the message-based approach, the system sends messages to notify the users of potentially affected versions. The message-based approach is further distinguished as immediate or deferred, based on whether the users are notified immediately after the changes to a version are committed or at some later time that the users may have specified. In the flag-based approach, the system simply updates data structures that it maintains so that the affected users will become aware of the changes in a version only when they explicitly access the version. In this sense, the flag-based approach is necessarily a deferred notification strategy. Further, they suggested that the notification might not only have to consider all the versions directly referencing the particular object but also include those that indirectly reference it.

Kafer and Schoning [20] (1992) presented a version model that would provide support for version control of complex CAD objects in database systems. They suggested that while creating a version model to map a complex object, functionality should be provided to create a new object and specify its attributes, add a new version, update information about existing versions and retrieve information and versions. In order to support the version history graph proposed by Katz and represent a version model in a Database Management System, they proposed a Molecule Atom Data model to map a version model to a complex-object data model. In this model, the basic building blocks, called Atoms, are created based on atom types, and are defined by a set of attributes. Comparing this to the model of a programming language, an atom type would be similar to a variable type such as an integer or character. An atom would then be equivalent to an instance of such a variable type, i.e., a variable, with the value
representing its attribute. A molecule type and a molecule can then be defined by specifying a coherent grouping of atom type network and assigning directionality to linking between atoms.

Talens, Ouassalah and Colinas [16] (1993) proposed a model of versions to manage simple and composite objects. They suggested that some of the important data about a version that might be needed include the number of the next version, current version of the object, state (Permanent or Temporary) of the version, derivation hierarchy and the versions that have been derived from it. The number of the next version would be used to provide the identifier when a new version is derived from an existing version. The contractor defines the current version of the object so that the designers can derive their versions from it. The state of a version defines whether the version can be modified or deleted. The derivation hierarchy and the versions that have been derived from a version allow a designer to keep track of the evolution of a version.

Bernhard Westfechtel [7] (1996) developed a domain-independent and adaptable configuration management model called CoMa to manage engineering design documents. The paper proposed that the versions, configurations and the document groups be represented in graph schema, with transactions and control structures being modeled as complex graph transformations. The paper depicted how the model could be adapted from a domain-independent generic model to a specific domain to obtain a concrete model. The paper further defined certain constraints for consistency such as unique labels/IDs for objects and their versions, objects as part of only one document group, locality of history and dependency relations within a sub-graph and the acyclic nature of history, dependency and composition relations.
Fig. 2.2 Object and version planes (From Westfechtel, 1996)

Fine, Smith, Hwang and Turner [12] (1998) provided a Configuration Management tool, SAMS, which they used to educate scientists on the advantages of using CM to maintain integrity of research information. They proposed using Configuration Files to reliably access current versions of code and coordinate work. Each configuration file would consist of:

- A global section with general details of the project;
- Module descriptions describing the modules/components and files involved; &
- A list of other miscellaneous files necessary for the project, which are not a part of any module.
2.3: ISSUES WITH EXISTING DOCUMENT CONTROL SYSTEMS

The present efforts in CM and VC can be classified as server-oriented, based on the Katz concept of a version server. All communications are directed through a central computer, which processes the request. Thus, versions are registered only through the server check-ins and check-outs executed explicitly by the contractor or a designer. This requires the designers to diligently communicate with the server in order to keep it up-to-date.

Due to the highly elaborate network services available these days, it is very common for people to share their files and designs drawings without contacting the server. The copies of the files that are created by designers without contacting the server can be referred to as Floating Copies. The present efforts in the field of Document Control generally do not consider floating copies because the server does not have any information about the location of these drawings and the designers who access these drawings. Therefore, these floating copies are not accounted by any mechanism of Document Control including change notification and Intellectual Property control.

In addition to being unable to support floating copies, the server-oriented nature of the existing Document Control systems results in the concept of Intellectual Property being provided only at the point of initial access from the server. While the action of checking out a document from the server is often protected through various means, the Document Control server has no control over the access of the document once it has been obtained from the server. Therefore, it can be modified and copied by anyone who is able to access it. In such situations, both Intellectual Property control and the ability to track the drawings are not available.
2.4: MOTIVATION

In order to address the issues with present Document Control systems, the approach we propose can improve existing systems by providing the following:

2.4.1: Support for floating copies

It was mentioned earlier that present Document Control systems do not support floating copies. Therefore, this research aims to provide a mechanism by which a Document Control system can support floating copies. In order to be able to support floating copies, a Document Control system should be capable of identifying and tracking floating copies.

2.4.2: Reduce designer's interaction with the Document Control server

It was mentioned earlier that present Document Control systems require active client participation. Therefore, the research aims to automate the client-side interaction of Version Control systems by automating Document Control tasks.

2.4.3: Enhanced communication and collaboration

It was mentioned earlier that designers often lose a lot of time when they search for documents and project specifications. Therefore, the research aims to provide mechanisms for enhanced communication and collaboration between designers. Further, it is desired that communication be version-based rather than person-based. The majority of existing communication mechanisms are person-based, but in the area of Document Control, there are many benefits for providing version-based communication. For instance, there are many situations where a designer might need to contact people not based on their identity but based on the fact that they derived their drawings from him. In such situations, the designer does not
have information about the intended receiver(s). Therefore, version-based communication is more suitable for this situation.

2.4.4: Support for Intellectual Property

It was mentioned earlier that the concept of Intellectual Property is not supported after the document has been obtained from the server. The research aims to provide a framework to enhance this by providing Intellectual Property control every time the document is accessed.

The objective of this research is, therefore, to create an improved mechanism for Document Control that extends the existing methods of Document Control to support floating copies, reduce client-side participation, enhance communication and collaboration between designers, and to provide a framework to support the concept Intellectual Property at access-time.
CHAPTER 3: PROPOSED APPROACH

Research was conducted in order to determine how the use of agents could help to address some of the issues with existing Document Control systems. This chapter discusses the philosophy behind the research and the technology that makes it possible.

3.1: FLOATING COPIES

As mentioned earlier, floating copies refer to drawing versions created outside of a Document Control server without notification. Since the research aims to provide support for floating copies in Document Control systems, we identified the various circumstances under which floating copies can be created and the necessary functionality that a Document Control server should provide in order to consider floating copies.

Theoretically, a version represents a new significant change to the design. Therefore, it is necessary to address the issue of how one can detect when a new version is created. The first option would be to ask the user to notify the server. This is not a practical solution since it requires active participation from the designers. The second option is to check the design for significant changes to the contents. This is also not a practical solution currently because it involves “intelligent” knowledge to decide what represents a significant change. The third option is to specify that each physical instance of the design of a component represent a version. In most Document Control systems, this is the adopted approach for defining versions. For this research also, we will adopt this approach.

Under this definition of a version, floating copies can be created under various circumstances such as:
• The user copies the file;

• The user moves the file to a different location; or

• The user issues a 'Save As' command on the file.

These circumstances can occur within the same computer or between two machines within the same local network or between two machines in 2 different networks. Further, the user can make the new floating copy for his use or for someone else.

In order to take floating copies into consideration, a Document Control Server should support mechanisms that can provide the following features:

• Identify floating copies as and when they are created and accessed;

• Track floating copies along with all other versions;

• Communicate with the server about the status of floating copies;

• Provide Change Notification; and

• Intellectual Property control

3.2: RECENT TECHNOLOGY THAT PROVIDE SOLUTIONS

Until recently, there was no possible mechanism to keep track of floating copies. This was because the application, the programming languages and the data were separated. The data always represented information that changed only when the user modified it. The application always represented a medium for the user to create and modify data. The programming language represented an independent mechanism to allow the user to execute a series of
commands together. However, new technologies make it possible to integrate the programming languages with both the application and the data.

In Microsoft Word and many other recent applications, a macro is a saved sequence of commands or keyboard strokes that can be stored and then recalled with a single command or keyboard stroke. Recently, Microsoft released a new language based on Visual Basic called VBA (Visual Basic for Applications), which provides macro creation and editing capabilities within applications such as Microsoft Office and AutoCAD. This provided a mechanism by which certain macros that could be triggered as a result of an event, such as opening a document, or adding a new piece of text. In VBA, it is possible to create two kinds of macros — project macros and embedded or document macros. Project macros refer to the macros that are stored in the application, and can be called from any document. Document macros, on the other hand, refer to the group of macros that are stored within the document and are therefore specific to that document and can be triggered only with that document.

Until recently, the design data has been perceived as static information residing on the computer. With the advent of document macros, this scenario has changed. Document macros allow data files to store knowledge along with the design data and react to specific changes in its environment when the file is accessed. Thus, the perception of design data has now moved from that of static information to that of an “intelligent” dynamic entity. This allows a contractor to provide mechanisms for improved Document Control, wherein:

- The floating copies can be identified and tracked.
The design drawings can now contact the server on behalf of the user, making it convenient to update the server without the designers having to do so explicitly.

The contractor can implement a control within the document if he desires to restrict access to the data files (considering them to be Intellectual Property).

Communication and collaboration between designers working on the same project component can be improved.

When document macros are embedded within the drawings to act on behalf of the users, the macros act as agents that react to the changes of the environment. In common world terminology, the word agent generally refers to: 1) one who acts, or who can act, and 2) one who acts in place of another with permission [26]. Since "one who acts in place of " acts, the second part of the definition requires the first. In computer terminology, an agent refers to a program that can monitor the environment it exists in and respond based on its characteristics on behalf of a user or another program. Often, the term agent is used to refer to an autonomous agent, which represents a system situated within and a part of an environment, that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future. For example, consider a spell checker in a word processing software. When called upon by the user to check the spelling of a document, it is not an agent, because it is merely executing a program. On the other hand, if the spell checking were to monitor the document as it was being edited and corrected it on the fly, it can be considered as an agent because it now monitors the environment (of the document) and responds to changes in the environment.
In order to effectively manage the information involved in a Version Control system, researchers proposed to store all the information about the various projects, the components and the versions in a central repository, called the server, from which users can obtain information and the contractor can update information as the project progresses. This idea was based on one of the earliest technologies behind the advent of the Internet called Client-Server Computing [31]. There are three main aspects of client-server computing – server, client and protocol. A server refers to the computer program (and often, the computer itself) that awaits and fulfills requests sent to it from other programs in the same or other computers. The programs that send the requests to the server are called client programs, and computers that generate client programs are also usually called as clients. The protocol refers to the set of guidelines for the format in which the client should send a request and the server should reply. In client-server computing, the user at the client machine sends a request to the server from which he desires to obtain information, sending necessary details of what he wants. The server receives these requests from various clients, processes the information and sends the results back to the clients.

While existing approaches adopt the client-server computing technology, this research explores the advantages and possibilities if another Internet technology called Server Push is used. Server push involves the delivery of information on the Web that is initiated by the Server rather than the client [31]. The information pushed from a server to a user actually comes as the result of a programmed request from the client in your computer. Therefore, in order to use this technology, it is necessary to download a client program, which acts as an
agent. This program captures your profile and then periodically initiates requests for information on your behalf from the server.

Sometimes, it might be necessary for two or more client machines to communicate among one another. Consider for example, when two or more designers desire to discuss some aspect of the project design over the network. In such situations, it might be advantageous to communicate directly without having to send information through a server. Peer-to-peer is a communications model in which each party has the same capabilities and either party can initiate a communication session. On the Internet, peer-to-peer (referred to as P2P) is a type of transient Internet network that allows a group of computer users with the same networking program to connect with each other and directly access files from one another's hard drives. Instant messaging (also called IM or IMing) is the ability to easily see whether a chosen friend or co-worker is connected to the Internet and, if they are, to exchange messages with them. Conferencing and Chat are mechanisms that use IM by which users can communicate between each other. Chat generally refers to communication between two individuals while Conferencing refers to communication between many users.

3.3: PHILOSOPHY

As mentioned earlier, the objectives of this research are: to provide support for floating copies in Document Control systems; to automate client-side Document Control interaction; to enhance communication and collaboration between designers; and to provide Intellectual Property control at access-time.
3.3.1: Supporting Floating Copies

As mentioned earlier, we have decided to use file location as the primary mechanism for identifying versions. Therefore, in order to provide support for floating copies in a Document Control system, it is necessary to have a mechanism by which the computer can identify each copy of the file uniquely by its location.

In this approach, the information regarding the location of a version is stored within the drawing. By creating a mechanism to trigger events when a document is accessed (see 3.3.2), the stored information can be compared with the current location of the document every time the document is accessed. If the location of the document matches the information stored within the document, it would imply that the document accessed represents the same version. On the other hand, if the location of the document were different from the information stored within the document, it would imply that the document represents a new version that is based on the document whose location information is stored in the document. The information about the new version would then have to be registered and the information within the document should then be overwritten with the current information. Therefore, the client-side drawing needs to store information regarding the location of the drawing, activate code on opening the drawing to compare data stored on the drawing to the current location of the drawing and send a request to the server either to register a new version or to update information regarding the existing version based on the decision it makes as to whether the file represents a floating copy or not.

The information stored within each drawing can be divided into two categories - component-specific and version-specific. For each project component that needs to be designed, there is some
basic information such as the project details and the component details that do not change from one version to another. This set of information is component-specific, and would allow the drawing to identify what project and what component of the project that the drawing represents. On the other hand, the variables such as the location of the drawing have different values from one version to another and are hence version-specific.

When a designer starts working on the design of a component, he needs to have a drawing that already contains the knowledge, in the form of document macros, about how the drawing should react to the environment when it is accessed. Such a drawing file is termed as a template drawing for the project component. Since the contractor provides the component-specific information at the onset of the project, it would make sense to provide this information also in the template drawings. These template drawings should then be made available to the designers in order to start designing the corresponding component.

A new version can be created by two ways – when a designer obtains a blank template drawing from the Document Control system, and when a designer creates a copy of an existing version. In either situation, the new version already contains the data block describing the project component and the document macros. The document macros would then verify the stored information about the location of the drawing with the environment and determine whether the drawing version represents a floating copy or an existing version. Based on that decision, the document macro could then request the Document Control system to either register the version or update information (such as date last accessed) about the version.
3.3.2: Automating Client-side Document Control Interaction

In order to minimize the need for designer's interaction with the Document Control system, it is necessary to have a mechanism by which a drawing can automatically execute commands to interact with the Document Control server as and when required. As mentioned earlier, document macros allow for storing commands within the document. In computer programming terminology, events refer to changes in the environment that are detected by programs. For instance, it is possible to have an event triggered when a document is opened. When an event is triggered, it is possible to instruct the computer to run a series of commands through an event-handler.

Instead of having the designer interact directly with the Document Control server, it is possible to trigger events when specified changes in the environment take place and use an event-handler to interact with the Document Control server on behalf of the designer. By this process, it is possible to provide automation for the designer's interaction with a Document Control system.

3.3.3: Communication and Collaboration

In addition to providing support for floating copies and client-side automation to Document Control, this research also aims to provide enhanced communication and collaboration mechanisms for designers working on the design of the same project component. In many large projects, a lot of time is lost due to lapses in communication between designers. For instance, when a designer realizes that there is an error in the design (such as a constraint violation), he would like to inform those designers who have already obtained a copy of the design from him. Also, if this error was not his fault, but already existed
in the drawing when he obtained it, he would like to inform the designer from whom he obtained his copy. Keeping track of this information manually can be tedious and some designers might be omitted by mistake in the process. By providing a mechanism to enable communication based on versions rather than the identity of the intended receiver, it will be possible to enhance communication between designers.

Since the document macros allow the creation of event-handlers, it is possible to make use of this facility to initiate communication channels between the designers working on the same project component, when they start working on the drawings. Using this, it is possible for designers to communicate with one another as they work, discussing approaches and alternatives. Further, it is possible to provide the capability for designers to collaborate in their efforts by allowing designers to exchange instructions for design modification between one another.

3.3.4: Intellectual Property Control

As mentioned earlier, the most common mechanism for Intellectual Property control for data files is using passwords and access restrictions at the server level. The problem with this mechanism is that there is no control over the access of the documents once they have been obtained from the server. It is desired to provide control of document access every time a user tries to access a document. In this respect, there is a second possible approach to providing Intellectual Property control for data files, using document macros. This approach inserts "intelligence" into a data file so that the file can decide based on the creator's specifications whether the user accessing it has the necessary permissions to do so.
In order to support the concept of the design drawing as an Intellectual Property, it is necessary to create a framework that can provide guidelines as to the necessary functionality required on the part of the data files. In our approach, the design drawing is no longer considered as a static representation of design information but as a combination of design data and knowledge. Since we desire to move the control of access from the server to the drawings themselves, the knowledge should not be separated from the data. The knowledge stores information about access rights for the documents. Further, this decision should be made before the data is displayed. If the knowledge decides that the user accessing the drawing has sufficient rights to access it, the data is displayed. On the other hand, if the knowledge determines that the user does not have authority to view or modify the drawing, the knowledge must force the file to close.

Since document macros allow the use of events (see Section 3.3.2), it is possible to decide whether the user has the permission to access a file at the point of access. There are two possible mechanisms to achieve this decision making process. One is to use the standard password protection mechanism. While this may be a good idea when a single individual tries to maintain the privacy of his data, controlling the flow of knowledge about the password within an organization is often neither straightforward nor reliable. The other mechanism is to provide an IP address-based blocking system. This mechanism is similar to those implemented by web-servers in order to protect the computers from unauthorized access. In this mechanism, the information about the list and range of IP addresses (or domains) that can access the computer is stored either within the data file itself or on the server. When the user accesses the data file, the document macro verifies that the user is working from a computer
that has been permitted to access it. If the information about the domains with the access rights were stored on the server, it would require that the user be able to connect to the Document Control server whenever he wants to access the data file. On the other hand, if the information were stored within the data file, there is a danger that it might be tampered with. In addition, when the information is stored within the data file, it will be difficult to update the information. Therefore, it might be better to store the information regarding the access restrictions on the server.

3.4: REVIEW OF APPROACH

Based on the proposed approach, the life cycle of a project might proceed along the following lines:

When a project is started, the contractor splits up the design into components. Having decided on the decomposition of the project in terms of the components, the contractor inputs this information into the Document Control server. As a result, template drawings are created and stored on the Document Control server. The template drawings contain the details of the object/component it represents and the Document Control knowledge in the form of document macros.

Researchers in CM and VC [16, 25] suggest that some of the important information that a version has to provide include details of the object it represents, a unique version ID, the owner, the version state, its version derivation history, and the version ID that should be assigned to its next child. These are the information stored within each drawing. As designers obtain copies of the templates and/or other versions and modify the design, the version information (version ID, owner, version state, etc.) is created, retrieved, verified and updated.
When a drawing file is opened, the first step is to verify whether the designer has sufficient rights to access the drawing. Here, the macro provides Intellectual Property control. The macro decides, based on the creator’s specification, if the designer is allowed to access the drawing and if so, grants the designer access to the drawing.

Every time a drawing file is opened, the macro determines whether the drawing is a floating copy or not. Based on the decision, information is sent to the Document Control server to either register a new version or to update information about the version. In order to update the version information, the server requires information regarding the project and component it represents, the version number and information regarding the user accessing the drawing. Similarly, in order to create a new version, the server requires the same data. However, the server in this case would have to return the new version number for this drawing based on the version number sent (which would be the parent version number used to decide the new version number).

In addition, a communication channel is created every time a drawing file is opened, in order to allow the designer to communicate and collaborate with other designers working on the same project component. Functionality can be provided to store preferences for communication, in a manner similar to other Chat mechanisms such as Yahoo! Chat and MSN Messenger.

When a designer creates a significantly better design of a component, he sends it to the contractor. The contractor then verifies that the design is both an improvement over the previous design and is also compatible with the designs of the other components of the
project. Once this is verified, the contractor makes this version available to the other
designers and instructs the server to notify the designers about the update.

When the designer opens any version of the design, the macro retrieves information
regarding the latest released version and if it is new, the macro notifies the designer that an
update is available. The designer can then obtain the latest released version and work with it.
In this manner, the design cycle continues until the contractor is satisfied with the final design.
CHAPTER 4: IMPLEMENTATION

In order to demonstrate the advantages of the proposed approach, a sample implementation was constructed that highlights the benefits of this approach. This chapter provides details of the implementation.

4.1: IMPLEMENTING CLIENT-SIDE AUTOMATION OF VERSION CONTROL

In order to highlight the benefits of the proposed approach to version control, it was necessary to decide the platform on which the implementation would be built. On the client side, it was necessary to have a CAD package that the designers could use to create and modify their designs. The CAD package must provide the ability to:

- Embed information and macros, and
- Allow the drawings to be able to trigger program code when it is accessed.

Many of the present-day CAD packages come with the ability to create and use macros with a range of macro languages. For example, AutoCAD comes with AutoLISP and VBA; Microstation/J comes with Java; while VectorWorks comes with VectorScript. For this research, AutoCAD and VBA were chosen. This decision was based on the wide popularity of AutoCAD and the powerful programming capabilities of VBA. In addition, AutoCAD allows drawings to embed information in the form of attributes of blocks, and since Release 2000, it allows drawings to contain embedded VBA macros¹. VBA also offers the ability to code event-driven procedures through which action can be taken when a particular change in the environment is identified by the macro.
AutoCAD provides support for two types of macros: embedded macros and global project macros. In the first case, the execution code (i.e., the agent) is embedded within the drawing along with the data about the environment variables. In the second case, the execution code is stored in a project file that has to be opened by the designer when he starts AutoCAD. Since it was desired to have the knowledge integrated within the drawing in order to allow the drawing to provide Intellectual Property for itself, embedded macros were used.

In order to embed information within a drawing, AutoCAD provides the concept of attributes within a block. A block is a named collection of objects such as lines, circles and other blocks that one can associate together to form a single object, or block definition. Using blocks helps users to streamline their designs. In order to allow users to store information about the group of objects that define a block, AutoCAD allows designers to provide additional information within a block by storing parameters such as part number, cost, comments or owner’s name and their values. These parameters are called attributes and can be set to be invisible, constant, pre-set or any combination of the three. For this research, the information regarding the component and the version can be stored in the form of attributes within a uniquely labeled block created specifically to store this information.

Thus, on the client-side, each design drawing contains macros and data about the component and the version embedded within it. The embedded data include both component-specific and version-dependant information, and is stored in the form of invisible attributes

---

1 Though AutoCAD R14 started allowing using VBA macros, it was only in AutoCAD 2000 that embedded macros within drawings was first achieved.
within a block. When the template drawings are created, it already contains the macros, the block and the component-specific information.

In order to keep track of the drawings and identify floating copies, it is necessary to establish a set of attributes that define the uniqueness of a version. As mentioned earlier, a version may be defined uniquely either by its location, or by a combination of its location and its timestamp. Since good designers make it a habit to take a backup of their design drawing before making any major changes they might want to revert, most other Document Control systems use the former approach. For the purpose of this research, only the location of a drawing is considered in defining the uniqueness of a version. In this research, the location of a drawing file is considered to be uniquely defined by the complete path of the file and the machine on which it resides.

Based on previous research in Document Control and additional requirements for this research, the information that a drawing has to store include: server location, project ID, project name, component ID, component name, version of the component, user information, path and machine.

The Project ID and the Component ID uniquely identify the projects and the components within that project respectively. These are generated by the server using a numeric-sequencing algorithm and are used to establish a name space for the versions in a project. The server location, project ID, project name, the component ID and the component name represent project-specific data and are set when the templates for the components are being created.
The version number of a component is an identifier that helps determine the position of the version in the version-derivation graph and is generated each time a new version is registered. The template version is always specified as 1.0. Its children can be considered as sibling versions, and are provided with identifiers 1.1, 1.2, 1.3, etc successively as they are created. The children of 1.1 then are identified as 1.1.1, 1.1.2, and so on. By using such an identifier scheme, it is easy to track the evolution of any version, given the identifier. In addition, it is easy to identify the descendant versions of any specific version.

The other three attributes (file location, user information and machine) represent the environmental variables that help determine the uniqueness of a version. The path of the file represents the location of the drawing within a computer. The user information can be considered to involve the username-domain combination with which the designer logged in to the computer. The username is the designer's login name. The domain represents information regarding the designer who has logged in to the system. The machine is identified by the machine ID or name assigned to it in the network of computers that it exists in. The domain and the machine ID are part of the system information that can be obtained from the registry file of the computer.

Once a designer obtains a drawing file, he can open it, modify it and transfer it. When a designer opens a drawing file, the code embedded within the drawing first verifies if the server has the necessary information about the user. If the user information is not available in the server, the code obtains the user's personal data such as email address, full name & company, and sends the information to the server. Then, it compares the location of the drawing file with the values of the corresponding variables (i.e., the path of the file and the machine ID)
stored in the drawing. If they are the same, it notifies the server to update information (date last accessed and the person accessing it) about the version. Otherwise, it requests the server to register this drawing as a new version specifying the version from which it is derived. In response, the server determines the version identifier based on the version identified of the specified (parent) version and the count of already created children. This identifier is sent back to the drawing and the drawing updates the version identifier information in the embedded block. This mechanism allows the Document Control server to keep track of the floating versions.

4.2: VERSION CONTROL SERVER

In order to highlight the benefits of the new approach to automate client-side interaction for Version Control, it was necessary to work with a Version Control server. There are many commercial Version Control servers available in the market. Since most vendors of Version Control servers do not allow buyers to modify their code, we decided to build our own Version Control server. The server we built was a stripped-down version of commercially available servers and only the basic functionalities needed to highlight the advantages of the proposed approach were supported.

To handle server-side requests and to store information about the various projects and their components, PHP and PostgreSQL were chosen. A PHP program runs on the server to handle requests from the clients and the contractors. A web interface was provided for the users to obtain information and drawings (latest updated versions of components of projects) from the server. The server also allows the contractor to create the template drawings for each component of a project. Since the objective of this research is to provide automation for the
client-side interaction of Version Control, functionality was provided to allow for the
drawings to automatically request and receive information. This allows the drawings to update
information about itself or to request the server to register itself as a new version.

In order to create the template drawings for the components of every project, a VB
program is created that can be called to by PHP. In addition, the supervisor has to provide a
drawing file that already has the macros embedded within. The PHP code passes information
regarding the project and the components to the VB program. The VB program opens the
drawing file and creates a block to store the information regarding the component and the
version. Then, the program uses the information sent to it by the PHP program to insert the
component-specific information into the template drawings.

Welcome to Server For Embedded Version Control

List of Existing Projects

- Project 1-Story Building  Edit & Update  View Details and Download Templates

New Project

Fig. 4.1 Server Menu Page displaying list of existing Projects
Using PostgreSQL, databases are created to store information about the projects and their components. A base database is created to store information about the list of projects, and the name of the databases used for each of them. For each project, a database is created to store information about the components, and the designers accessing them. Also, a table is created within the project database for each component of the project to store information regarding the versions. All these information are created and modified based on the user requests by PHP through SQL queries.

At the onset of each project, the contractor uses the web interface and instructs the Document Control server to create a new project and the various components involved. When the server receives such a request, it creates a unique ID for the project. Using the VB program on the server side, a project template drawing is created. Then, it adds the information about the project and the template on a database common to all the projects. Using this unique ID, the server creates a new database for the project. Similarly, for each of the component specified, it generates a new unique ID within the project. After that, it creates new tables for each of the components that the contractor specifies and adds the new information into it.
Create New Project and Add Components

Project Name: ________________________________

Name for AutoCAD Drawing storing Project Template: ________________________________

Component Name | AutoCAD Daw File storing Component Template

| | |
| | |
| | |

Submit | Submit All Add More Components | Reset

Fig. 4.2 Form to create a new project and specify its components

Once the information regarding a new project has been processed, drawing templates are created for each of the components. Each drawing template contains the embedded macros and the embedded project-specific data such as the server location, the project ID and name, and the component ID and name. This is now available for the designers.
Project Description: Project 1-Story Building

Supervisor: sakumar@rice.edu

List of Existing Components:

Architecture - First Floor Download Template | Download Current Version

Electrical - First Floor Download Template | Download Current Version

Structural - First Floor Download Template | Download Current Version

Mechanical - First Floor Download Template | Download Current Version

Back to Main Page

Fig. 4.3 Server displays information about a project and providing links to download template drawings and current versions of its components

4.3: IMPLEMENTING ENHANCED COMMUNICATION & COLLABORATION

Having discussed how the version server has been implemented, let us now discuss how communication and collaboration can be enhanced by this approach.

In order to facilitate communication between designers working on the same project/component, features are provided to allow the designers to interact, share ideas, and inform others about possible issues that he might have observed in the design. In large
construction projects, a lot of time is often lost due to lapses in communication between
designers or divisions. When the designer knows whom he has to communicate to, there are
several mechanisms available. The problem occurs when the designer is not sure whom he has
to contact. For instance, if a designer needs to notify all those who obtained a design drawing
from him, he has no way to be sure that the message gets delivered to all of the intended
receivers. To address this issue, it was decided to provide version-based communication
instead of person-based communication.

Two mechanisms of communication were provided to support both synchronous and
asynchronous communication. Synchronous communication refers to a two-way
communication, such as talking over a phone. This was provided through a chat mechanism.
Further, collaboration and CAD integration was provided by allowing designers to send
AutoCAD commands through the chat mechanism. Asynchronous communication refers to
one-way communication like sending e-mail or fax. This was provided by allowing designers to
post and view messages.

The embedded macros create a menu (Fig. 4.4) when the drawing is opened to allow the
designer to access the features provided for enhancing interaction between the various
designers working on the same project/component. This includes allowing the user to chat
and to post messages to a shared forum. Functionality is also provided to enable collaboration
between designers by sending commands in the form of Instant Messages that can then be
executed.
In order to support synchronous communication, functionality was provided to allow designers to chat as they work on the design. When the user opens a drawing, he is automatically connected to chat with the other designers who are working on the design of the same component. This allows the designers to discuss problems, strategies and solutions. In order to create a chat system, various options were considered. Among them were to use PHP Chat programs, to use Yahoo! Chat, or to create a simple custom chat system. Each approach had its advantages and drawbacks. For the sake of a sample implementation, it was desired to use an approach that would be simple and not time-consuming to build or adapt. Further, the chat mechanism for the project required certain features such as support for multiple chat rooms (one for each project component), which were permanent in nature. The problem with PHP Chat programs available on the Internet was that they were either too complex to modify
for the research, did not have the ability to host multiple chat rooms, or involved complicated procedures for administration. Yahoo! Chat provided a possible solution since many programmers have provided source code in various languages including Visual Basic and C/C++ to create personal programs using Yahoo! Chat. The biggest drawback of using Yahoo! Chat was that the chat rooms created here were transient in nature and would be destroyed as soon as everybody left the room. Keeping track of the rooms that are available and those that need to be created turned out to be inefficient and unreliable. Finally, none of the available programs provided information on how to create personal conference rooms. The best option turned out to be to create a personal chat system, custom-made to suit the needs of the research.

An important advantage of embedding the chat mechanism into the drawing is that it makes it possible to transmit commands to modify the document through the chat. This could be achieved by allowing the user to specify AutoCAD commands and the corresponding parameters and using the chat mechanism to send it so that others can use the same command. As a result, functionality was provided to allow the designers to send commands to the chat room he is in. In order to distinguish between a chat message and an AutoCAD command, a unique prefix can be attached to the command. When VBA receives the chat messages, it can parse through the message to identify if the message is an AutoCAD command or not. When VBA recognizes an AutoCAD command through the chat interface, it asks the designer receiving the command if he would like to execute the command. If the designer agrees, the command is executed.
Figs. 4.5 and 4.6 show how the designer can send AutoCAD commands to other designers. The first form shows the request made by the macro to the designer to enter the command that he wants to send to the others. The second form shows the request made by the macro that has received the command (here, “line 0,0 1,1”) to the designer for permission to execute the command. The form specifies who sent the command and what the command is. If the designer agrees, the command is executed.

Fig. 4.5 Text box to send commands to execute through chat interface

Fig. 4.6 Request to execute command received through chat interface

In order to create a chat system, a table is added in each project database to store the recently posted messages. In most modern IM systems, messages are transmitted using the Peer-to-peer networking philosophy. This would then not require storing messages on the server. However, such a mechanism proved to be beyond the scope of this research due to
time constraints. Therefore, it was decided to use a triggering mechanism that could call a function at regular intervals and this function would receive updated chat information from the chat server and display the new messages, if any.

To support asynchronous communication, the user is allowed to post messages to other designers. The information about the receiver is specified in terms of the version of the drawing rather than the identity of the intended receiver(s). While there are many open-source forum software available, none of them support a version-based addressing mechanism. As a result, we decided to build a simple custom forum that highlights the benefits of this approach.

Two preset list of intended receivers are provided by the document macros to allow the designer to notify the designers who obtained the design drawing from the sender and to notify the designer from whom the sender obtained his drawing. In addition, the designer was also provided with the functionality to define his own list of receivers (Fig. 4.7). This is accomplished by allowing him to use wildcards to specify one or more recipients. Wildcard characters allow the user to match a group of characters. In common computing practice, the wildcard ‘*’ is used to match a sequence of characters (of any length) while the wildcard ‘?’ is used to match a single character. For the purpose of this research, the wildcard characters chosen were ‘%’ (in place of ‘*’) and ‘_’ (in place of ‘?’). Therefore, specifying the receivers list as ‘%’ would match every version. Similarly, specifying ‘1.2.%’ would match all the children versions of version ‘1.2’. One could also use ‘_’ and specify ‘1._’ which would match versions ‘1.0’ to ‘1.9’. In order to post a message using the wildcards (Fig. 4.7), the macro displays the version number that the user is working on and prompts him to specify the search criterion for the version number. Then, he can post any message that he wants. When a designer opens his
drawing, the macros embedded within the drawing checks to see if the user has any new messages to read and displays this information to the designer. Then, the designer can browse through the messages that are relevant to the design he is working on.

![AutoCAD dialog box](image)

Fig. 4.7 Posting message in message board – Specifying receiver list

In order to create this asynchronous communication mechanism, a table is created in each project database in the server to store messages along with information about the sender, the time it was sent and a search pattern to match for the intended receivers. When a designer posts a message, this information is added to the table. Within each drawing, a parameter is used to store the information about when the messages were last checked for that version. When the drawing is opened, it sends this information in order to retrieve data about whether there are any new messages for which his version matches with the search pattern specified. If so, the designer is notified so that he may browse through the messages.
CHAPTER 5: CONCLUSION

This chapter describes the contribution of the research, the issues that remain as a significant barrier to the wide adoption of this approach, and directions for future work.

5.1: CONTRIBUTION OF THIS RESEARCH

There are four major contributions provided by this research:

- It addresses the issue of identifying floating copies in Document Control systems. This is important because it is very common for designers to transfer files amongst one another without notifying the Document Control server. This functionality allows Document Control systems to track the floating copies and the designers using them.

- It provides client-side automation of basic version control functionality by tracking versions and interacting with the Document Control server on behalf of the designer. This allows the designers to work on the design without having to spend time on communicating with the Document Control server. At the same time, it ensures that the data in the Document Control server is complete and up-to-date.

- It provides enhanced communication by supporting synchronous communication through chat mechanism and asynchronous communication through message boards. These mechanisms provide additional features that further enable collaboration between designers by allowing the designers to send AutoCAD commands through the chat mechanism.

- The research provides a framework to support the concept of design drawings as Intellectual Property.
5.2: ISSUES & LIMITATIONS

Having discussed the contribution of this research, it is necessary to identify the issues and limitations involved. This can been subcategorized into three components: the security issues involved when using macros, issues unresolved with the proposed approach, and the limitations of the implementation provided.

5.2.1: Security issues when using macros

The significant benefits of using embedded macros can be fully realized only if the security issues surrounding their usage are addressed. One of the impediments to the wide use of embedded macros is the security of running foreign macros on your computer. In the recent past, VBA has provided hackers a powerful platform for creating many viruses. A virus is a piece of programming code (usually disguised as something else) that causes some unexpected and usually undesirable event. A virus is often designed so that it is automatically spread to other computer users. Viruses can be transmitted as attachments to an email note, as downloads, or be present on a diskette or CD. Broadly speaking, viruses can be classified as File infectors, Boot-sector viruses, and Macro viruses. Of the three, the latter is of concern here. Macro viruses are currently the most popular form because of the ease with which they can spread and the damage that can be inflicted. While macros need not necessarily be a virus, most programs that provide macro capabilities, often identify that all macros might be potential viruses. It is essential for users to ensure that only certified macros from reliable sources are loaded/used. Unlike Java, VB and VBA do not have a mechanism to prevent malicious code from harming your computer. It is desirable to provide a mechanism in VB and VBA by which users can specify safety restrictions on the macros that they run. By doing this, the users can prevent macros from causing damage, both intentionally and accidentally. Visual
Basic is yet to implement a strategy similar to Java's ability to prevent any code from modifying the system on which it runs. Since we do not control the AutoCAD VBA interpreter, the security of VBA scripts cannot be assured.

If VBA implements a security model similar to Java, it will not have any effect on this approach and implementation since our macros only read information from the system, but does not modify the system variables. In order to identify the system, the program has to obtain details such as the machine name and the domain of the network to which he has logged in. This information is available in the Windows registry. Therefore, it is necessary for the VBA code to access the registry. Even if a mechanism were implemented to prevent tampering of system information, the macros here would not be affected as they only read the system information and do not attempt to change it.

A notion of macro security can also be enhanced if AutoCAD gives details about the macros when opening a drawing (created by, last accessed, size, etc.). Although it can be set, and is usually set by default, to inform when opening a drawing containing macros, it does not specify any detail about the macros that are embedded. This can be a little disconcerting for the common user since macros are presently the most common tools for virus coding. For example, Microsoft Office, which also uses VBA macros, has a feature called Certificate, which allows the users to specify the names of the creators of the macros that are acceptable to be loaded. Such a feature is presently not available in AutoCAD VBA.

Another issue when considering the security of using VBA macros in AutoCAD is that only a single level of security for the macros is provided for reading and writing the macros.
When a macro is created, the creator has the functionality to provide a password to prevent others from tampering with it. However, this password is required to both read as well as to edit the macros. This single level of security provides the programmer with a dilemma. If he does not protect the macros with a password, it is possible for someone to modify them (possibly into a macro virus) and then pass the drawing to others. If he does protect the macros with a password, the users cannot even peruse the macros to obtain a certain level of confidence that the macros are safe. If a two-level security were provided in AutoCAD so that the contractors can specify two different passwords for reading and for writing, this problem would be alleviated.

5.2.2: Issues with the approach

An important limitation with our approach is the mechanism to uniquely identify a version. In our approach, a version is defined by the name of the computer in which it resides, and the location of its file within the computer. For instance, it is possible for two computers to have the same computer name (especially when they are working on stand-alone computers, users tend to leave the name of the computer to its default value). If a file is copied from one of these computers to the other, and they are stored in an identical file path, the drawing will not be recognized as a new version. In the other extreme, it is also possible to have the same version identified as two different versions. Let us consider the situation when drawings are stored on removable storage media, such as floppy diskettes or Iomega Zip Diskettes, or over a local network directory. When such a drawing file is opened in different computers, they will be recognized as different versions because the program does not identify that the version resides outside the computer. In most cases, however, the parameters chosen (name of computer and path of file) identify versions uniquely.
A related issue regarding the present approach is that it does not retain the design implementation of a version from which a copy was obtained. When both the parent version and the children versions have changed, it is not possible to use this approach to obtain information about either what the changes were or what the original design was. While answering this issue is very straightforward and might improve the productivity and efficiency of the design team, it would involve storing the same file in multiple snapshots on the server, thereby occupying a lot of storage space. Many Document Control systems also do not use a timestamp to identify versions uniquely, but instead expect the designer to create a backup to his version if he intends to test some modifications and plans to revert to the old design if the tests fail. After careful consideration, this research also decided to adopt this philosophy.

Another notable limitation is the difficulty in introducing changes to the executable code embedded within the drawings after they have been distributed. If there are any updates to the code, the update of the code within drawings that have already been checked-out from the server is not straightforward. The significance of this problem depends on the circumstances under which such an update is desired. If there is an important error that needs to be corrected, or a major improvement that can be achieved, it is very desirable to keep the code updated. In such circumstances, this limitation would cause considerable trouble. The best way to tackle such a situation is to provide a flag-based communication mechanism so that the user can know that a macro update is available. He can then obtain the new template drawing, in which functionality can be provided to copy the design details from the version that had been opened. In this way, a new drawing is created with the design details of the version, and the updated code from the new template drawing.
5.2.3: Issues with the implementation

AutoCAD does not support an implementation of our approach to the concept of design drawings as Intellectual Property. In our approach, we propose that the drawing should store knowledge about the access rights of various individuals within the drawing. Then, when the drawing is opened, this knowledge would verify if the user has sufficient authority to access the drawing. If the knowledge decides that the user does not have authority to access the drawing, the macros would close the drawing. However, in order to implement this approach, AutoCAD must provide the functionality to allow creators of macros to specify that a particular macro would have to be executed if someone wants to open the drawing. As of now, this is not provided. AutoCAD allows users to disable all macros, making this approach currently infeasible.

One of the most serious limitations at the present is that AutoCAD has not yet integrated VBA completely. For instance, even though AutoCAD triggers events when the user draws a new line, no information can be obtained as to the parameters (to and from points) specified by the user. If this were possible, it would be easier for designers to collaborate. They can specify that they wish to collaborate with a certain group of people, and then the macro would track their commands and send it to the others. The macros running on each of the group member's drawings would then obtain this information and perform the same operation.

In addition, some of the features of Visual Basic are not available when macros are embedded within a drawing, because the macro is considered as a class module. So, some controls and functions that are specific to forms and modules are not available to these macros.
For instance, the callback functionality requires the programmer to use a module in which the procedure called upon is written. Therefore, it is not possible to use this in embedded macros.

One of the most important restrictions imposed by this implementation is that it is completely based on the Windows environment. Even the server has to have access to a VB program that requires a Windows environment to operate. Further, a basic assumption is made that some of the DLLs used in the client-side, such as the WinInet DLL and the Internet Explorer object DLL, are available on the system. While this is a valid assumption, it is possible that someone might delete the DLLs or remove Internet Explorer from their system.

5.3: AUTOCAD WISHLIST

A wish-list is provided to indicate the new features that, if added in AutoCAD, would be useful in the context of this research.

- **Better Integration between AutoCAD and VBA:** AutoCAD is yet to provide complete integration of VBA into itself. For instance, consider the situation where VBA causes events to be triggered when AutoCAD executes a command. It would be useful if the event handler would be able to obtain the set of arguments that are passed along with the command name. This would allow the programmer to obtain details of commands as they are being executed, allowing the programmer a mechanism to create heuristics that can decide when a version has undergone major modifications. In addition, it would now be possible to store versions as a set of commands that need to be executed on the parent version.
• **Allow essential macros**: Allow the creator of macro to be able to specify that that macro would have to be executed if a user wants to access the drawing. Using this feature, it would be possible to provide Intellectual Property control as envisioned in this research.

• **Details of Macros**: Provide the user with information about the macro he would be running if he opens the drawing. This would be much more useful than just warning the user that the drawing contains macros which might be viruses.

• **Signed Macros**: Allow macros to have certificates of authenticity provided similar to what Microsoft Office provides. This would allow the users to know that macros made by a particular author represent those that are part of the Document Control system.

• **Two level Password Protection for Macros**: Allow the creator to specify two separate passwords – one to allow users to view the macros and another to allow users to modify them. This would allow the creator of the macros to ensure that the macros are not tampered and the users to peruse the macros.

### 5.4: SCOPE FOR IMPROVEMENT

Because of the limited integration of VBA into AutoCAD, some interesting features that were proposed could not be implemented.

One of the possible features that were considered was the ability for any user to compare two drawings for differences between them. Most of the work in any drawing is built on the work in the parent drawing. Therefore, it might be useful to compare two drawings to see how they diverge in design. Further, it might be possible to identify heuristics that “intelligently”
determine when one drawing is significantly different from the other. This approach can be used as an alternative method of identifying new versions.

Another feature that was considered (and is already a part of AutoCAD 2000i to an extent) is the functionality to allow multiple users to simultaneously work on the same copy of a file. AutoCAD 2000i allows one user to modify the drawing while others can view snapshots of the drawing. Instead of just showing a snapshot, sometimes it might be useful if each of the designers could modify the drawing by sending instructions through the Internet. However, though AutoCAD triggers events in VBA whenever a command is called, it does not pass the parameters that are used to complete the command. Without knowing what was the exact command, it is not possible to transmit the information to the other designers. While this has been overcome by allowing users to send commands while chatting, it would be possible to provide a cleaner and more direct solution if AutoCAD provides the details of the parameters passed along with the command.

Some useful features of Version Control are not yet supported on the client-side. Further, most of the concepts of Configuration Management such as equivalences, and dynamic configuration binding have not been implemented. While dynamic configuration binding might not have been a very useful concept in this research, the use of equivalences to easily allow designers to obtain equivalences for various components of the project could have been a useful tool. Support for any of these features can be provided on the client-side based on the Document Control system chosen.
An interesting feature that can be provided is the ability to support peer-to-peer communication. Presently, the chat system goes through the server. It is possible to store the information about the IP address of a designer when he accesses a drawing and use this to provide functionality for peer-to-peer communication between two or more designers. In this manner, a designer can enter communication between just those designers that he decides, and collaborate with those designers that he trusts. This can improve the speed of communication and provide better control for collaboration.

When advanced features are provided to the user, it might be desirable to store the user’s preferences. For instance, a designer might want to provide preferences to restrict the group of people with whom he communicates and collaborates. This can be provided by storing the preferences either within the drawing, or on the computer in which he works. This allows the designer to use either specific preferences for a drawing, or a single preference set for all the drawings he works on in that computer.

5.5: CONCLUSION

The research has provided functionality to identify and track floating copies while providing client-side automation for version control. Where a commercial Document Control system is available, the client-side code can be modified with relative ease to provide additional functionality to support the advanced features available in that system. Further, the research has provided functionality to enable better communication and collaboration between designers working on the various project components.

In addition, the research has also provided a framework for supporting Intellectual Property. It has specified the features in a CAD package that would be required to provide
such functionality and has discussed the necessary mechanism that can implement the concept of Intellectual Property for design drawings.
BIBLIOGRAPHY


18. Wojciech Cellary, Geneviève Jomier, “Apparent Versioning and Concurrency Control in Object-Oriented Databases”,


29. Sascha Schumann, Harish Rawat, Jesus M. Castagnetto, Deepak T. Veliaht (1999), "Professional PHP Programming", Wrox Press Ltd., USA

30. http://www.devx.com for programming instructions on how to post form data without using a browser in VB.


APPENDIX A: SAMPLE CLIENT-SIDE CODE (VBA)

The following sample code is a part of the client-side code that is called when a drawing with the embedded macros is opened. It obtains the stored information about the various parameters, compares them to the values of the corresponding parameters in the environment in which it is opened and based on the results, decides what to request from the server. Also, there is a Timer based chat mechanism created and the menus are generated.

Public Sub Initialise()
    Dim Attribs As Collection
    Dim objEnt As AcadEntity
    Dim objAttrib As AcadAttribute
    Dim Data As String, Tag As String, Result As Boolean

    pos(0) = 0: pos(1) = 0: pos(2) = 0

    On Error Resume Next
    If ThisDrawing.Application.Visible = False Then Exit Sub
    Set dataBlock = ThisDrawing.Blocks.Item("VerCtrlData")
    If (dataBlock Is Nothing) Then
        Set dataBlock = ThisDrawing.Blocks.Add(pos, "VerCtrlData")
        SetServerInfo
        Set dataBlock = ThisDrawing.Blocks.Item("VerCtrlData")
    Else
        Set Attribs = GetAttributes(dataBlock)
        For Each objAttrib In Attribs
            Tag = objAttrib.TagString
            If Tag = "Server" Then
                Set Server = objAttrib
            ElseIf Tag = "PrjID" Then
                Set PrjID = objAttrib
            ElseIf Tag = "PrjName" Then
                PrjName = objAttrib.TextString
            ElseIf Tag = "DwgID" Then
                Set DwgID = objAttrib
            ElseIf Tag = "DwgName" Then
                DwgName = objAttrib.TextString
            ElseIf Tag = "User" Then
                User = objAttrib.TextString
            ElseIf Tag = "Path" Then
                StoredPath = objAttrib.TextString
            ElseIf Tag = "Version" Then
                Ver = objAttrib.TextString
            ElseIf Tag = "Machine" Then
                Machine = objAttrib.TextString
    End If

    ' Rest of the code...

ElseIf Tag = "Domain" Then
    Domain = objAttrib.TextString
ElseIf Tag = "DateMsgsChecked" Then
    DateMsgsChecked = objAttrib.TextString
End If

Next
If (Server Is Nothing) Then
    SetServerInfo
    Exit Sub
End If

If PrjID Is Nothing Then
    Method = "AddPrj"
    If (PrjName = "") Then
        PrjName = InputBox("Enter Project Title")
        SetAttribute "PrjName", PrjName
    End If
    User = ThisDrawing.GetVariable("LoginName")
    SetAttribute "Contractor", User
    Query = Server.TextString & "?Method=" & Method & 
    "&PrjName=" & ConvertString(PrjName) & "&User=" & ConvertString(User) & 
    "&Domain=" & ConvertString(Domain)
ElseIf DwgID Is Nothing Then
    Method = "AddDwg"
    DwgName = InputBox("Description of Project component being designed", "Drawing Title")
    SetAttribute "DwgName", DwgName
    PresentPath = ThisDrawing.FullName
    Query = Server.TextString & "?Method=" & Method & 
    "&PrjID=" & ConvertString(PrjID.TextString) & "&DwgName=" & 
    ConvertString(DwgName) & "&Path=" & ConvertString(PresentPath) & 
    "&Domain=" & ConvertString(Domain)
Else
    PresentPath = ThisDrawing.FullName
    Result = GetKeyValue(HKEY_LOCAL_MACHINE, 
    "System\CurrentControlSet\Control\ComputerName", "ComputerName", 
    ThisMachine)
    If Not (Result) Then Result = 
    GetKeyValue(HKEY_LOCAL_MACHINE, 
    "SYSTEM\CurrentControlSet\Control\ComputerName\ComputerName", 
    "ComputerName", 
    ThisMachine)
    ThisUser = ThisDrawing.GetVariable("LoginName")
    Result = GetKeyValue(HKEY_CURRENT_USER, "Volatile Environment", "USERDNSDOMAIN", ThisDomain)
    If ThisDomain = "[none]" Or ThisDomain = "" Then
        ThisDomain = ThisMachine

    If (Domain = "" Or User = "" Or ThisDomain <> Domain Or 
    User <> ThisUser) Then
        SetAttribute "Domain", ThisDomain
        Method = "VerifyUser"
        Query = Server.TextString & "?Method=" & Method & 
        "&UserID=" & ConvertString(ThisUser) & "&Domain=" & 
        ConvertString(ThisDomain)
ElseIf (ThisDomain <> Domain Or PresentPath <> StoredPath Or ThisMachine <> Machine) Then
    Method = "AddVer"
    Query = Server.TextString & "?Method=" & Method & "&PrjID=" & ConvertString(PrjID.TextString) & "&DwgID=" & ConvertString(DwgID.TextString) & "&User=" & ConvertString(User) & "&Parent=" & ConvertString(Ver) & "&Path=" & ConvertString(PresentPath) & "&Domain=" & ConvertString(ThisDomain) & "&Machine=" & ConvertString(ThisMachine) & "&DateMsgsChecked=" & ConvertString(DateMsgsChecked)
    SetAttribute "Machine", ThisMachine
    SetAttribute "Domain", ThisDomain
    SetAttribute "Path", PresentPath
Else
    Method = "UpdateVer"
    User = ThisDrawing.GetVariable("LoginName")
    Query = Server.TextString & "?Method=" & Method & "&PrjID=" & ConvertString(PrjID.TextString) & "&DwgID=" & ConvertString(DwgID.TextString) & "&Version=" & ConvertString(Ver) & "&User=" & ConvertString(User) & "&Domain=" & ConvertString(Domain) & "&DateMsgsChecked=" & ConvertString(DateMsgsChecked)
End If
    Set objTimer = New VbTimer.Controller
    RefreshRate = 90
    SplitAddr Server.TextString, SrvrLoc, ScriptLoc
    Query2Cht = "Method=ViewChatMessages&PrjID=" & ConvertString(PrjID.TextString) & "&DwgID=" & ConvertString(DwgID.TextString) & "&RefreshRate=" & Str(RefreshRate)
    objTimer.Interval = CSng(RefreshRate * 1000#)
    objTimer.Enabled = True
    MnuCreate False
End If

Set brwsr = CreateObject("InternetExplorer.Application")
brwsr.Visible = False
brwsr.Navigate2 Query, 12

End If

End Sub
APPENDIX B: NOTE TO ADMINISTRATORS

For the administrator, one of the first things needed to be done is to setup information about file locations, etc. In the PHP files, it is necessary to input information of the location of the PostgreSQL server, and the user-name/password combination needed to access it (including write access). Next, there is a configuration file to be located in the same directory as the PHP file in which the location of the Visual Basic Executable (which is required to create the Template Drawings) is to be specified. Also, for the Visual Basic Executable, there is another configuration file that specifies where the Basic Template Drawing (which must contain the macros and the empty Block to store information) is located. Once this information is setup, the administrator can access the web server and create new projects and components through a simple interface.

Also, it is necessary for the system administrators to install Visual Basic 98 Runtime files on each of the computers used by the designers in order to be able to use the macros. This is a free program that can be downloaded at the Microsoft's support web-site http://support.microsoft.com/support/vbasic/runtime.asp. This is necessary because the VBA macros in AutoCAD use Visual Basic-specific procedures that are included in these DLLs. It is also necessary to note that in the case of Windows NT/2000 (and probably also Windows XP), it will be necessary for the Administrator to install this. In Windows 95/98, anyone can do the installation.

Further, it is necessary to install a DLL on each of the computers used by the designers. This DLL is called vbdTimer and it provides the functionality of a timer (Timers are objects that can trigger events every so many micro-seconds). First, this DLL must be installed in (or
copied to) the system32 sub-directory of the Windows Folder. Then, it must be registered in the windows registry. To do this, the administrator can use one of the many freeware utilities that provide functionality to add information about DLLs and other ActiveX components into the windows registry. This DLL is necessary for providing Chat functionality.

The following sections provide some hints to the administrator who might have to face with rare circumstances where he would like to delete information about a project or a component after the project is over.

1. Removing all information about all projects

In order to clear all created information (except the PHP file), the contractor has to:

- Remove all sub-directories created under the Templates directory specified in the PHP configuration file.
- Open the template1 database and drop the common database and all the project databases

2. Removing all information about a particular project

In order to clear all information about a particular project, the contractor has to:

- Remove the sub-directory created under the Templates directory (specified in the PHP configuration file) that has the name corresponding to its Project ID.
- Open the template1 database and drop the project database named as the project ID
- Open the common database and remove the record in the project list table corresponding to this project
3. **Remove all information about a particular component**

In order to clear all information about a particular component, the contractor has to:

- Go to the sub-directory created under the Templates directory (specified in the PHP configuration file) that has the name corresponding to its Project ID. There, remove the sub-directory whose name corresponds to the component ID.

- Open the project database and drop the table named as the component ID.

- Still in the project database, remove the record in the component list table corresponding to this component.
APPENDIX C: SUPPORTING EXISTING DOCUMENT CONTROL SYSTEMS

This section provides information regarding the necessary modifications that need to be made when another Version Control system (or Document Control system) is used. Some modifications need to be made both on the client-side (provided by this research) and to the Document Control system being used. These are discussed below.

4. Modifying existing VC systems to support client-side automation

In order to support the client side automation as provided by this research, existing Version Control servers would have to be modified to accept certain function calls and also to store some information on the server that is not necessarily stored in traditional VC servers.

In order to be able to support chatting and posting messages on a message board, and identifying users, the server should be able to store user information (such as login id and domain, name and e-mail), chat information (sender, time sent, message and whether the message is an IM or a command to execute) and message board information (sender, time sent, message and intended receivers). Functionality should therefore also be provided to verify existence of user (and if not, add new user information), post and receive messages in chat mechanism and in message boards.

In addition to the above-specified functionalities, the servers should be able to extend the existing ability of creating new versions and updating version information to handle such a request over the Internet. Further, the server would need to create the template drawings and provide a mechanism by which they would be available to the designers for development.
5. Modifying client-side code to support existing VC servers

Based on the database structure of the Version Control server selected, the requests that are issued from the client side would have to be modified. Further, since the client-side code created for this research supported only the basic functionalities of version control, it would be necessary to add more code to provide support for advanced Document Control features such as frozen versions and equivalences.