INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6” x 9” black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700  800/521-0600
Integrated Building Analysis and Design System Using Distributed Object Computing

by

Zhe Wang

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

Master of Science

APPROVED, THESIS COMMITTEE:

Michael Terk, Chairman
Assistant Professor of Civil Engineering

Panos C. Dakoulas
Associate Professor of Civil Engineering

Satish Nagarajaiah
Associate Professor of Civil Engineering

Houston, Texas
May, 1999
ABSTRACT

Integrated Building Analysis and Design System

Using Distributed Object Computing

By

Zhe Wang

This study describes a new approach to building integrated design using distributed object technology. It identifies a number of issues involved in integrating tools initially developed for stand-alone use into an integrated design environment and shows how this approach can be used to address these issues. The benefits of this approach are illustrated by the description of IBADS, an integrated building analysis and design environment, implemented during this research effort. IBADS uses the proposed approach to integrate commercial off-the-shelf software packages AutoCAD, Etabs and SteelEr into an environment for design of 2-D frames.
Acknowledgments

I wish to express my gratitude to all those persons who made it possible for me to accomplish this thesis.

My special thanks to my advisor, Professor Michael Terk, for his invaluable guidance, encouragement, constructive ideas and suggestions. Without his help, I will never be able to complete this work.

My appreciation also goes to Professor Panos C. Dakoulas and Professor Satish Nagarajaiah, for their interest in my work and their suggestions for its improvement.

I also wish to thank my friends and colleagues, Kobe, George, Panos, Frank, Yaming and many others, for having provided such a wonderful work and academic environment. My years at Rice will be my most memorable because of them.

Finally, I want to express my deepest appreciation to my parents, my wife and the rest of my family for their understanding and unceasing support.
# Table of Contents

Abstract ......................................................................................................................... ii

Acknowledgement .......................................................................................................... iii

Table of Contents ............................................................................................................ iv

List of Figures ................................................................................................................ vii

1. Introduction ...................................................................................................................... 1

    1.1 Purpose ..................................................................................................................... 1

    1.2 Objective .................................................................................................................. 1

    1.3 Motivation ................................................................................................................ 1

    1.4 Organization .......................................................................................................... 4

2. Background .................................................................................................................... 6

    2.1 Integrated Design Software Environment ............................................................... 6

    2.2 Review of Related Integration Research ................................................................... 7

        2.2.1 Integrated Building Design Environment (IBDE) ........................................ 8

        2.2.2 Agent-based Framework for Integrated Facility Engineering ..................... 8

        2.2.3 Circle Integration ............................................................................................. 9

        2.2.4 Intelligent Computer-Aided Design System (ICADS) .............................. 10

        2.2.5 Knowledge-Aided DataBASE management system (KADBASEx) ............ 10

    2.3 Agent Management ................................................................................................ 11

    2.4 Agent Management and Distributed Object Computing ........................................ 14

        2.4.1 Distributed Object Computing ...................................................................... 14
3. Distributed Tool Integration Framework .............................................................. 19

3.1 Overview of the Approach ...................................................................................... 19

3.1.1 Process Management ....................................................................................... 19

3.1.2 Information Management ................................................................................. 21

3.1.3 Agent Management ........................................................................................... 24

3.2 Approach Description ......................................................................................... 26

3.2.1 Central Controller and Its Interfaces ............................................................... 27

3.2.2 Data Hub and Its Interfaces ............................................................................. 27

3.2.3 Tool Agents and Their Interfaces .................................................................. 28

3.2.4 Locator Component and Its Interfaces ............................................................ 31

3.3 Benefits of the Approach ..................................................................................... 33

3.3.1 Information Management ................................................................................. 34

3.3.2 Agent Management ........................................................................................ 35

4. IBADS: Components and Implementation ............................................................ 38

4.1 Overview of IBADS ............................................................................................. 38

4.2 IBADS Definition Process ................................................................................... 40

4.2.1 Define Interfaces for The Central Controller ................................................ 41

4.2.2 Define Interfaces for The Data Hub ................................................................. 42

4.2.3 Define Interfaces for The Tools ...................................................................... 43

4.2.4 Define Interfaces for The Locator ................................................................. 45

4.3 Implementation of IBADS Components ............................................................... 46

4.3.1 GUI .................................................................................................................. 46
5. IBADS: A Case Study .................................................................63
   5.1 System Deployment .........................................................63
   5.2 Case Study ..................................................................65
      5.2.1 Background ...........................................................65
      5.2.2 Start a New Design .................................................65
      5.2.3 Open Previous Design ..............................................67
      5.2.4 Dynamically Change the Tool-set ............................69

6. Conclusion and Future Work ..................................................71
   6.1 Conclusion ..................................................................71
   6.2 Limitations of This Work and Future Extensions ...............71

References ............................................................................74
List of Figures

Figure 2.1 Components communicate through interface ........................................... 16
Figure 3.1 Linked-Chain Architecture ........................................................................ 22
Figure 3.2 Hub-and-Spokes Architecture .................................................................. 23
Figure 3.3 Overview of distributed tool integration framework ..................................... 33
Figure 4.1 DCOM vs. CORBA ..................................................................................... 40
Figure 4.2 System overview of IBADS ........................................................................ 41
Figure 4.3 Interface diagram of IHUB .......................................................................... 44
Figure 4.4 Interface diagram of EtabsWrapper .............................................................. 45
Figure 4.5 Interface Diagram of Locator ..................................................................... 45
Figure 4.6 Components in IBADS and their interfaces ................................................. 46
Figure 4.7 The normal design process the user goes through ........................................ 47
Figure 4.8 AutoCAD Object Model ............................................................................ 49
Figure 4.9 Class diagram of IHub ............................................................................... 51
Figure 4.10 IDL of interface IParseDrawing ................................................................. 53
Figure 4.11 IDL of interface IPersistDesignInfo .......................................................... 55
Figure 4.12 Format of saved design file ....................................................................... 55
Figure 4.13 IDL of interface IGetBuildingInfo and interface IStandardBeam .............. 57
Figure 4.14 IDL of interface IAnalyze .......................................................................... 58
Figure 4.15 IDL of interface ICodeCheck ..................................................................... 58
Figure 4.16 File recording the information regarding each agent .................................. 59
Figure 4.17 IDL of interface ILocator .......................................................................... 60
Figure 4.18 Class diagram of ETABS wrapper ......................................................... 61
Figure 5.1 Deployment of IBADS ............................................................................ 63
Figure 5.2 Wrapper A and D are registered in IBADS ............................................. 64
Figure 5.3 A two-span, two-story frame structure in AutoCAD .............................. 65
Figure 5.4 Client start new design, drawing in AutoCAD is parsed and labeled ........ 67
Figure 5.5 Client dialog where user starts design and gets results ....................... 68
Figure 5.6 Stress ratio dialog, showing the results of the code-checking by Steeler .... 69
Figure 5.7 The dialog showing the results of the code-checking by Steeler ............. 70
Chapter 1. Introduction

1.1 Purpose
This document describes a new approach to building integrated design environments aimed at simplifying the development and integration of the agents that provide the problem-solving functionality of these environments. The benefits of this approach is illustrated by the description of IBADS, an integrated building analysis and design environment, implemented during this research effort.

1.2 Objective
The objective of this study is to provide a more reliable, flexible and efficient way to build an integrated building design software environment. This must be accomplished in a way that allows different design tools implemented on a variety of hardware platforms and using various computer languages to seamlessly exchange control information and design data. It must also allow the tool-set of the environment to change gracefully, as new tools are added and obsolete tools are removed.

1.3 Motivation
The building design/construction industry in the US is characterized by a dispersed organizational structure in which a number of diverse organizations participate in the planning, design and construction of each building project. This structure brings together a wide range of engineering disciplines (e.g., architecture, structural, mechanical and electrical) and combines them in a complex design process. In order to be successful, this
process requires the cooperation of a large number of individuals (e.g., architects, structural engineers, electrical engineers etc.) that represent these engineering disciplines.

Faced with increasingly global and competitive nature of the construction industry, the Architect/Engineering/Construction (A/E/C) companies are continually seeking new ways to address competitive pressures, shorten design cycles, improve quality and increase productivity. In response to these challenges, the industry has turned to automation as one of the enabling mechanisms for addressing the competitive pressures.

Since the advent of computers, engineers have been using software technology in automating portions of their work. Over the past 30 years, the engineering software community has developed and integrated into practice a large number of software tools that are designed to simplify a variety of tasks in a variety of domains. As a result, all participants involved in today's building design process have access to many types of advanced design software (e.g. CAD systems, analysis programs, knowledge-based design and planning systems). All of this software is used extensively by designers in the performance of their specific design tasks.

The nature of these computer tools and their utilization in the building design process reflect the disjoint and distributed nature of the AEC industry. The vast majority of these tools is developed to address a clearly defined task in a specific domain and lacks any notion of how this task is influenced by the overall design process. Because these tools
are targeted at automating a small portion of the overall process, they are not capable of communicating with anyone other than the designer that is using the tool.

This has lead to a situation that is referred to as "islands of automation", where each person involved in the building design uses automation for the performance of their task but is unable to use similar automation to communicate with other participants in the same process.

A study by the Construction Industry Institute (CII) has shown that computer automation has resulted in significant increase in the productivity of the individual organizations involved in the building design process, but that the productivity gains for the entire development process have been negligible. The primary reason for this lack of significant impact is that design information is distributed among various software tools and isolated within the boundaries of these tools [1].

In order to increase the productivity of the overall design process, the software tools available to different groups of designers must work together effectively in the cross-functional design process that shares information from related parties. However, because of the lack of communication and integration capabilities, the coordination of the design knowledge that is distributed between software tools can be only accomplished with the intervention of a human. Without the facilities of seamless knowledge integration, much of the productivity and efficiency that is gained by the using engineering software is lost due to the problems involved in manual coordination of this knowledge. This issue is
becoming more acute with the increasing use of Internet and the improved network capability of computers, since it has reduced the cost of inter-computer communication.

This work describes a new approach aimed at bridging these "islands of automation" by simplifying the process of building integrated environments. This is accomplished by allowing different software tools to communicate seamlessly, regardless of their network location, language preference and implementation details.

1.4 Organization

The remainder of this thesis is organized as follows:

- Chapter 2, entitled "Background", discusses:
  - The definition of integrated design software environment and reviews related research projects;
  - a set of requirements to address the agent management issues in building such an environment; and
  - two distributed object computing framework DCOM and CORBA.

- Chapter 3, entitled "Distributed Tool Integration Framework", discusses the approach of building an integrated building design software environment that leverages the facilities provided by distributed object computing framework.

- Chapter 4, entitled "IBADS: Components and Implementation", illustrates the implementation and benefits of IBADS, an integrated environment built using the proposed approach.
• Chapter 5, entitled "IBADS: A Case Study", demonstrates the operation of IBADS by giving a case study of design and analysis of a 2-D frame structure.

• Chapter 6, entitled "Conclusion and Future Work", summarizes the results of this study and makes recommendations for possible extensions.
Chapter 2.  Background

Integration is achieved when all components in an environment—the set of software tools, one or more hardware platforms, the problem solving process and the human organization that uses them—function as part of a single, consistent, coherent whole.

A/E/C industry is rapidly moving from isolated computer applications that address individual design and planning tasks to the use of computers in the broad integration of these tasks. This integration involve software tools that address the numerous phases of planning and design within the same discipline (vertical integration) and across the many sub-disciplines involved (horizontal integration) [2].

2.1 Integrated Design Software Environment

As shown in the CII study, increasing the impact of software on the design process requires that software be integrated in a way that supports collaborative design. This involves developing an integrated design software environment that allows engineers to work concurrently with various design software tools running on different computers. It also requires computer support of exchange of design information in an automated and coordinated fashion. The issue of providing computer support for collaborative engineering is significantly different from the issues encountered in developing software aimed at automating a single design task. As stated by Fenves [3], a design environment that supports design integration and collaboration must address the following three issues:
• **Process management.** Process management deals with issues of how a problem that is too complex to be solved by a single tool can be solved by pooling the problem-solving capabilities of multiple tools. Efficient process management strategy implies the ability to plan and carry out the solution to the original problem while eliminating or reducing the number of conflicts that may arise.

• **Information management.** Information management deals with issues of how information is exchanged between software tools that make up the integrated design environment. It deals primarily with issues of how the shared design information is represented and managed. The issue of information management plays a major role in integrated environments that have a high level of concurrency and must ensure that the data used by individual tools is correct and complete throughout the problem solving process.

• **Agent management.** Agent management describes the communication functionality required from each tool in the design environment. These requirements are closely tied to the issues of process and information management because they define how each tool will interact with all other components of the design environments. The issue of agent management plays a major role in environments that mix the off-the-shelf and custom-built software tools. It is also important in the design environments where the tool-set is expected to evolve and grow over time.

**2.2 Review of Related Integration Research**

In recognition of the necessity and difficulty of the integration problem in A/E/C, a number of research projects have been carried out in universities, research centers and
companies, each using a different methodology to address the problem of integration and coordination. Here five important research projects in this area are briefly highlighted.

2.2.1 Integrated Building Design Environment (IBDE)

Integrated Building Design Environment (IBDE) [3] is a research project that was carried out in Carnegie Mellon University. IBDE integrates seven knowledge-based systems, one algorithmic system and one critic into an environment for preliminary design and construction planning of hi-rise buildings. It utilizes a hierarchical planner combined with a market mechanism for task allocation for process management. It also provides a centralized database that manages all information exchanged during the problem-solving process. The main emphasis of this work is to provide a tool-independent mechanism for representing problem-solving knowledge that could be applied to a variety of integrated design environments in the same domain.

2.2.2 Agent-based Framework for Integrated Facility Engineering (ABFIFE)

Agent-based Framework for Integrated Facility Engineering (ABFIFE)[2] is a research project from Stanford University that uses agent-based software engineering approach to create a design environment for preliminary building design. Information management is realized with the help of Agent Communication Language (ACL) to communicate design information and knowledge partially and incrementally among agents. ACL is a formal language proposed as a communication standard for disparate software. It is based on a logic-based language called Knowledge Interchange Format (KIF) and a message protocol called Knowledge Query Manipulation Language (KQML). Design agents are
linked and their communication of design information is coordinated via software tools that are called facilitators. Agents communicate only with their local facilitators and facilitators communicate with each other. In effect, the agents form a "federation" in which they surrender their communication autonomy to the facilitator. Therefore, process management is handled via facilitators who are responsible to direct the control of registered agents.

2.2.3 Circle Integration

This is a research project carried out at Center of Integrated Facility Engineering (CIFE) in Stanford University. It proposed the use of a circle as the architecture for integrating pre-construction activities.[4] In contrast to architectures used in IBDE or IBFIFE, there is no central controller. The process management is handled by forming a circle of tools whose problem solving activities have a single predecessor and a single successor. Thus, integration control involves passing information around a circle of applications. Each application receives information it might choose to ignore or process the individual data it just received and pass them on to the next application in the circle. This way, changes made to the design or construction plan of a facility will be propagated automatically to all other applications. A pass "around the circle" thereby defines and completes one design iteration, making the definition of design versions explicit and simplifying information management.
2.2.4 Intelligent Computer-Aided Design System (ICADS)

Intelligent Computer-Aided Design System (ICADS) [5] is a prototype model of a computer-aided architecture design system intended to explore the potential of partnerships between computers and human designers. This system aids in developing schematic floor plans during conceptual design by horizontally integrating a CAD drawing environment with six knowledge-based systems specialized in daylighting, sound control, structural system selection, thermal behavior, cost and spatial access. One important aspect of the ICADS model is an intelligent design assistant that provides process management strategy through continuously monitoring the evolution of the current design solution. This design assistant has the capability to interrupt the human designer under certain circumstances. The ICADS model uses a blackboard control system that consists of two experts: a message router for communicating information and a conflict resolver, which makes counter suggestions to the evaluation provided by different knowledge bases [2].

2.2.5 Knowledge-Aided DataBase management system (KADBASE)

Knowledge-Aided DataBase management system (KADBASE) [6] is a flexible interface intended to provide a knowledge-based communication between multiple expert systems and databases. This project has focused on information management issues, through the help of a central database. The main components of KADBASE are a network data access manager (NDAM), which handles and processes all data queries and updates initiated by databases and knowledge bases, knowledge-based system interfaces
(KBSI) for semantic and syntactic translations for knowledge-based systems, and knowledge-based database interface (KBDI) for semantic and syntactic translations for database systems. When linked to KADBAS, various knowledge-based systems can obtain their design data from the databases through KADBAS by issuing query [2].

2.3 Agent Management

The five systems described above have concentrated on the issues of process and information management. In KADBAS, ICADS, and Circle Integration project, the issues of agent management are ignored completely. IBD and ABFIFE have looked at some issues in agent management but only in so far as they impact the representations and management of the process management knowledge used in an integrated environment.

The increasing use of the Internet in AEC industry presents a major shift for the use of computing in the building design process. Until now, most computers in use in design offices offered no or limited networking capabilities, thus restricting the amount of computer-based interaction that could be accomplished. As a large number of these computers are being upgraded with networking capabilities and connected to the Internet, there is an increasing interest in combing their hardware power and software tools into design environments that span both single design offices and across design offices. In order to realize this vision, there is a need to both create new software tools capable of taking advantage of the Internet and to extend existing tools to be able to communicate across the networks. Because of this, the issue of agent management is becoming more prominent in the design of integrated environment.
Given the fragmented nature of the building industry and software tools used in it, the integrated software environment targeted at A/E/C applications should be able to address the following specific issues regarding agent management:

- **Support for software written in different languages.** The agent management strategy must accommodate software tools written in a variety of languages. Historically, the majority of software tools used in A/E/C have been written in Fortran software language, but in last several years there has been a growing trend to use BASIC, C, C++ and Java programming languages for engineering applications. As a result, an agent management strategy should provide support for all software regardless of their software language. If this can not be accomplished, an agent framework should support the most frequently used software languages listed above.

- **Support for software running across the network.** Because of rapid adoption of Internet technology and availability of network-capable computers, engineering software is now accessible across wide area networks. A successful agent management strategy must support situations where the applications in an integrated environment are geographically distributed and must be able to utilize the network infrastructure to support the interaction between these applications.

- **Support commercial off-the-shelf (COTS) software as well as legacy systems.** As software technology continues to evolve, more and more high-quality COTS packages will be available on the market. At the same time, design offices have developed legacy systems representing thousands of staff-years of domain expertise. These applications are
strategic assets that must be accommodated by an integrated environment [7]. The agent management strategy for a design environment must be able to support the combination of COTS, legacy software and software written specifically to function as part of the environment.

- **Flexibility of the tool-set.** Since the field of A/E/C software is expected to undergo a number of significant transitions, it is likely that the software tools found in a typical design office will change on a periodic basis. As a result, an integrated design environment should be able to gracefully accommodate the changes to the tool-set. This implies two sets of functionality required from a successful agent management strategy:

  1. The changes to the tool-set of an environment should minimize the impact on process management and information management strategies.

  2. When multiple similar software tools are available, the agent management strategy should allow the user to dynamically select the most suitable one.

- **Maintainability.** The agent management must also allow the tools in the tool-set to evolve without impacting other aspects of the environment. Since it is expected that if a tool is not replaced, it will be periodically updated, the agent management must provide a separation between the communication functionality provided to make the tool’s knowledge available to other components in the environment and the details of the implementation of the tool.

### 2.4 Agent Management and Distributed Object Computing

As described above, the current integrated design environments generally neglect the issues of agent management. Environments that do provide some agent management utilities, such as IBDE and ABFIFE, only provide low level application programming
interfaces (APIs) that define how tools in the environment are expected to communicate. In most cases these APIs are based on the lowest level of communication available on the network such as Transmission Control Protocol/Internet Protocol (TCP/IP) sockets. These APIs are targeted mainly at providing the flexibility aspects of agent management by reducing the linking between the process and information management strategies and the tools in the environments’ tool-set. This approach to agent management places significant development burden on adding new software or changing existing tool in the design environment. Therefore, systems integrated in this manner are usually closed (custom proprietary solution), slow to develop, expensive to maintain and evolve and lack reusability.

A number of these issues could be addressed by building an integrated design software environment built by leveraging the existing distributed object computing technology.

2.4.1 Distributed Object Computing

Object-oriented technology was introduced to help software developers to better express the problem and concepts by modeling the real world entity through programming structure—object. Programs are composed of object which is a programming module consisting of its own private information (data), its own private procedures (private methods) that manipulate the object’s private data and a public interface (public methods) for communicating with other objects. In other words, an object contains both data and processing logic in a single software entity. One of the main advantages of this approach is the ability to reuse the same object definition in developing multiple programs.
While a single program can reuse an object definition in multiple places, multiple programs either located on the same machine or distributed on the network can not share access to a single object. Standing in the way of this vision are three major problems [8].

- Standards for linking binary objects together do not exist. In order to link and reuse the binary object, source code must be provided.
- It is difficult to link and reuse an object written in one language with another object written in a different language.
- Modification of an object's implementation requires other participants in the system to recompile or re-link.

As a result, the wide scale adoption of the object-oriented technology has produced "islands of objects" that can't talk to one another across the sea of application boundaries in a meaningful way [9].

Distributed object technology has been developed as a solution to this issue. In distributed object systems, the unit of work and distribution is a component. A component is a reusable piece of software in binary form (as opposed to source code) that communicate with each other through a set of predefined interfaces, which is a group of related methods the component supports. Components are very similar to objects, but unlike traditional objects, components can inter-operate across languages, tools, operating systems, and networks. This is accomplished by augmenting the components with a standard infrastructure that takes care of all the low level details required to make the
inter-component communication possible. Components live and act in this standard infrastructure and communicate with each other through interfaces (Figure 2.1).

![Diagram](image)

**Figure 2.1** Components communicate through interface

Today, there are two candidates for a standard infrastructure to support component inter-communication: the Microsoft Distributed Component Object Model (DCOM), and Object Manage Group's Common Object Request Broker Architecture (CORBA).

**CORBA.** The Common Object Request Broker Architecture (CORBA) is an open distributed object computing infrastructure being standardized by Object Management Group (OMG), a consortium of more than 800 software, hardware and end-user developer companies. It is a standard for the distribution of objects across heterogeneous networks and designed as a platform-neutral infrastructure for inter-object communication.
The CORBA specification is created to support the definition of services needed by OMG’s vision of a completely distributed Object Management Architecture (OMA). OMA object model strictly separates an object’s interface from its implementation. Objects communicate with each other through their interfaces. OMG offers the Interface Definition Language (IDL) as an abstract, symbolic language, for specifying the object interfaces, including their complete signatures of methods as well as the names and types of accessible attributes. The interface specifications written in IDL can in turn be translated into the constructs of a desired programming language, such as C++. These constructs will provide all the necessary functionality for inter-component communication.

*Distributed Component Object Model (DCOM).* DCOM is Microsoft’s component software architecture. It is a binary compatibility specification and associated implementation that allows clients to invoke services provided by DCOM-compliant components (DCOM objects) [10]. It is an object-based programming model designed to allow development of software components at different times by different vendors using a variety of languages, tools, and platforms. Once developed, DCOM components can be easily deployed and integrated.

DCOM objects can launched in a separate process on the same machine or any machine on the network. As a result, DCOM components will be able to communicate with other components, no matter where they live. The underlying communication is handled by DCOM automatically.
DCOM interfaces are defined by using Microsoft's Interface Description Language (MIDL) which in turn can be translated into their corresponding C++ header files and a binary library that can be used by other languages.

The Distributed Object Computing technology was developed to support interactions of objects that reside in different processes on different machines. Both CORBA and DCOM accomplish this by abstracting the code as an object that has a specific set of interfaces and providing a mechanism for invoking those interfaces from external code that may be written in a different programming language. A similar approach can be used to implement an agent management strategy in a design environment. The next chapter describes a new approach that uses the facilities and philosophy from Distributed Object Computing to provide a blueprint for agent management in an integrated design environment.
Chapter 3. Distributed Tool Integration Framework

This chapter outlines a new approach to integrating a set of distributed tools into an integrated design software environment. It also presents a number of benefits this approach brings to build an integrated design software environment.

3.1 Overview of the Approach

As described in chapter 2, three major issues need to be considered in building an integrated building design software environment: process management, information management and agent management.

3.1.1 Process Management

Process management deals with insuring that the tools in the environment cooperate in the solution of the overall problem. Process management requires establishing a control strategy aimed at reducing the amount of conflicts in the problem-solving process and a conflict resolution strategy for situations where the control strategy fails. The process management in design environments can be classified into the following four categories:

- User driven. The environments that use the user-driven process management strategy rely on the human expertise to address all control and coordination issues. This strategy is used in situations where the tasks assigned to the environment have a large amount of variability. In these cases, programmed process management strategy is of little use and control decisions are best left to the user of the system. Environments that use this type of process management are more difficult to use since they only address information management and agent management components of integration.
- Hard coded. The environments that use hard coded process management strategy have a well-defined control strategy that is not expected to produce any conflicts. This type of strategy is easy to develop and implement, but the vast majority of design tasks can not be partitioned in a way that all conflicts are eliminated. This approach to control strategy is also difficult to extend to accommodate new tasks.

- Data driven. Data driven control determines the control strategy based on comparison between the current state of the system and the goal state. This strategy allows each tool to decide whether they can make a contribution to the current state of the system and to volunteer their services. If several tools volunteer to act, either all of them are allowed to proceed or the user is expected to choose one of the tools. This strategy is very flexible and frees the environment from the task of conflict resolution, but it is possible to encounter processing loops where tool's contributions are removed by downstream execution of other tools.

- Goal driven. Goal driven process management tries to identify and execute a sequence of tasks required to achieve a stated goal. Design tasks are sequenced with the help of a planner making this system very flexible and efficient. However, this strategy requires the creation and implementation of an effective conflict resolution strategy to accommodate feedback loops in the original plan.

The approach to building an integrated environment, which is proposed in this chapter, is targeted at design environments that solve a well-defined task and include a very small number of tools. As a result, it targets at environments that use a mixed process management strategy that includes elements from both user driven and hard coded
approach. This class of environments uses a central controller that receives directives from the user and uses its internal control logic to guide the execution of these directives. This control structure can be classified as single-hierarchy architecture, and is common to a large segment of design environments developed for civil engineering tasks.

3.1.2 Information Management

Information management deals with issues of how the information is represented and exchanged among tools. If an integrated system is composed solely of tools that are specifically developed to work together, the tools can be made to share the same representation of the building, therefore simplifying the information exchange issues. However, the majority of integrated design environments are developed to integrate COTS software packages that were developed for stand-alone use. Due to the fragmented nature of the A/E/C industry, each COTS package developed by different vendors using a proprietary representation of building. This complicates the issues of information management in an environment that mixes COTS software. In order to exchange information between software packages, the integrated design environment must provide a translation mechanism between the representations used by its tool-set. There are two possible approaches for organizing the translation and information management in such environments: the linked-chain architecture and the hub-and-spokes architecture.

Linked-Chain Architecture

The linked-chain architecture (Figure 3.1) identifies all data flow paths needed in the system and provides data translations for all links in the data flow path.
Figure 3.1 Linked-Chain Architecture

This approach is best suited for environments where the problem-solving strategy and the tool-set are not expected to change. Because of the requirements for point-to-point translation, any change to the information flow in the environment can require creation of new translators. In addition, this approach makes it difficult to identify the current state of the design and to ensure that all information in the environment is up to date.

"Hub-and-Spokes" Architecture

The hub-and-spokes architecture maps the output of all tools to a common representation (the hub) and generates input information for the tools by extracting the required information from the common representation and passing it to the agents (the spokes). A central representation is provided for all data used by the environment. Translators are only needed between each tool's local representation and the hub (Figure 3.2).
This hub-and-spokes architecture provides a great deal of flexibility by limiting the number of translators that are needed to support the evolution of the process management strategy and the tool-set. The only time a new translator is added is when a new tool is added to the tool-set of the environment. In this case two translators need to be added to the environment: an input translator capable of generating the input file for the tool based on the information in the hub and the output translator capable of parsing the information in the tool’s output file and updating the design information in the hub.
The additional advantage of this architecture is that the hub provides a central location that always provides the current state of the design and is a logical place for the implementation of facilities such as version control and consistency checking.

One key issue in implementing the hub-and-spokes architecture is developing the common data representation that can support the information needs of all tools expected to form the environment. Since each tool has its own representation and the tools are changing, the central building representation has to be able to accommodate the differences among the individual tool's private representation. Therefore, the common building representation must be created after analysis of an overall task and not as union of a specific building representations used by a specific tool-set.

The approach to building an integrated environment, which is proposed in this chapter, is designed to support the hub-and-spokes architecture for information management. This architecture was selected because of its ability to support the evolution of the tool-set. Since the A/E/C community has not settled on a fixed set of tools for building design, the evolution of tool-set is expected to be the norm for any design environment.

3.1.3 Agent Management

Agent management strategy defines the communication and collaboration functionality that is required from all participants in the environment. It provides guidelines for building tools designed specifically to function inside an environment and tools for converting
existing stand-alone tools into agents capable of interacting with the environments. A successful agent management strategy must address the fact that these tools can be written in different languages, run on different machines at different locations.

Traditionally, integrated design environment provides limited support for agent management. As a result, developers who wanted to develop new tools or to integrate existing tools into a design environment must deal with low-level detail of network infrastructure, machine incompatibility and language. This process is very time consuming, difficult and error prone.

As described in chapter 2, the distributed object technology (DOT) can be leveraged to provide the functionality needed to simplify the task of agent management. DOT treats any piece of software as a component that can be used as stand-alone module or accessed by other components. The components expose their functionality by defining a set of public interfaces that can be invoked by other components. Components can inter-operate across languages, operating systems, and networks at high-level through the manipulation of the interfaces. Communication details such as network location, data type representation and language difference are all handled by the underlying distributed object infrastructure at system level, providing the developer a much clean, standard and high-level way of integrating the various components. Therefore, the notion of component, with the help of DOT, emerges as an ideal candidate for building agents in the design software environment where agents need to be able to communicate with language, platform, and location transparency.
In summary, this approach targets at centrally-controlled process management strategies and hub-and-spokes architecture for information management. The DOT facilities are leveraged to address the issues of agent management. This approach requires that each participant in the environments is either created from scratch as a DOT component or it can be wrapped as a DOT component from a stand-alone program. Agents participate the design process and exchange design information among each other, therefore, the fact that an agent is built as a DOT component and communicate with the help of DOT will inevitably affect the implementation of process management and information management strategy in building the integrated environment. The following section will explain in detail the proposed approach of building an integrated design environment by leveraging the facilities of DOT.

3.2 Approach Description

The proposed approach includes the following five phases of development:

- Analyze the function of the environment and define the interfaces needed by the central controller to implement the selected control strategy
- Analyze the functions of a central data hub component and define the interfaces needed to support its functionality
- Analyze the capabilities of the tools in the tool-set and define the interfaces that they must support in order to perform their task
- Analyze the capabilities of a locator component needed to dynamically link the various participants in the environment and define the interface needed to support this function
• Implement new tools or extend the current tools to support the sets of interfaces developed in the previous four stages.

3.2.1 Central Controller and Its Interfaces

Each design environment functions to address a set of design tasks towards the solution to an overall design problem. Therefore, the tasks needs to be identified first. Suppose two design tasks have to be addressed, taskA and taskB. A central controller component should therefore be designed with two interfaces, ITaskA and ITaskB to address the two tasks respectively. Since the central controller works as a portal to direct requests to each specific tool, methods in these two interfaces are implemented to forward to the task-solving functionality of each corresponding tool in the environment.

3.2.2 Data Hub and Its Interfaces

For environment built with hub-and-spokes architecture, there will be a central data hub, which maintains a central building representation. This data hub component therefore needs to support the functionality of creating the building representation, retrieving and updating information regarding the building from/to the central representation, and saving the building representation to a persistent storage. This functionality can be exposed through a set of interfaces, ICreateBuilding, IAccessBuilding and IPersistBuilding respectively. Depending on the specific implementation, these interfaces can have a number of variations. For example, ICreateBuilding can be expanded into two interfaces, ICreateBuildingFromDrawing and ICreateBuildingFromStorage, indicating there are two ways of creating a building representation, either by parsing an AutoCAD drawing or from
a persistent storage. ICreateBuildingFromStorage can also be merged with IPersistBuilding as a specific method in it, considering the fact that both interfaces are handling the similar read/write operation on persistent storage.

3.2.3 Tool Agents and Their Interfaces

Each tool serves a specific function to solve a design task. There are two kinds of tools in a design environment:

- tools that are built specifically for the environment
- tools that are existing legacy system or COTS packages

Both kinds of tools will be built as DOT components and expose their functionality through a set of interfaces. For the tool that is built specifically for the environment, it can be built from scratch as a DOT component and, depending on its functionality, it will support a standard interface with methods to perform that function. For example, for a tool performing taskA, a standard interface IToolTaskA will be specified with standard method performTaskA(). All other tools that are designed to solve taskA will have to support this standard interface too in order to be recognized as agents for solving taskA. Since the interface is specified as a standard interface, any tool that supports this interface can then be plugged into the system and replace an old tool with the same functionality.

A tool might support multiple interfaces corresponding to the multiple functionality it supports. For example, it may implement IToolTaskA and IToolTaskB simultaneously. It
therefore can be recognized both as an agent for tool A and an agent for tool B, and be selected to solve both the two tasks.

For existing legacy systems or COTS packages, a wrapper component needs to be built around the tool to make it act like an agent in the system. A wrapper is a custom built DOT component that wraps the functionality of the existing tool into a standard IWrapper interface. All tools in the design environment share the same execution pattern: read an input file, run the tool and generate the result into an output file. In recognition of this pattern, a standard IWrapper interface is defined with three methods, namely, prepareInput(), run() and parseOutput().

Each tool acts like a spoke in hub-and-spokes architecture. As described in section 3.1, a translator is needed between local building representation the tool has and the central building representation the hub maintains. For a tool built from scratch, it could select its local representation to be the same as that of the central hub, therefore, the translator can be eliminated. However, for existing tools, a translator is a must since the two representations will not be the same in most of the cases. Therefore, in the wrapper method prepareInput(), the central building representation will first have to be translated into the local representation that the tool understands and lay out the requested format for the input file. Method run() will issue a system call to the execution command of the tool. In parseOutput() method, after reading the results from the output file into the local building representation, the central building representation needs to be updated accordingly. Since data hub is a DOT component and provides an IAccessBuilding interface to retrieve and
update building information, the update can therefore be done easily through manipulation of the IAccessBuilding interface of the hub.

Each existing tool acts like an agent with the help of wrapper component through standard IWrapper interface. If the tool is for task A, the implementation of methods prepareInput(), run() and parseOutput() will be specific to task A. However, if the tool is for task B, the implementation of these three methods will then be specific to task B. Any tool can participates the systems as an agent as long as it supports the standard IWrapper interface. The actual implementation of the interface varies depending on the specific tool. This provides us a generic approach to wrap various specific tools. However, there are three problems in this approach:

- If a tool supports both taskA and taskB, one implementation of IWrapper will only be able to expose one functionality of the tool, not both;
- The tools are strictly restricted to the three generic methods. Suppose there is an analysis tool, that supports two methods of analysis – normalAnalysis() and advancedAnalysis(), only one of the two methods will be wrapped in IWrapper;
- The fact that existing tool supports IWrapper while custom-built tool supports IToolTask will make the system coupled with one type of tool. It will therefore reduce the flexibility of the system.

Based on the above observation, the following interface hierarchy for both wrapper component and built-from-scratch tool component is proposed:
• Each tool must support standard IWrapper interface with prepareInput(), run() and parseOutput();

• Each task specific interface IToolTaski must be inherit from the IWrapper;

• Each tool may optionally support IToolTaski.

By imposing the above inheritance hierarchy on every tool component, be it a custom-built DOT component or a DOT wrapper, the above mentioned problems can be solved,

• A tool can support multiple tool specific interface IToolTaskA and IToolTask2 at the same time, if it would like to expose all the function it supports;

• Each tool can support tool specific interface if it would like to expose more specific functionality;

• Both custom-built tool component and wrapper component share the common interface IWrapper. The system will not notice any difference between the two kinds of tools, therefore, it is de-coupled from any specific type of tools

3.2.4 Locator Component and Its Interfaces

A central controller, a data hub and various tool agents has been designed. The system can then be built with these existing tool agents. However, if the tool set is changing dynamically, the system has to change accordingly. A locator component is therefore proposed to separate the specific tool set from the system.

The role of locator is to find a matching agent dynamically at run time according to the functionality requested. Each tool will then register the specific information about its name,
ID, functionality and location into a central table upon its joining the system. When it leaves the system, it de-registers itself by removing its information from that table.

Locator is developed as a DOT component with interface ILocator. ILocator has one method locate(), where locator will search the table at run time, gather the specific information about an agent based on its functionality and return the reference of its IWrapper interface to the requestor.

Locator serves as the glue to bring central controller, data hub and various tool agents together. The central controller creates locator, and asks locator to find the data hub component. It then asks locator to locate an agent dynamically based on the requested functionality. The central controller will then contact the agent with the reference to the data hub. The agent can therefore carry out the translation, execution and update functions based on the run-time communication between the hub, controller and the tool.

In summary, as illustrated in Figure 3.3, central controller, data hub, tools and locator are designed as DOT components in order to build an integrated design software environment using DOT. A set of standard interface regarding central controller and data hub are specified. An interface hierarchy of building a tool agent, either custom built or by wrapping an existing tool, is proposed. Each tool must support a generic IWrapper interface and could optionally support one or more extended standard interfaces based on the
functionality the tool wants to expose. A locator component is created to separate the specific tool-set from the system implementation and therefore allows the dynamic changing of tool-set.

3.3 Benefits of the Approach

The previous section outlines a way of building an integrated design software environment based on DOT. It brings various benefits to the design environment.

![Diagram of distributed tool integration framework](image)

**Figure 3.3** Overview of distributed tool integration framework
3.3.1 Information Management

The data hub component uses the network features of DOT to simplify the information exchange between the components of the environment. In a typical design environment, each component that forms a spoke in a hub-and-spokes information architecture requires two classes of translators: a translator between its own representation and that of the hub, and a translator capable of updating the information stored at the hub with the new information received from the tool. By using DOT, the integrated design environment removes the need for the latter translator and allows the translators associated with each component in the environment to directly update the information stored in the hub through interface IAccessBuilding. This can be accomplished without explicit knowledge about the network location of the hub.

The data hub component defines a set of standard interfaces that consist of operations that can be used to manipulate the contents of these objects. Distributed object framework creates a mechanism where these interfaces can be accessed directly from any other component that uses DOT, regardless of its network location, hardware platform and programming language. This allows all the translators implemented in the environment to manipulate the contents of the data hub directly without concern for its implementation detail. In addition, the interfaces provided by the data hub do provide generic operations supported by the hub's representation of the frame and are independent of the hub's actual implementation of information storage. As a result, the developer of data hub can easily change the structure of the objects used to store the current state of the frame without requiring any changes to the translators that access this state information.
3.3.2 Agent Management

Design environment implemented with the help of DOT provides support for all five aspects of successful agent management strategy.

**Support for software written in different languages.** The component technology provides supports for a variety of programming languages. These include C, C++, Java etc. This gives the developers the flexibility to choose any language they like to implement their tools and allows modifications of existing tools to allow them to operate as part of the environment.

**Support for software running across the network.** The network communication aspect of agent management leverages the capabilities of DOT. The details of the communication is hidden from the user allowing the developers to build components without regard for how and where these components will be deployed. The same code is used to establish communication between components that run on the same machine as used to establish communication across the network. In addition, DOT provides built-in support for a variety of security protocols ensuring that the inter-process communications that travel over open networks are secure.

**Support commercial off-the-shelf (COTS) software as well as legacy systems.** Existing tools act like agents through the help of DOT wrapper component. As mentioned in section 3.3.1, the translation functions of the wrappers is greatly simplified because data hub
component provides an interface IAccessBuilding for all data manipulation operation allowing the wrappers to modify the central design information regardless for its actual location or the network protocol used to communicate this information. The wrapper is developed as a DOT component and exposes its set of functionality by defining a public interface that can be directly accessed by a central controller. Again, this interface can be accessed without concern for location, hardware or language that is used to implement the wrapper and the legacy tool that it is associated with.

**Flexibility of the tool-set.** Flexibility aspect of agent management can be achieved by reducing the knowledge about the tools that is embedded in the process and information management strategy of the environment. In the proposed approach, this is achieved through locator component. The locator component provides a linkage between the process management strategy and the tool-set. Each new tool integrated in the environment must support the standard IWrapper interface and be registered with the location service. As described in section 3.2.4, the agent is selected dynamically at run time by locator based on the specific information of each tool. The locator is the only component that has knowledge about all other tools in the tool-set, all the other components in the system have no knowledge about the existence of other agents. Therefore, tools can be added/removed dynamically to the tool set without affecting other participants, which brings high flexibility to the system.

**Maintainability.** The notion of the public interface required from all components in DOT provides a level of separation between the implementation of the component and the
communication interface used to invoke its processing. This separation provides a great deal of maintainability since the remaining tools in the environment will not be effected by any change in the internal operations of the component. The only element of the component's implementation that must remain unchanged is the public interface that is used by all other components to contact the component.

In summary, in this chapter, an approach to building an integrated design software environment using distributed object technology is outlined. The environments this approach targeted at are those that use central-control strategy and use hub-and-spokes architecture to manage information exchange. Various benefits this approach brings to information management, and especially to agent management issues in building an integrated design environment are also discussed.

In next chapter, this approach is going to be applied to implementing a design environment for the analysis and design of 2-D frame structure.
Chapter 4. IBADS: Components and Implementation

In this chapter, an Integrated Building Analysis and Design System (IBADS) is built following the approach outlined in chapter 3. The implementation detail of each component in IBADS is then explored.

4.1 Overview of IBADS

IBADS is an integrated building design software environment designed to illustrate the approach proposed in chapter 3. It vertically integrates a set of highly distributed and loosely coupled tools into a design and analysis environment for 2-D frame using distributed object computing framework.

The tools integrated in IBADS form a mix of COTS standalone software and software specifically developed for this class of design environments. The components of IBADS include:

- AutoCAD. AutoCAD is a commercial off-the-shelf (COTS) software package that is very common in the A/E/C industry. AutoCAD is a drafting package and is used as an input and presentation tool in IBADS.

- Etabs. Etabs is a module in ETABS package. It is a COTS software package that performs linear, nonlinear, static and dynamic analysis of the building. Etabs is used to provide the analysis capabilities in IBADS [11].

- Steeler. Steeler is another module in ETABS package. It is a COTS software package that performs code checking for steel frames. Steeler is capable of evaluating design
conformance of a frame or building based on such design codes as UBC 94, CISC 89 and LRFD 93. In this version of IBADS, UBC 94 is chosen.

- **Graphical User Interface (GUI):** This software tool is written specifically for IBADS. The GUI provides the user with the mechanism to direct the problem-solving activities of IBADS and to view the results of these activities.

- **Information Hub (IHUB).** This software tool is written specifically for IBADS. The IHUB manages the information and problem-solving strategy encoded in IBADS. It can be implemented as two separate components, a central controller and a data hub.

- **Locator.** This software is also written specifically for IBADS. It is designed to link the problem-solving strategy in the IHUB with the other components in the environment. The locator is the only component that is aware of the presence of other components in the environment.

In building IBADS, DCOM is chosen over CORBA as the underlying integration infrastructure due to the following reasons:

- According to [12], DCOM provides support in more programming languages and network protocols, as shown in Figure 4.1.

- DCOM is a standard and also an implementation. It is implemented as part of the Windows OS. Every PC running 32-bit versions Windows have some levels of DCOM support built in, providing a compelling argument for its use in Windows environments.
- Etabs, Steeler and AutoCAD are all running on Windows NT platform. Besides, AutoCAD 14.0 is built as a DCOM component already.

- Microsoft Visual Developer Studio (VC++, VJ++, VB etc.) provides great built-in COM support, which will significantly simplifies the application development.

Therefore, DCOM is selected as the underlying component architecture in IBADS, to utilize the strong support of DCOM in Windows environment. Each tool in IBADS acts as an agent after being turned into a DCOM component.

Figure 4.2 describes the system overview of IBADS.

<table>
<thead>
<tr>
<th>Features</th>
<th>DCOM</th>
<th>CORBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Languages</td>
<td>C, C++, VB, Java, Fortran, Cobol, Perl, REXX, JavaScript, many others</td>
<td>C, C++, ADA, No VB, multiple Java bindings, no scripting languages</td>
</tr>
<tr>
<td>Network Protocols</td>
<td>TCP, IP, IPX, SPX, HTTP, many others</td>
<td>TCP only</td>
</tr>
</tbody>
</table>

**Figure 4.1 DCOM vs. CORBA**

### 4.2 IBADS Definition Process

According to the approach listed in chapter 3, there are five phases in building IBADS.
4.2.1 Define Interfaces of Central Controller

In step one, the set of tasks IBADS needs to address will have to be identified first. Two simple tasks are selected, Analyze and CodeCheck. Task “Analyze” is to carry out the static structural analysis of the 2-D frame structure to get the internal forces of each member and the story displacements. Task “CodeCheck” is to check the analysis result according to building code UBC 94 in terms of stress ratio of each member. Stress ratio is the ratio of the actual stress over allowable stress level defined by UBC 94.

IHUB is built as a DCOM component to serve as the central controller. It has to implement two interfaces, IAnalyse and ICodeCheck, in support of the two design task.
To be more specific, IAnalyze has two methods, InputExternalForces() and AnalyzeStandardBuilding(). Design forces are given through method InputExternalForces() and AnalyzeStandardBuilding() to forward the task of “Analyze” to corresponding analysis agents. Similarly, ICodeCheck supports one method codeCheck(), which internally forwards the task of “CodeCheck” to the corresponding code checking agent.

4.2.2 Define Interfaces of Data Hub

Data hub component and its interfaces need to be specified in this step. In the current implementation of IBADS, data hub and central controller are chosen to be combined as a single DCOM component—IHUB, due to the following reasons:

- The process control strategy taken is relatively simple and the research focus of this study is on agent management issues, not on process management or information management issues;

- It can demonstrate that combining or separating central controller and data hub do not bring much difference to the normal operation of the system, as long as the corresponding set of interfaces are supported.

Therefore, besides the supported IAnalyze and ICodeCheck interfaces, targeted at the control strategy, IHUB has to support another set of interfaces to address information management issues. ICreatBuildingFromDrawing interface is designed to create the central building representation from AutoCAD drawings; IAccessBuilding is designed to retrieve and update the building representation and IPersistStorage to build and save
building information from/to persistent storage. In the current implementation of IBADS, the three interfaces are named differently as IParseDrawing, IBuildingGetInfo and IPersistDesignInfo respectively.

IParseDrawing has one method parseBuildingDrawing() to create the building from parsing an AutoCAD drawing; IBuildingGetInfo has eleven methods for accessing and updating the building representation; IPersistDesignInfo has two methods saveDesignInfoToFile() and getPreviousDesign() to deal with the read/write from persistent storage.

Figure 4.3 shows the interface diagram of IHUB component.

### 4.2.3 Define Interfaces of Tools

In this step, interfaces that various tool agents must support need to be defined. In IBADS, the tool-set includes AutoCAD, Etabs and Steeler, all of which are COTS packages. AutoCAD is developed as a DCOM component, but Etabs and Steeler were developed as stand-alone COTS packages. Therefore, IBADS must provide wrappers to integrate these tools. The wrappers support standard IWrapper interface and/or
Figure 4.3 Interface diagram of IHUB

extended interface IToolAnalyze and IToolCodeCheck. IWrapper is the generic wrapper interface with three methods, prepareInput(), run() and parseOutput(). These three methods are implemented to wrap each specific functionality the tool provides. IToolAnalyze and ICodeCheck inherits from IWrapper interface and they support more specialized analyze() and codeCheck() methods respectively besides the generic methods provided in IWrapper. This approach allows wrapper components to support more
specialized functionality while maintaining the generality of a common wrapper. Figure 4.4 shows the interface diagram of a wrapper component.

![Interface Diagram of EtabsWrapper](image)

Figure 4.4 Interface diagram of EtabsWrapper

4.2.4 Define Interfaces of Locator

In this step, a locator component needs to be defined to glue the various components in IBADS together. As shown in Figure 4.5, locator supports ILocate interface with one method locate().

![Interface Diagram of Locator](image)

Figure 4.5 Interface Diagram of Locator

After these four steps, the interfaces of the major participants in IBADS are defined, as shown in Figure 4.6. Each component then needs to be implemented to support the above
defined interfaces. Besides, a GUI needs to be added to deal with the user interaction with the system. The implementation of these components will be explored in the next section.

![Diagram of IBADS components and their interfaces](image)

**Figure 4.6** Components in IBADS and their interfaces

### 4.3 Implementation of IBADS Components

#### 4.3.1 GUI

The GUI component allows the user to interact with the system. It does not provide services to other participants in IBADS so no public interface is exposed to the environment. The GUI window is composed of three related dialogs which are presented to user at various stage of the design. Figure 4.7 shows the flowchart for a normal design process and identifies the corresponding dialogs that are presented to the user by GUI.
Figure 4.7 The normal design process the user goes through
The first dialog is displayed to the user when he starts an IBADS session. Using this dialog the user can choose to open a previous saved design or to start a new design. If the user chooses to start a new design, he must specify an AutoCAD file that represents the frame to be designed by IBADS. Otherwise, the user may specify a name of the building that was defined in a previous session. The second dialog provided by GUI allows the user to specify trial cross-section geometry for beams and columns and provide external forces for analysis. The contents of this dialog are created at run-time and reflect the number of building elements and bays defined in the AutoCAD drawing.

This dialog is also used to display the results of the analysis. The final dialog shows the results of the code check operation. If the design does not satisfy the UBC94 code, the reason will be shown to the user, suggesting a redesign on the building and the user can update the design by selecting different cross-sections of beams/columns, and repeat the analysis and code checking process. At any point in the overall process, the design and analysis results can then be saved in the central data store that is located in the IHUB.

4.3.2 AutoCAD

AutoCAD is a widely used commercial drafting tool that is used to enter and display the frame being designed by IBADS. The frame is defined based on three entities, defined as blocks in AutoCAD, namely, beam, column and the support block. The user can define a frame in AutoCAD by inserting and positioning a combination of these three block types.
The process of integrating AutoCAD into IBADS is simplified by the fact that AutoCAD is designed to serve as both a stand-alone program and a DCOM component. The object model used by AutoCAD (shown in Figure 4.8) can be accessed using the public interfaces associated with each object. For example, object BlockRef supports interface IAcadBlockRef which includes methods like getName, getInsertionPoint, getXScaleFactor and getRotation etc. Each block entity in AutoCAD drawing supports these methods. Any DCOM component that has access to the interface IAcadBlockRef of a block in an AutoCAD file can get the information of the block's location, length, and orientation by invoking the getInsertionPoint, getXScaleFactor and getRotation methods that are defined in the IAcadBlockRef interface.

![AutoCAD Object Model Diagram](image)

Figure 4.8 AutoCAD Object Model [13] (partial)
4.3.3 IHUB

As described in section 4.1, the IHUB component serves two functions in the operation of IBADS: it serves as the hub in the hub-and-spokes architecture for information management strategy and the central controller to provide the process management strategy. IHUB is developed specifically for IBADS and is designed to function as a DCOM component. It has an internal data structure which represents central design information and the control flow of the design process, as shown in Figure 4.9.

**IHUB as a Data Hub**

The IHUB component of the environment serves as a hub for the information management strategy and manages the common representation developed for the task of analysis of 2D structural frame. The IHUB component provides the following elements of information management:

- Object-oriented representations of structural frame consisting of objects representing the beams and columns that form the frame.
- Support for persistent storage of the design information. The IHUB component provides a mechanism for disk storage and retrieval of the design information. This features provides IBADS with the ability to terminate and restart the operations of the environment without any information loss. It also provides a mechanism for future implementation of the version control strategy.

The central building representation is implemented as a collection of objects with the CStandardBuilding object as main entry point in navigating this collection. A frame is
Figure 4.9 Class diagram of IHub
A node is defined with attributes like nodeNumber, location and whether or not it is a support. The CStandardNode class implements the IStandardNode interface, which is implemented as a nesting class XStandardNode. This interface provides accessor operations like getXLocation and mutator operations like setNodeNumber. These operations are designed to allow external components to access and update the node’s state.

Similarly, CStandardBeam and CStandardColumn support the IStandardBeam, IStandardColumn interfaces respectively. These interfaces allow external components to access and modify information regarding each beam and column.

As part of its responsibility as a data hub, the IHUB is responsible for creating the central building representation and providing a mechanism that allows other components in IBADS to retrieve and update relevant building information. IHUB supports this functionality through a set of interfaces, IParseDrawing, IGetBuildingInfo and IPersistDesignInfo, which are implemented by nesting classes XParseDrawing, XGetBuildingInfo and XPersistDesignInfo respectively.

**IHUB as a Central Controller**

As described in chapter 3, the process control strategy is hard coded in IHUB to deal with the process management issue. The typical iteration through a typical design process consists of the following tasks that are carried out by the central controller:
• Create the building representation by parsing a design drawing or reading from a file;

• Get the external design forces from the user;

• If the building is a valid structure and no analysis has been done, create the locator to get the analysis agent and perform the analysis;

• If the building in a valid structure, analysis has been done but no code checking has been done, then create the locator to get the code-checking agent and perform the check.

• Save the current design to a file

As described in section 4.1, IAnalyze and ICodeCheck are implemented to help support the control strategy.

Implementation of IParseDrawing Interface

The IParseDrawing interface provides the functionality to parse the design drawing through method parseBuildingDrawing(). As described in section 4.2.2, AutoCAD supports a set of interfaces through which it exposes its functionality. As shown in the IDL description of IParseDrawing (Figure 4.10), parseBuildingDrawing() method

```plaintext
Interface IParseDrawing : IUnknown
{
    HRESULT parseBuildingDrawing([in]IUnknown* pUnkOfApp, [out]long* isStructure…);
}
```

Figure 4.10 IDL of interface IParseDrawing
takes an input argument. This is reference to the interface of object AutoCAD Application (the root object as shown in Figure 4.8). Other interfaces like interface of object BlockRef can be retrieved from this root interface. By manipulating these interfaces supported by AutoCAD, the AutoCAD object model of graphic entities can be translated into a building instance at the hub. Therefore, the parseBuildingDrawing() method serves as a translator between the hub and AutoCAD. When this method terminates, the IHUB expects that a central building representation is created based on the information in the AutoCAD drawing. If the drawing is not well connected then the drawing does not represent a valid structure. These type of error conditions will be detected and reported through the isStructure parameter.

Implementation of IPersistDesignInfo Interface

The IHUB component supports the persistent storage of the design information by using flat files. The set of operations for storing and retrieving design information from persistent storage in supported by the IPersistDesign interface show in Figure 4.11. The getPreviousDesign() method is used to retrieving a previous design from the persistent storage and the saveDesignInfoToFile() method is used to store current set of design information.

Figure 4.12 shows the format of the saved design File. The getPreviousDesign() method will read the specified file and create a set of objects that represent the current state of the design. Other components will be able to query this information through the methods provided in the IBuildingGetInfo interface.
Interface IPersistDesignInfo : IUnknown
{
    HRESULT saveDesignInfoToFile([in]long buildingAnalyze,
        [in]long buildingCodeCheck, ...);

    HRESULT getPreviousDesign([out]long* buildingAnalyze,
        [out]long* buildingCodeCheck, ...);
};

**Figure 4.11** IDL of interface IPersistDesignInfo

**Figure 4.12** Format of saved design file
Implementation of IGetDesignInfo Interface

The IHUB component maintains a common building representation and provides a set of interfaces that allow other components to access and modify this representation without knowing the details of the internal implementation of the data representation. The operations that can be used for querying the current state of a frame are grouped in the IGetDesignInfo interface. Information about the frame can be retrieved using methods like getTotalBeams, getTotalColumns (Figure 4.13). Methods getColumnArray() and getBeamArray() will return arrays of references to objects that represent individual beams and columns in the frame. Each of these objects is implemented as DCOM component and supports either the IStandardColumn or IStandardBeam (Figure 4.13) interface. Using these interfaces, the remote component can get information about the moment, stress ratio and current design of each beam/column.

Implementation of IAnalyze Interface

The IAnalyze interface groups methods that deal with the function of analyzing the structure. The IAnalyze interface defines two operations that are required for analysis of a frame. The method inputExternalForce() (Figure 4.14), allows an external component to add information about external load to the design description of the frame. Any previously stored load information will be overwritten.
Interface IGetBuildingInfo : IUnknown
{
    HRESULT getTotalNodes([out]long* totalNodes);
    HRESULT getTotalBeams([out]long* totalBeams);
    HRESULT getTotalColumns([out]long* totalColumns);
    HRESULT getTotalLoads([out]long* totalLoads);
    HRESULT getTotalSupports([out]long* totalSupports);

    HRESULT getColumnArray([out]IUnknown** columnsArray);
    HRESULT getBeamArray([out]IUnknown** beamsArray);
    HRESULT getNodeArray([in]long totalNodes,[out]IUnknown** nodesArray);
    HRESULT getLateralLoadArray([in]long totalStories,[out]IUnknown** lateral
    HRESULT getStoryDispArray([in]long totalStories,[out ]double* storyDispsArray);

    HRESULT setStoryDispArray([in]long totalStories,[in ]double* storyDispsArr
};

Interface IStandardBeam : IUnknown
{
    HRESULT getLeftMoment([out]double* leftMoment);
    HRESULT getRightMoment([out]double* rightMoment);
    HRESULT getStressRatio([out]double* stressRatio);
    HRESULT getVerticalUniformLoad([out]double* verticalUniformLoad);

    HRESULT getNodeFrom([out]long* nodeFrom);
    HRESULT getNodeTo([out]long* nodeTo);
    HRESULT getDescription([out]BSTR* description);
    HRESULT getMaterialType([out]BSTR* materialType);

    HRESULT setLeftMoment([in]double leftMoment);
    HRESULT setRightMoment([in]double rightMoment);
    HRESULT setStressRatio([in]double stressRatio);
};

Figure 4.13 IDL of interface IGetBuildingInfo of component StandardBuilding and
IDL of interface IStandardBeam of component StandardBeam

Interface IAnalyze : IUnknown
{
    HRESULT inputExternalForces([in]double* lateralLoads, [in]double* beamUniformLoad);
    HRESULT analyzeStandardBuilding([in]BSTR* columnDescriptions,
    [in]BSTR* beamDescriptions);
};

Figure 4.14 IDL of interface IAnalyze
The `analyzeStandardBuilding()` method will control the analysis of the frame using the current set of loads. This method is designed to forward the analysis request to an external component capable of performing the analysis task. As described in the previous section, the control policy rules such as:

"if building is valid and has not been analyzed, than locate the analysis tool and perform the analysis; otherwise, show the previous analysis result"

is hard-coded in `analyzeStandardBuilding()` method.

**Implementation of ICodeCheck**

The `codeCheckStandardBuilding()` method in ICodeCheck interface (Figure 4.15) codifies the following control rule:

"if building is valid, the analysis has been performed, and code check has not been done, then create the locator to locate the code checking tool, and perform the code checking"

The results of the code-checking, in terms of stress ratio of each column and beam, will be stored in the current design description.

```c
interface ICodeCheck : IUnknown {
    HRESULT codeCheckStandardBuilding();
};
```

**Figure 4.15** IDL of interface ICodeCheck
4.3.4 Locator

The locator has been developed specifically for use in IBADS and is the only component that is explicitly aware of other components in IBADS. The locator is responsible for identifying components in IBADS that are capable of specific tasks and forwarding that information to the IHUB. Since the locator is implemented using DCOM, the IHUB is not aware of the internal operations of the Locator and can only invoke the functionality that is exposed in the ILocator interface. Each agent joins the environment by recording its name, the functionality it supports, its location and the ID of the agent into a file. As shown in Figure 4.16, Tool A is located at machine “grad0”; it supports the “analyze” function and it has the ID of \{D9062290-FB9F-11d2-BC39-00104B8828E8\}. As described in Figure 4.17, based on the name of the functionality requested, the method locate() will parse the file to find the agents with the matching functionality and return reference to the interface of that agent. In the current implementation of locator, it chooses to randomly pick one agent out of the possible many available matching agents. Other possibilities for future implementations include selecting agents based on machine load or cost.

<table>
<thead>
<tr>
<th>Analyze ToolA D9062290-FB9F-11d2-BC39-00104B8828E8</th>
<th>grad0</th>
</tr>
</thead>
<tbody>
<tr>
<td>codecheck ToolB D9062290-FB9F-11d2-BC39-00104B8828E8</td>
<td>grad1</td>
</tr>
</tbody>
</table>

Figure 4.16 File recording the information regarding each agent
4.3.5 Wrapper Implementation

As an example of the wrapper implementation, IBADS includes multiple wrappers for ETABS COTS package. Included in this package are Etabs analysis software and Steeler code checking software. Several wrappers are developed to simulate the availability of several tools with similar capabilities in the design environment. These wrappers include:

1. Wrapper for Etabs supporting the IWrapper interface
2. Wrapper for Steeler supporting the IWrapper interface
3. Wrapper for Etabs supporting the IToolAnalyse interface that is an extension of the IWrapper interface
4. Wrapper for Steeler supporting IToolCodeCheck interface
5. Wrapper for ETABS supporting both the IToolAnalyse and IToolCodeCheck interfaces

All of these wrappers must deal with the task of translating the design information provided in IHUB into the input file for the appropriate component of ETABS and translating the output file into updates to IHUB. The translation process must deal with the difference in data representations between IHUB and ETABS. For example, the
translations must deal with the fact that ETABS defines beams, columns and frames using concepts such as bay, story, and column line. To simplify the process of translation between these concepts and the data representation in ETABS, the wrapper component creates a local representation of the frame. Figure 4.18 shows the classes that comprise that representation.

Figure 4.18 Class diagram of ETABS wrapper
The wrapper component for the entire ETABS package demonstrates the need for more specialized interfaces that serve as an extension of the IWrapper interface. Because this wrapper supports two separate functions it must define two separate interfaces. This is accomplished by extending the IWrapper interface with interfaces IToolAnalyze and IToolCodeCheck. In the current implementation, the new interfaces simply repackaging the three methods from the IWrapper interface into a single method, but this approach allows developing new interfaces supported by wrapper components that allow greater access to the internal operations of the tools that they control.
Chapter 5. IBADS: A Case Study

5.1 System Deployment

IBADS is a truly distributed system. Different components are running on different machines. In this case study, IBADS is deployed on the NT network in department of Civil Engineering at Rice University. ETABS is running on CADIL server; various wrappers of Etabs and Steeler are deployed on machine Grad0 in CAD domain; IHUB and the locator are running at CAD/Grad2. The GUI and AutoCAD can run on CAD/Grad3 or any machine outside CAD domain. (Figure 5.1)

![Diagram of IBADS deployment]

Figure 5.1 Deployment of IBADS
On grad2, a file needs to be created to store the registration information regarding each currently available agent in the system. The name, ID, functionality and location of each agent are recorded in the file. Five wrapper agents are built in IBADS, including,

1. Wrapper A for Etabs supporting the IWrapper interface
2. Wrapper B for Steeler supporting the IWrapper interface
3. Wrapper C for Etabs supporting the IToolAnalyse interface that is an extension of the IWrapper interface
4. Wrapper D for Steeler supporting IToolCodeCheck interface
5. Wrapper E for ETABS supporting both the IToolAnalyse and IToolCodeCheck interfaces

In this case study, wrapper A, C, D and E are used. Initially, wrapper A and D are registered with the locator, as shown in Figure 5.2. All the wrappers are deployed on machine grad0.

<table>
<thead>
<tr>
<th>Analyze</th>
<th>WrapperA D9062290-FB9F-11d2-BC39-00104B8828E8 grad0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeCheck</td>
<td>WrapperD D9062290-FB9F-11d2-BC39-00104B8828E8 grad0</td>
</tr>
</tbody>
</table>

**Figure 5.2** Wrapper A and D are registered in IBADS

On client machine, a text file where all the possible selections of beam and column's cross-sections are listed, need to be deployed.
5.2 Case Study

5.2.1 Background

As described in chapter 5, the client has the choice to start a new design or open a previous stored design. In this case study, the user will start the client by a new design of a two-span, two-story frame structure (Figure 5.3). When the design is done, it will be saved to the design store as an old design, and later it will be opened again.

![Figure 5.3 A two-span, two-story frame structure in AutoCAD.](Image)

5.2.2 Start A New Design

In client dialog, the user gives the building name as Building1, specifies the name of the drawing file, and opens it in AutoCAD. The drawing will be parsed by the IHUB
component in machine grad2 and a central building representation will be created. Beams, columns and nodes in the AutoCAD drawing will be numbered accordingly (Figure 5.4). Another dialog shows up (Figure 5.5). This is where the user picks the cross-sections of beams and columns gives external forces at each story and the uniform load at each beam. The dialog can be resized to fit the screen. As an experiment, column 5 is chosen intentionally to have a weak cross-section, W10X15. When the “Analyze” button is hit, the locator will be created and contacted at machine grad2. It searched the agent registration file where the information of all the current agents is recorded. Since the user is requesting the “Analysis” function, the locator will therefore find wrapper A and create wrapper A on machine grad0. Wrapper A will then start performing the analysis task, updating the central building representation at IHUB, with the analysis results in terms of moments and/or axial forces of each member. The client GUI will then retrieve the information from the central building object through the corresponding interface and update the dialog box in client machine accordingly. A short delay might be experienced since it is running across the network and Etabs analysis itself takes time.

After finishing the analysis, the user may choose to do the code checking. The same process goes on as described in the “Analysis” operation, only this time the locator will find wrapper D as the agent since it’s the one that supports “CodeCheck” functionality. In the client machine, a third dialog will show up, presenting the code-checking results based on the Steeler’s output, in terms of stress ratio of each member. Since column 5 has a weak cross-section, it’s stress ratio is greater than 1, suggesting a redesign (Figure 5.6). The design can then be saved by hitting the “SaveDesign” button.
Figure 5.4 Client start new design, drawing in AutoCAD is parsed and labeled

5.2.3 Open Previous Design

Since the user has saved the design of building1 in the design store as a file, that design can now be retrieved back directly from the file. First, in the client dialog, a drop down list will show the current available buildings in the design store. Choose building1 and open it. The user will be presented with a dialog with all the previous design and analysis result on building1. Besides, the time last design saved will also be shown. If the user tries to do code-checking now, instead of running wrapper D for Steeler on machine grad0, this time the client will query IHUB for the code checking result of last time,
Figure 5.5 Client dialog where user starts design and gets results

because nothing has changed since last design. This query therefore will be very fast since Steeler will not be run. The same stress ratio dialog will be popped up. Suppose the user changes the weak cross-section of column 5 from W10X15 to W10X45. After another round of analysis and code checking, the stress ratio of all the members will be less than 1, suggesting the current design satisfy UBC94 (Figure 5.7). The user can further experiment on different selections of beams and columns and pick the best design.
Figure 5.6 Stress ratio dialog, showing the results of the code-checking by Steeler

5.2.4 Dynamically Change the Tool-set

The user can experiment changing the tool-set dynamically while IBADS is still running. The user can open the agent registration file, remove the information regarding wrapper A and D, add information about two new agents, wrapper C and E, to support “Analyze” and “CodeCheck” respectively. This will de-register wrapper A and D and register wrapper C and E into the system. The user can then continue to work on the design and analysis, but this time with wrapper C and E. The tool-set has been changed dynamically during run time, however, IBADS still works fine, the user will not notice any difference.
In this chapter, IBADS is deployed and executed to carry out the design and analysis of an example two-story, two-span frame structure. A design iteration is went through where the user carries out a number of analysis and code checking based on different designs. The design information is saved to persistent storage and brought back again later. The user further experiments to dynamically remove old tools and add new tools to the tool set at run time, without affecting the system operation.

Figure 5.7 The dialog showing the results of the code-checking by Steeler
Chapter 6. Conclusion and Future Work

The purpose of this chapter is to summarize this study and to outline some possible extensions and improvements.

6.1 Conclusion

In this study, a number of requirements for agent management issues in building an integrated design software environment are analyzed and distributed object computing is identified as the technology that could partially support these requirements.

A new approach is proposed to building the integrated design environment. This approach is targeted at environments that use central-control strategy for process management and hub-and-spokes architecture for information management. A new way of wrapping an existing stand-alone tool into an agent is presented in the approach through the introduction of an extensible interface hierarchy based on a generic wrapper interface. To support the evolution of the tool-set for an integrated environment, a locator service is proposed. Finally, the Integrated Building Analysis and Design System (IBADS) is implemented to illustrate the benefit of this approach.

6.2 Limitations of This Work and Future Extensions

Although the new approach is well designed, several problems remained unresolved in its current implementation, which needs to be addressed as future topics.
The first concern is to build a better locator. In the current implementation of locator, it chooses to randomly pick one agent out of the possible many available matching agents. In the future implementation, the locator can be implemented with more intelligence to allow it to select agents based on machine load or cost.

The second problem is the lack of a register service. Currently, each agent has to be registered to the locator manually by writing its description into a text file. In the future, a separate register component may be proposed to deal with the registration of agents, therefore, agents will be able to register through the interface of register component automatically.

The third problem is the lack of consideration of design versions. Basically if the design change is made, the updated one will overwrite the previous design, the previous design is thus lost. With design versions, multiple designs can be kept for historical and reuse purpose.

The fourth problem is security. Currently, the security mechanism has been turned off completely, so that any client running anywhere in the world will be able to access the server. This obviously is not an ideal solution in the real world. Different access right should be assigned to different group of clients, e.g., architects, structural engineers etc. This will involves more consideration on both Windows NT security model and DCOM security model.
Finally, in the current implementation, data hub stored previous design in a flat file. In the future implementation, the information hub should be extended to utilize a more advanced storage mechanism, such as a database, for the persistent storage.
References:


