APPLICATION OF PROJECT INFORMATION MODELS TO COMPUTER-INTEGRATED CONSTRUCTION

by
KEVIN QIANG YU
B.E., Northern Jiaotong University, 1987

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF APPLIED SCIENCE

in
THE FACULTY OF GRADUATE STUDIES

( Department of Civil Engineering )

We accept this thesis as conforming to the required standard

University of British Columbia

October, 1995
© Kevin Q. Yu, 1995
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Civil Engineering
The University of British Columbia
Vancouver, Canada

Date Oct. 13, 1985
Abstract

Architecture, engineering and construction (AEC) projects are becoming increasingly complex. This complexity is manifested in the diversity and fragmentation of the construction projects. Computer-integrated Construction (CIC) offers ever-increasing potential to improve project management practice for AEC projects in resolving the fragmentation problem of construction information using computers (Froese 94d). CIC, however, must rely on the AEC project information being shared among various AEC computer applications used by many different project participants. This information sharing is based on the development of AEC information models (Froese 94b, 94c).

In this thesis, we have proposed a set of AEC information models, the AEC Core Reference Models (ACRM). The ACRM have been developed based on our investigation of currently available AEC information models and on our own opinions of the desirable characteristics of the models.

While the current development of the models is still in a conceptual stage, we believe that some basic central concepts are sufficiently well developed for testing some of their implementation capabilities in a computer program using a real-world example (Froese and Yu 94). In this research, we have developed a computer program, StartPlan, based on the ACRM. StartPlan provides a test-bed for these experiments. Since generic concepts such as Activity are defined in the ACRM, StartPlan can represent a wide range
of AEC project information and allow the information to be shared and translated between different types of AEC computer applications.

A fictitious construction project originally developed in REPCON (Russell 95) has been used in StartPlan for the test. Although it is not a specific real-world project, it can be treated as a typical project because it has been tested in many real-world situations and proven to contain most of the information that a similar real-world project would have. The results from the test reveal specific advantages and disadvantages of the proposed ACRM.
# TABLE OF CONTENTS

ABSTRACT .......................................................... ii

TABLE OF CONTENTS ........................................ iv

LIST OF TABLES ................................................. viii

LIST OF FIGURES ............................................ ix

ACKNOWLEDGMENT ............................................. xii

CHAPTER 1: INTRODUCTION .................................... 1

1.1 INTRODUCTION TO THE DOMAIN: RESEARCH MOTIVATION ............ 1

1.2 INTRODUCTION TO THE RESEARCH: RESEARCH OBJECTIVES .......... 3

1.3 INTRODUCTION TO THE RESEARCH METHODOLOGY: BASIC APPROACH .... 4

1.4 INTRODUCTION TO THE THESIS: READER'S GUIDE .................... 6

CHAPTER 2: LITERATURE RESEARCH ON COMPUTER-INTEGRATED CONSTRUCTION (CIC) ................................................................. 9

2.1 INTRODUCTION ........................................... 10

2.2 COMPUTER-INTEGRATED CONSTRUCTION ................................ 12

2.3 INFORMATION MODELS .................................... 14

2.3.1 Conceptual Models And Application Models ..................... 15

2.3.1.1 Conceptual Models And Modeling Languages ............. 16

2.3.1.2 Application Models .................................. 19

2.3.2 Layers of Information Abstraction .................................. 23

2.3.3 Implementation Techniques -- Object-oriented Techniques ........ 25

2.4 INFORMATION SHARING .................................... 27

2.4.1 Mechanisms of Information Sharing .............................. 27
LIST OF TABLES

TABLE 2-1: A SAMPLE TABLE IN A RELATIONAL DATABASE ........................................ 23

TABLE 3-1: C++ CLASSES IN DLL ............................................................................ 90
TABLE 3-2: BORLAND BIDS LIBRARY (BORLAND 93) .................................................. 107
TABLE 3-3: STARTPLAN EXPERT SYSTEM OBJECTS .................................................. 111

TABLE 4-1: DESCRIPTION OF THE EXAMPLE ............................................................ 117
TABLE 4-2: REPCON LOCATIONS OF THE EXAMPLE PROJECT ................................. 118
TABLE 4-3: REPCON ACTIVITIES OF THE EXAMPLE PROJECT ................................. 119
TABLE 4-4: CONTRACTORS IN THE EXAMPLE PROJECT ............................................ 120
TABLE 4-5: RESOURCES IN THE EXAMPLE PROJECT ............................................... 120
LIST OF FIGURES

FIGURE 1-1: RESEARCH METHODOLOGY ........................................... 5
FIGURE 2-1: INTEGRATION HIERARCHY ............................................ 11
FIGURE 2-2: COMPUTER INFORMATION MODELS .............................. 16
FIGURE 2-3: MODELING LANGUAGE ............................................... 18
FIGURE 2-4: AN EXAMPLE OF EXPRESS ........................................... 19
FIGURE 2-5: LAYERS OF INFORMATION MODELING ............................ 24
FIGURE 2-6: DIRECT MAPPING THROUGH TRANSLATORS .................... 29
FIGURE 2-7: DIRECT MAPPING THROUGH A GLOBAL TRANSLATOR .......... 29
FIGURE 2-8: SHARE CENTRAL DATABASE THROUGH TRANSLATOR(S) ........ 29
FIGURE 2-9: SHARE CENTRAL DATABASE WITHOUT TRANSLATORS ........ 30
FIGURE 2-10: MAPPING TO CENTRAL MODELS THROUGH TRANSLATORS .... 31
FIGURE 2-11: DIRECT CENTRAL MODELS ADOPTION ............................ 31
FIGURE 2-12: THE UNIFIED APPROACH MODEL .................................. 34
FIGURE 2-13: SOME HIGH-LEVEL OBJECT TYPES OF GENCOM ............ 35
FIGURE 2-14: CONCEPTUAL PROJECT MODEL, CORE OF THE BPM .......... 36
FIGURE 2-15: INHERITANCE RELATIONSHIPS IN IRMA ......................... 37
FIGURE 2-16: CLASS RELATIONSHIP IN IRMA .................................... 37
FIGURE 2-17: A PORTION OF THE ICON CONSTRUCTION PLANNING OBJECT MODEL .......................... 38
FIGURE 2-18: A PORTION OF THE ATLAS LARGE SCALE ENGINEERING PROJECT TYPE MODEL ............ 39
FIGURE 2-19: GRM FOR LIFE CYCLE FACILITY MANAGEMENT .................. 40
FIGURE 2-20: LAYERS OF STANDARD AEC REFERENCE MODELS ............ 41
FIGURE 2-21: THE AEC_CORE_MODEL ............................................ 43
FIGURE 2-22: THE RESOURCE_VIEW_MODEL ..................................... 44
FIGURE 2-23: THE PRODUCT_VIEW_MODEL ....................................... 44
FIGURE 2-24: THE PROCESS_VIEW_MODEL ....................................... 45
Acknowledgment

The author expresses his sincere appreciation to the following people who have greatly assisted in the completion of this thesis:

Prof. Thomas Froese, my major advisor, who has offered superb mentorship and a contagious zeal for scholarship with his rich knowledge in the field and patience in teaching.

Prof. Alan Russell, my secondary advisor, who has consistently provided thoughtful and invaluable guidance.

Many friends, colleagues and staff of the Department of Civil Engineering (UBC), who have helped make my work productive and my time rich.

I appreciate the effort and time of my family in Beijing, and of my friends in Vancouver, who have always made my life rich and varied.

Funding in support of this research was provided by the Canadian Natural Sciences and Engineering Research Council.

A special acknowledgment to Ying, ...

Thank you all!
Chapter 1

Introduction

Chapter Abstract:

This chapter introduces Computer-integrated Construction (CIC), and presents the importance of the development of AEC information models. It then describes the basic approach of testing the implementation capabilities of the models. The chapter then outlines the research objectives and provides a reading guide for the thesis.

1.1 INTRODUCTION TO THE DOMAIN: RESEARCH MOTIVATION

Architecture, engineering and construction (AEC) projects are becoming increasingly complex (Froese 94d). This complexity is manifested in the diversity and fragmentation of the construction projects that involve a large number of participants, making the sharing and exchange of project information increasingly difficult and time consuming. It has been reported that the highest priority in construction industry research is the integration of construction information (ICON 94).

It is Computer-integrated Construction (CIC) that holds the promises to resolve the fragmentation problem of construction information with the use of computers (Froese
The goals of CIC require that project information be shared and exchanged freely by many project participants using various computer applications. This information sharing is based on the development of AEC information models (Froese 94b, 94c).

Two basic types of information models are conceptual models and application models. The conceptual models (or core models) are those that define the basic concepts of the problem domain. They provide a high level support for the more detailed application models. Application models are those that define the data structures of a specific problem in a computer application. They directly support the development of computer applications. There are many types of application models such as aspect models, classification models and data models, and each type uses different methods to define the entity attributes of the models. More detailed discussion about the types of information models will be given later in Chapter 2.

Among these models, our research interest is in the development and implementation of AEC conceptual models. Although standardized AEC core models are still in their conceptual stage, we believe that they are sufficiently well developed to support the development of a computer program to test some of their central concepts and implementation capabilities using a realistic example (Froese and Yu 94). The results of the tests will reveal the advantages and disadvantages of the models, and thus will be useful and significant to the development of AEC information models.
1.2 INTRODUCTION TO THE RESEARCH: RESEARCH OBJECTIVES

The specific objectives of this research are as follows:

- To investigate a variety of AEC information models developed by other researchers, and to develop a set of AEC core models based on these models -- the AEC Core Reference Models (ACRM).

- To develop a computer application program, StartPlan, based on the ACRM and use it as a test-bed to test the basic central concepts and some of the implementation capabilities of the ACRM using a real-world construction project example.

- To implement the ACRM in an object-oriented computer language, in this case C++, so that computer tools based on it can be developed.

- To develop an object-oriented central AEC database based on the ACRM, so that many other AEC computer applications can use the central database as a core for project information sharing and exchange among many project participants.

- To develop a dynamic computer program library containing a set of object-oriented central or shared AEC models, so that they can be used by many AEC computer applications during run-time.

- To use a real-world example in StartPlan to represent a large amount of AEC project information and to translate it between different types of AEC computer applications.

- To use the results of the tests to help future developments of the AEC information
models.

The fulfillment of these objectives will eventually benefit the development of CIC, and more generally, the AEC industry as a whole.

1.3 INTRODUCTION TO THE RESEARCH METHODOLOGY: BASIC APPROACH

Figure 1-1 shows the basic methodology used in our research project. The entire research includes two phases, the Literature Research Phase and the Implementation Phase. The Literature Research phase contains three steps. The first step is to collect all the information about AEC information models currently developed by other researchers. These models are those that can be used as AEC conceptual models. This work is mainly based on currently published papers, international conferences and different types of international workshops in the research area. In the second step, the advantages and disadvantages of each version of the models are summarized and compared with each other. Based on Step 2, Step 3 involves proposing a single set of the AEC conceptual models, Model A, combining the best features of the each version.

Phase II is the Implementation phase. This phase involves 5 steps from Step 4 to Step 8. In Step 4, the results of our research on the characteristics of the models are summarized. Based on Model A obtained from Phase I and our perspectives on the models, a new version of the model, Model B, is created in Step 5. Up to this stage, the original version of Model B does not yet contain any feedback information. In
Step 6, we will treat Model B as a version of the AEC Core Reference Models (ACRM) and base the development of a computer application on it. After the computer program has been developed, a real-world example will be taken into the application in Step 7 to test the implementation capabilities of Model B. In Step 8, all the results of the tests in

Figure 1-1: Research Methodology
Step 6 and Step 7 will be used as the feedback information to support the recreation or refinement of Model B in Step 5. The new version of Model B will then be re-tested again and again in later steps. Along the way, we retain the right to change our own thoughts about the models and keep updating Model A according to the development of the models by other researchers. All the new information will be put together into the refinement of Model B. Consequently, some version of Model B will contribute (Step 9) to the development of the AEC information models and CIC as a whole. Step 9 also represents the immediate goal of our research project.

1.4 INTRODUCTION TO THE THESIS: READER’S GUIDE

This thesis is organized as follows:

- Chapter 2 provides background on Computer-integrated Construction (CIC) by introducing the basic concepts and current development state of CIC, and two key issues relevant to the development of CIC — Information Models and Information Sharing. The fundamental theme is that CIC relies on the sharing of project information among all the computer applications used by many project participants; and in turn, this information sharing is based on AEC information modeling. The chapter then discusses the development of current AEC information models by exploring and comparing their advantages and disadvantages. Based on this discussion, a version of AEC conceptual models, AEC Core Reference Models (ACRM), will be shown. ACRM will also be used as the base of the development of a
computer program -- StartPlan.

- Chapter 3 describes StartPlan by presenting its objectives, functionality, and basic components. StartPlan is a program that creates template construction schedules by using a built-in expert system according to a user’s description of a construction project. The program is developed based on ACRM as a test-bed to test the basic concepts and implementation capabilities of the ACRM models. Therefore, the chapter also discusses how the models are implemented in the program, such as using object-oriented techniques, ACRM supporting shared AEC models, ACRM supporting central AEC databases, and ACRM supporting file exchanging, etc. In addition, some of the important issues regarding system design and programming are addressed.

- Chapter 4 presents the real-world example used in StartPlan to test the implementation capabilities of ACRM and the workability of the program. The chapter first describes the nature of the sample project. It is a typical mid-rise residential building project and its plan was originally developed on REPCON (Russell 95) as a standard plan for research and education. The chapter then describes how the plan is translated from REPCON into StartPlan and how it is used in StartPlan for the testing. The results of the tests are summarized at the end of the chapter.

- Chapter 5 completes the thesis by discussing the conclusions and contributions of the research.

- Many of the figures in this thesis use EXPRESS-G notations. The EXPRESS-G is a modeling language, which is used to represent object information. It has three
basic elements -- boxed text, heavy linked lines and light linked lines. A boxed text means an entity, heavy linked lines represent sub-type relationships, while light linked lines are to show the associations described by their labels. Circles at one end of the links are to indicate the relationship directions. More discussion about modeling languages will be given in 2.3.1.1.
Chapter 2

Literature Research On Computer-Integrated Construction (CIC)

Chapter Abstract:

Computer-Integrated Construction (CIC) represents the next generation of using Information Technology (IT) in AEC industry. The major bottle-neck in current computing solutions and the key factor of CIC is sharing information among many computer applications. However, AEC information sharing must be based on AEC information modeling; and therefore, AEC conceptual models are the basis of all levels of AEC computer applications. We have been investigating such models and are interested in using some version of them as the core models to test some of their implementation capabilities in a computer program. This chapter first introduces the development of the CIC. And then, it discusses some of the basic concepts about the mechanisms of information modeling and information sharing. It will also highlight some of the current and important AEC conceptual-type models developed by other research efforts. Based on that, it will finally summarize a set of the AEC Core Reference Models (ACRM) that we deem are good to be used in the implementation of the computer program.
2.1 INTRODUCTION

Architecture, engineering and construction (AEC) projects are becoming increasingly complex. This complexity resulted originally from stricter requirements by the owners and the employment of new technologies, techniques, materials and tools. Also, contractors and suppliers are much more specialized and thereby becoming more competitive in their certain specialties. Thus, many projects are performed by larger organizations; the organizations are made up of loose associations of such small companies (Froese 94). General contractors are more like the role of an organizer of many small contractors, sub-contractors and their suppliers. This has led AEC projects to be increasingly fragmented, and the fragmentation may cause the project management to be much more difficult and inefficient (ICON 94).

To overcome this, all the fragmented units such as project participants, the pieces of their knowledge, and construction efforts must be integrated. In this context, the term integration is defined as the continuous and interdisciplinary sharing of information, knowledge, efforts, and goals among all project participants. There are two types of integration: Managerial Integration and Technical Integration (Figure 2-1) (Froese 94d). Managerial Integration focuses on the integration of personnel emphasizing reorganizing the roles of people, working groups and companies (e.g., the early assignment of construction managers, partnering, and design-build). Technical Integration stresses using workplace technologies to improve integration as a whole.

One may think that using computers in project management will resolve the
fragmentation problem automatically, but it is not always the case. Because different project participants use computer applications for different purposes, the computer information they create and use is often unlinked, duplicated and fragmentary, although they all relate to the same project.

It is Computer-Integrated Construction (CIC) that holds the promise of using computers integratively to perform the construction management tasks and to improve the overall quality of project management. CIC is a sub-field of Technical Integration centered on sharing AEC information among all computer applications used by all project participants. The information sharing is based on AEC information modeling, and both mechanisms belong to the family of a sub-field of computer science called Information Technology (IT).

This chapter first introduces the current development of CIC. It then discusses
some basic concepts about information modeling and information sharing. Afterwards, some particular AEC information models will be reviewed. Finally, a version of AEC core models, which is the one to be used for the implementation in a computer program, StartPlan, will be explained in detail.

2.2 COMPUTER-INTEGRATED CONSTRUCTION

People have been researching Computer-integrated Construction (CIC) since the 1980's (Björk 92). CIC is a term that describes an ideal approach in which many computer applications are used in an integrated environment in construction projects. Unlike the traditional approaches where AEC information is mainly represented and exchanged through paper drawings and reports, the use of computers in AEC industry has allowed the information to be represented in electronic digital forms using information Technologies. The pervasive availability of computers has led them to be the basic information “vehicle”. Infrastructures of network hardware and software are being established. Nevertheless, until recently, the integration of the computer applications has not been successfully developed because the major goal of most current applications focuses only on individual tasks. Consequently, similar to and perhaps partly because of the general fragmentation problem in the industry, fragmentation also exists among the computer applications. In other words, the AEC information cannot be shared by all the applications. CIC is intended to solve this problem since it promises to reduce the time and errors for information transfer, to speed the communication among project participants, and more importantly, to allow them to share one complete set of project
Integration in CIC can apply at different levels (Froese 94f), which are:

1. **The Application Level**: focuses on the integration of different applications in one system used by a small work group or single users sharing a single database. This is the base of the following two integration levels and requires the minimum effort to achieve. Initial implementations of integration must be done or tested first at this level;

2. **The System Level**: focuses on the integration of different systems within one big organization, e.g., different sections on different types of duties, or systems used by different companies working on one project. A central database is needed so that it can be shared among all the machines in the systems. Then each machine performs like a terminal allowing users of disciplines to view the central data for different purposes;

3. **The Industry Level**: focuses on the overall integration of all the computer applications across all companies and AEC projects. This is a long term target requiring the fulfillment of the integration at the other two levels.

Generally, a lower level supports the higher level and is easier to achieve.

There are many research topics within CIC. Some current topics are:

- AEC Information Modeling,
Among these topics, the research interest of this thesis is on the modeling of the AEC information since it is the basis of the AEC information sharing -- both are key factors of CIC. The rest of this chapter discusses the two mechanisms.

2.3 INFORMATION MODELS

People use computers to help them solve real-world problems more quickly and accurately. Therefore real-world objects must be represented (or modeled) in the computers. Such a process is called "Computer Information Modeling". Computer-integrated Construction (CIC) relies on information regarding AEC objects shared by many project participants using computer applications. In turn, this information sharing is based on AEC information modeling. Hence, well-defined AEC models are the cornerstone of AEC information sharing and CIC. This section describes two fundamental types of information models, Conceptual Models and Application models. It then introduces some mechanisms of information modeling. After that, the section discusses
some implementation techniques of the information modeling focusing on object-oriented techniques.

2.3.1 Conceptual Models And Application Models

The term "computer information" has been used loosely to mean the models in computers representing, describing and abstracting real-world objects. In other words, any real-world objects once represented in computers are modeled as computer information.

However, there are many different types of information models and the intended role of each can be vague. As a result, though they may not have been much misused, the differences between them have been somehow ignored when developing such models. For example, some high level models are the support for the lower level models and therefore are not intended to be directly instantiated (i.e. not to be implemented directly in software). However, people sometimes tend to mix up high level and low level models in one. While this may not be a problem during the initial development of the models, it can have negative effects eventually when the models are implemented. Also, the process of modeling is in fact that of an abstraction of the objects’ information. In order to actually use the computer information, for example, in information sharing, different levels of information abstraction must be addressed. Thus, it is necessary to make clear the types of the models, their relationships and the information abstraction layers on which the models lie.
Generally, computer information models fall into two categories: conceptual (or core models) models and application models; and the conceptual models support the application models (Figure 2-2).

**Figure 2-2: Computer Information Models**

### 2.3.1.1 Conceptual Models And Modeling Languages

The conceptual models (or core models) are intended to be high-level models that provide a unifying reference for more detailed application models that are constructed on top of them. Unlike application models, conceptual models are normally not intended to be directly instantiated for representing actual data about a real-world object. This means that a few attributes are given in any elements of the models and no objects can be derived directly from the models. The information at this level determines the data structures of the application models. It stipulates what the objects' TYPEs and what relationships among them are. The conceptual models provide formal definitions of the basic entities and relationships modeling real-world objects and their relationships about some discipline.
area, e.g., construction projects. These entities and relationships are needed to represent the object information in computers. Once the conceptual models are used to support application models, for example, storing data or sharing the data among applications, they in practice play the "core" roles for the others.

There are two types of conceptual models: reference models and type models. Reference models are more generic and focus on a whole branch of industry. They describe some topic that is relevant to many different areas of applications, such as Geometry or Drafting. An example of such reference model is the STEP General AEC Reference Model (ISO 92,93). Compared to the reference models, the type models are more specific and often describe the information structures of the basic objects within some domain for some specific problem, e.g. the Road Model Kernel for highways (Willems 90).

Conceptual models are described by modeling languages (Figure 2-3). The languages must be meaningful to the individual and "understandable" to computers. Since they must be written and read by people, they should either symbolically or linguistically make sense to humans. In most cases, English is involved as keywords. They may not be restricted to only certain types of objects. In other words, they must be generic enough to describe all types of objects. While to date some of them may be suitable for some problem domains and others may be good for other areas, ideally, they should be applicable industry-wide for all domains. International standard modeling languages may eventually emerge and be widely accepted. Again, the modeling languages are used for
analyzing real-world problems and designing computer systems to solve the problems. However, those problems are normally complex and complicated. Therefore, the languages themselves, as the problem solving tools, must be concisely organized so as to be easy to use. A modeling language normally has a syntax and semantics. The syntax prescribes the symbols that can be used in the language while the semantics define the symbols' meaning. Syntax and semantics are described in the specification of the modeling language.

![Diagram of Modeling Language](image)

**Figure 2-3: Modeling Language**

There are basically two types of modeling languages: data definition languages and graphical schema languages. The former is normally written in English and well organized by keywords. Therefore, they are understandable to people and excellent for computers to "read" and compile. They can also be an integrated part of a central database system. EXPRESS (Spiby 91) is a typical example. It is an alpha-numerical modeling language developed by the International Organization for Standardization - Standard for the Exchange of Product model data effort (ISO-STEP). Many research groups use EXPRESS to describe their models in their formal publications such as STEP. Figure 2-4 depicts an example that uses EXPRESS to describe an Activity entity. In contrast, the graphical schema languages provide a good overview of object relationships and so are
easy for people to use to do initial or conceptual analysis of the schema domain. They are excellent for research discussions too. Some examples are EXPRESS-G, NIAM (Nijssen and Halpin 89), and IDEF1X (IDEF1X 85). EXPRESS-G is the graphical representation of EXPRESS. It uses boxed text to mean an entity, heavy linked lines to represent subtype relationships and light linked lines to show the associations described by their labels. Circles at one end of the links are to indicate the relationship directions. Most of the figures that describe entities or objects and their relationships in this thesis use EXPRESS-G such as Figure 2-2 and Figure 2-3.

<table>
<thead>
<tr>
<th>Entity Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBTYPE OF (AEC_Object);</td>
</tr>
<tr>
<td>id: INTEGER;</td>
</tr>
<tr>
<td>name: STRING;</td>
</tr>
<tr>
<td>start_date: DATE;</td>
</tr>
<tr>
<td>finish_date: DATE;</td>
</tr>
<tr>
<td>result_from: SET(1:? OF Product;</td>
</tr>
<tr>
<td>result_in: SET(1:? OF Product;</td>
</tr>
<tr>
<td>uses: SET(1:? OF ResourceUse;</td>
</tr>
<tr>
<td>predecessors: SET(0:? OF Activity_Linker;</td>
</tr>
<tr>
<td>successors: SET(0:? OF Activity_Linker;</td>
</tr>
<tr>
<td>participant: SET(1:? OF Participant;</td>
</tr>
<tr>
<td>UNIQUE:</td>
</tr>
<tr>
<td>uq: id;</td>
</tr>
<tr>
<td>INVERSE:</td>
</tr>
<tr>
<td>predecessor_of: SET(0:1 OF Activity_Linker FOR;</td>
</tr>
<tr>
<td>successor_of: SET(0:1 OF Activity_Linker FOR;</td>
</tr>
<tr>
<td>END Entity</td>
</tr>
</tbody>
</table>

Figure 2-4: An Example of EXPRESS

2.3.1.2 Application Models

On the other hand, application models are more detailed information models than the core models (Figure 2-2). They are developed based on the core models. They can be instantiated and therefore directly used in computer applications. The physical data in
every computer database system is the instantiation of the application models. If the core models are considered to focus on modeling the relationships among the real-world objects, then application models are for modeling the properties of the objects. Therefore, they often provide full attributes for each entity. Unlike core models that define data structures at a high level, application models refine the data structures at a lower level. Values that correspond to the properties of the real-life objects can be directly filled in the data structures only if they have been refined in application models. Also, the application models can vary from one application to another even if they both are based on the same core models.

There are several different types of application models such as aspect models, classification models, and data models.

The aspect models are detailed because they provide all of the attribute definitions for each particular entity. These definitions are used for fully representing the actual computer information about particular real-world objects in the problem domain. The aspect models are the bases of computer programs. In fact, every computer application must use a set of aspect models for its specific program domain. The aspect models could be applied at run-time by the users. For instance, common database systems require the users to define the field names. They could also be put to use during the development stage of the applications, especially for most problem-specific ones. For example, the Activity attributes in the construction planing software REPCON (Russell 95) are defined by the developers during program development. The users of REPCON can only fill up
the attributes' fields pre-defined by the data models, but have no right to change the types of the attributes. The aspect models are written in computer languages. There are many examples available. For object-oriented analysis or design (OOA or OOD) of object-oriented applications, some languages are developed to write the aspect models, e.g., OOTher (Coad 94) for OOA. For application programming, most computer languages can be used such as C++ (C++ classes are good for describing object attributes).

Similar to the aspect models, the classification models are also detailed information abstraction, but use a different means for the detailed modeling. They adopt the basic concepts of the conceptual models and simply use the name of the entities for representing objects. Based on the entities, they develop extensive classification breakdowns of the domain by either specialization or aggregation classifying mechanism. Many subtypes or sub-subtypes of the original entities might be generated and they must be well organized. Although full and detailed attributes for each entity may not have to be provided in the classification models, they must be taken into account when classifying takes place because the differences between objects' attributes are the bases of the classification. The classification models are also directly useful for the implementation of the conceptual models in computer applications. Generally, the classification models are used to distinguish the differences between all low level objects so as to represent them in computers correctly and completely. Hence, the subtyping in the classification modeling differs from that in conceptual modeling as the conceptual models should break down an entity into subtypes only when they have different relationships with other entities. Again, conceptual models focus on entity relationships, while application models care about entity
attributes. Like the aspect models, the classification models can take effect at run-time when one is using a common database. They could also be reflected in a computer language during the application programming. Examples of the classification models within the AEC domain include the MASTERFORMAT classification and number system developed by the Construction Specifications Institute (CSI) and Construction Specifications Canada (CSC), the ISO Classification of Information in the Construction Industry Proposal (ISO 93) that prepossess several categorization dimensions, or the Integrated Building Process Model (Sanvido 90) that uses IDEF_0 models to itemize the steps involved in construction projects.

The data models are probably at the lowest level of information abstraction. They are a collection of information about a particular real world object. They are the actual data stored in a computer database. Some common data types of the individual items within the models are standard and widely used, e.g., integers, reals, strings or some picture formats like BMP or GIF, while others are application-dependent such as a record in a relational database or a class object in an object-oriented database. In this case, a group of values together represents one data model as the record or the object. Data models consist of the values entered by the users filling the data structures defined in aspect models and classification models. The data models are the direct computerized instances of the aspect or classification models with computer digital forms. For example, an object of C++ class "House" in a computer is the data model of a particular house in the real world, while the C++ class "House" is the aspect model or classification model of the computer object. (You may not be able to find a "House" entity in a conceptual
model; instead, there might be a “Building” entity that is treated as the supertype of the “House” in the lower levels.)

2.3.2 Layers of Information Abstraction

Figure 2-5 illustrates levels of information abstraction to demonstrate how a real world object, in this case “my house”, would be represented in a computer by different levels’ models.

At the first level, “my house” must be represented in a computer as a data model. All of the actual data about “my house” is entered in a computer database system. If the database system is a relational database, the data model of “my house” could be many tables to represent “my house”. Then the values for the attributes of “my house” will be under the corresponding fields. The shadowed record in Table 2-1 shows how the record for “my house” would appear on a relational database table. Again, the record (the set of field values) is the data models for “my house”.

Table 2-1: A Sample Table In A Relational Database

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Area</th>
<th>Height</th>
<th>Color</th>
<th>No. of door</th>
<th>No. of Windows</th>
<th>No. of Floors</th>
</tr>
</thead>
<tbody>
<tr>
<td>John’s house</td>
<td>1000 Main St</td>
<td>120 m²</td>
<td>12 m</td>
<td>green</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Little One</td>
<td>1234 King St</td>
<td>150 m²</td>
<td>15 m</td>
<td>dark brown</td>
<td>3</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The second level contains the aspect models and/or classification models. These models determine the data structures of the data models. For example, the field names in
Table 2-1, which represent the attributes of a house, should be defined in the aspect models. In this case, the user may have used some aspect or classification models at runtime to define the field names and data types. In an object-oriented database or any problem-specific database system, the developers may have used the models to define a C++ class "House", refining its attributes as Name, Address, and so on. In that case, the user would not have a chance to define something like the "field names", but only be able to fill a value for each attribute of "my house".

![Diagram of Layers of Information Modeling]

**Figure 2-5: Layers of Information Modeling**
The highest level is the conceptual models level. They support the development of the aspect models and classification models. The entity "House" may not be included in the conceptual models. Rather, an entity called "Building" is defined. This is because: 1) buildings are a common product type in the AEC domain, and a house is only a subtype of a building; 2) the basic relationships between the buildings and other major AEC objects like "Activity" are not different from those between a house and an activity. In other words, compared to a "Building", a "House" would not show any new relationships but only hold additional attributes; therefore, it need not to be counted in the conceptual models.

2.3.3 Implementation Techniques — Object-oriented Techniques

The development of the models is tools or techniques-independent. However, the implementation approaches for both conceptual models and application models are of interest, because one of our research objectives is to implement such models in a computer application and modeling techniques used affect the implementation tools for the application development. Many implementation techniques can be used in information modeling and some of them have been mentioned in the previous sections. Here, we emphasize object-oriented techniques because they have the following distinct advantages:

- Object-oriented techniques provide a natural methodology for modeling real world problems: information is represented as "objects" or "entities" that directly symbolized the objects in the real-world problem domain. Those computerized objects contain not only data describing the real-world objects but also methods
that manipulate the data. The collection of the objects makes up a model of the problem domain (e.g., a schema).

- **Object-oriented techniques recognize the existence of both similarities and differences among real-world objects**: they support inheritance relationships that match the same type of relationships among the real-world objects. For example, the class "house" is a subtype of the class "building". This means that a house must have something in common with a building, but it must also have something different (i.e., properties).

- **Object-oriented techniques are excellent at managing complexity**: each object can be manipulated as a whole ignoring the details of its contents. They provide the possibility of handling high-level modeling while still maintaining and manipulating the extensive details.

- **Object-oriented techniques are flexible and expressive**: they support data encapsulation. Some data can be hidden from others, while others can be opened to its "friend" or subtype objects. They also support data ownership. Changing one object has little impact to the rest in the system. They are expressive because they have no restrictions on what types of data they can hold. Indeed, they can contain almost all types of data such as text strings, numbers, sound, pictures and other objects.

- **Object-oriented techniques are efficient**: they support extensive code reusability. Also, they are applicable for different levels of data modeling, and thereby one
version of model can support different levels of data modeling. For example, a single object model can be used for problem analysis, system design, and application programming.

2.4 INFORMATION SHARING

Information sharing among all computer applications in AEC is the key factor to achieve the goals of CIC. The capabilities of the information sharing within the whole system can empower the applications. The fundamental functionality of the information sharing is to make the information used by one application usable by another. Information sharing is supported by information modeling (discussed in the previous section) and applies at different layers of model abstraction. This section first discusses the mechanisms of information sharing.

2.4.1 Mechanisms of Information Sharing

There are several information sharing mechanisms at different levels.

2.4.1.1 At The Application Models Level

The information sharing at this level is achieved using database files. Two mechanisms exist: one does not use a central database and the other does. Figure 2-6 shows that one application file can be directly translated into another type of file. For example, a Microsoft Project (MSProject 94) file can be saved as a Microsoft Excel (MSExcel 93) file. The drawback of this method is that one application would need to provide many translators for many types of files. An alternative is to use one global
translator as the core of all of the application files (Figure 2-7). While this approach is a little better than the first, both use direct data mapping mechanisms. This means that, in the translators, the data types for one application must be mapped to another. This is not always a perfect method because some data type in one application may not be able to be represented in another. The data may be lost or "skewed". The other mechanism is to use a central database. The central database is created from an application that directly implements core models of a problem domain. It therefore contains generic object data. This mechanism is better than the first two because no information should be lost through the central database that contains generic objects. Also, all of the data in the central database can be fully represented using many different applications, while the system can still take advantage of each application to represent certain types of data in different ways. For instance, if all of the data about the house is stored in the generic central database file, a CAD system can be used to represent its components graphically and an estimating package to represent all the pricing information. There are two methods for this mechanism: one uses translators and the other does not (Figure 2-8 and Figure 2-9). Using translators does not require the applications to understand the data structures of the data in the central database, while the data mapping problem still exists. The advantageous features of the second method are: first it does not have the mapping problem; and second it is faster because no translation is needed. However, it requires all applications to be developed based on the same information models.
Figure 2-6: Direct Mapping Through Translators

Figure 2-7: Direct Mapping Through A Global Translator

Figure 2-8: Share Central Database Through Translator(s)
2.4.1.2 At The Conceptual Models Level

Conceptual models also need to be shared so that information can be interpreted and exchanged uniformly between applications. Information sharing at this level uses the similar approaches to that at the application models. They are: 1) custom translation, which means that translators can be built between two different models used by two applications; 2) translation to a shared model (Figure 2-10), a shared model is needed and translators between each application and the shared model are required; 3) Direct adoption of shared model (Figure 2-11), applications directly adopt the shared model as their internal base. While each of the approaches has advantages and disadvantages, the first is rarely used at this level. For the later two, a shared model is needed in an integrated system. Thus, the definition and scope of the shared models are the key to the system as a whole. Selecting an appropriate scope of the shared models and defining the mappings between them to the individual application’s models is not an easy task because of the large range of applications needing the shared models. Some people have suggested layered shared models instead of a single level models to suit different degrees of requirement.
2.4.2 Standards

The shared models are the core to an integrated system (Froese 94a). It is logical that standardizing the shared models can strengthen the integration capabilities of an integrated system. Also, since a larger range of applications need to be in the integrated system, it would be difficult to have all the applications' developers working
together on developing one set of shared models. Hence, standard models as the shared models become necessary. In fact, standard models are believed to be one of the most important factors of making progress on developing powerful integrated systems and achieving the goals of CIC.

Many efforts to develop such standards are ongoing. STEP (mentioned in Section 2.3.1) is one of the largest and most formal among them. STEP (ISO 92, 93) generally provides industry wide product data models that are normally rich and well structured. In addition, the STEP models are application independent to support all types of applications in their domains. STEP is also currently the most active and influential effort in AEC information standards area.

2.5 AEC CORE REFERENCE MODELS

Computer-integrated Construction (CIC) relies on the computer systems that are capable of representing a large variety of AEC information and allowing the information to be shared among all the computer applications used by many project participants. More particularly, this capability depends on some central or standard models that can be shared by all the applications. It is believed that these kinds of models should be developed as conceptual models. This section describes the development of conceptual models for AEC information and summarizes a set of models -- the AEC Core Reference Models (ACRM), that we think provide useful AEC conceptual models.
2.5.1 Development of Construction Project Information Models

AEC project models fall into two categories: product models and construction process models. Although ideally a super set of AEC core models should embody the two, we currently focus only on the latter. We have been investigating high-level construction process models to support the eventual emergence of standard core AEC models (Froese 95a). The development of such process models or related ones have been ongoing for many years. This section will give a brief introduction to several major ones.

**Unified Approach Model**

The *Unified Approach Model* (Figure 2-12) was developed to support a single conceptual modeling technique for representing all types of construction information in order to achieve the integration of different applications using different information technologies, such as CAD, project management, EDI procurement and so on (Björk 92). An implementation that used the model to represent the information used for the erection of partition walls was also carried out (Karstila, Björk, and Hannus 91). One of the objectives of the model was to demonstrate the relationships between existing building classification systems and product models. The model also revealed the differences between application models such as activity models that are usually formalized by IDEF\(_0\) (IDEF\(_0\) 81), and conceptual models.
GenCOM

The General Construction Object Model, GenCOM (Figure 2-13), was developed as part of Froese’s Ph.D. project from 1989 to 1992 at Stanford University (Froese and Paulson 92, Froese 92a, b). It aims to improve the integration of project management software using standard object-oriented models of construction projects. This research investigated the role of standard models for implementing integrated software packages, for creating data exchange languages, and for providing industry-wide information categorization schemes. Details of the general objects’ attributes, constraints, and implementation characteristics were provided in the model. The implementation of the model was carried out in an integrated project planning application -- the Object-model-based Project Information System, OPIS (Froese and Paulson 94).
GenCOM centers on the project’s physical components and the activities that operate on them. In particular, a GenCOM activity represents the application of some action to some component using a construction method with a set of resources. A project participant responsible for the activity is represented in GenCOM as an actor. The actions, methods, and resource-uses are all explicitly defined at a lower level.

**Building Project Model**

The *Building Project Model (BPM)* (Figure 2-14) was developed by Luiten as part of his Ph.D. thesis on Computer Aided Design For Construction (Luiten 94). It is based on the General AEC Reference Model (GARM) (Gielingh 88), and the IMPPACT Reference Model (Gielingh and Suhm 92). BPM comprises a core that relates product, activity, resource, and actor information. It focuses on the relationships between products and activities. It also identifies the progression states of an object, e.g., initial requirements, proposed stage, and final realized object.
IRMA

*Information Reference Model for Architecture, Engineering, and Construction (IRMA)* (Figure 2-15 and Figure 2-16) was initially developed at the *International Workshop on Models For Computer-Integrated Construction in Finland*, October 1992 (IRMA 93). In the workshop, a group of researchers, including the developers of the above three models, shared their experiences and research results. By comparing the three models, they had found that the models had a lot of similarities and that similar methodologies had been used for developing the models. They thought that it was necessary to produce a single model that embodies the good ideas of all three. IRMA was the result. It is a combination of best features of each model. It centralizes the relationships among some important concepts of AEC such as products, activities, resources and participants and treats them as generic objects. In IRMA, the activity is the core around which all the entity relationships are established. IRMA is a conceptual reference model and stands at a high-level without giving any detailed information inside.
any entities. It can be used as a framework for standardization in AEC, and as a core AEC model for implementation (Froese 94c, Froese and Yu 94). Discussions about IRMA were continued later in November 1993 in an electronic workshop (Froese 93a). The revised version will be described in great detail in the next section.

Figure 2-15: Inheritance Relationships In IRMA

Figure 2-16: Class Relationship In IRMA

ICON

The Information/Integration for Construction (ICON) (ICON 94) proposes to investigate the possibility of establishing a framework for integrating information systems
in the construction area. It applies some techniques of information technology (IT) such as the Information Engineering Method, object-oriented analysis and design, and computer aided software/systems engineering (CASE). Figure 2-17 depicts a portion of ICON’s Construction Planning Object Model. Other process-models-related models in ICON are a tendering object model, a procurement systems model, and a number of specific construction activities’ models.

![Construction Planning Object Model](image)

**Figure 2-17: A Portion of The ICON Construction Planning Object Model**

**ATLAS Large Scale Engineering Project Type Model**

The *ATLAS* project is aimed at the development, demonstration, evaluation and dissemination of architectures, methodologies and tools for Computer Integrated Large
Scale Engineering (LSE) (Tolman, Bakkeren, and Böhms 94). The ATLAS LSE Project type Model (PtM) is an abstract model of LSE project information. It is based on the STEP resource models and can be implemented in different engineering specialization areas in AEC such as buildings or process plants. It supports data exchange between the different sectors. It is also intended to be used by different project participants for different views. A portion of ATLAS LSE PtM is shown in Figure 2-18.

![Diagram of ATLAS LSE PtM](image)

Figure 2-18: A Portion of The ATLAS Large Scale Engineering Project Type Model

**Generic Reference Model For Life Cycle Facility Management**

The *Generic Reference Model (GRM)* (Reschke and Teijgler 94) for life cycle facility management bases project information models on three research projects:
EPISTLE (processBase, PISTEP, and SPI-NL), the IDEF0 modeling standards, and the modeling techniques of Shell ICT (West 93). It is a generic conceptual AEC model and can support many AEC applications and AEC project life cycle phases. It is also intended to contribute to the research on the applicability of Application Protocols within STEP. A portion of the model is shown in Figure 2-19.

![GRM For Life Cycle Facility Management](image)

**Figure 2-19: GRM For Life Cycle Facility Management**

**STEP**

*Standard for The Exchange of Product Model Data (STEP)* (NIPDE) is the largest and most important product modeling standardization project and is documented in by *International Standards Organization (ISO) draft standard 10303*. Numbers
of world-wide research efforts are developing STEP to provide generic approaches for industry-wide product data exchanging among computer systems. The AEC building and construction group in STEP has started working on the conceptual core AEC models for project process information; and a draft building construction core model (ISO 1994) is now available.

2.5.2 AEC Core Reference Models

In November 1993, an electronic workshop called IRMA-tica '93 was held mainly by the original contributors to IRMA (Froese 93a). It aimed at continuing the development of IRMA by gathering more ideas on the models from a wider range of researchers in the AEC object modeling field. Many issues relating to the models where raised during the workshop and a revised version of the model was reported by Froese (Froese 94c). This section gives a detailed description of the new model because it is these models that were implemented as the AEC Core Reference Models (ACRM) in the computer application, StartPlan.

Figure 2-20: Layers of Standard AEC Reference Models
Layering was used as the basic approach to adapt different perspectives and objectives from different people in one single standard core model. A proposed AEC object layering is shown in Figure 2-20 (Tolman in Froese 93a). The AEC_Core_Model entity contains all of the AEC related classes that have been generally accepted in different sections in AEC industry. All of the view models at the second level are refinements of the AEC_Core_Model, but they focus on different types of AEC applications. For example, the Product_View_Model focuses on products, the Process_View_Model on construction process, the Resource_View_Model on resources, and the Control_View_Model on project controls. Each of the view models may still have some of the classes defined in another view model but this is essential only if the included classes affect those in the view model. Other possible views may exist at this level. The models at the upper two levels are conceptual models. They will support application models and information exchange; and the actual application models will be implemented at lower levels. For example, the Scheduling_View_Model refines the Process_View_Model and sits at the third level.

**AEC Core Model**

Possible contents of the *AEC_Core_Model* are shown in Figure 2-21. The modeling at this level is aimed at identifying the major entities of AEC. The *AEC_Object* is an abstract class (i.e., no direct object can be instantiated from it) that is also the root class for all AEC related objects. While there might be some more subclasses, the major ones are defined as follows:
- **Component**: any part of an AEC product;
- **Activity**: a task that produces a component;
- **Resource**: anything made use of in AEC projects;
- **Control**: anything that influences an Activity;
- **Participant**: one or more person(s) who play some role in AEC projects.

![Diagram of AEC_Core_Model](image)

**Figure 2-21: The AEC_Core_Model**

No detailed attributes and relationships of these classes are given at this level. Because they depend on the particular views, they will be defined at lower view levels.

The Project_Object is another abstract class that represents any object that belongs to a particular project. However, not all AEC_Objects are necessarily Project_Objects. They are Project_Objects only when used or performed in the project. For example, a general contracting company is not necessarily an object of a project if they do not have any projects at hand. They may become a Project_Object once they have gotten the contract for doing a project. The Project_Object class does not have to be an AEC_Object because all its subclasses will be AEC_Objects through multiple inheritance.
Resource View Model And Product View Model

The Resource View Model and the Product View Model are illustrated in Figure 2-22 and Figure 2-23 respectively. They are highly simplified because neither is the focus of our current research. The reason why they are shown here is because the classes have some relationships with that in the Process View Model and some others.

![Diagram of Resource View Model](image)

**Figure 2-22: The Resource View Model**

![Diagram of Product View Model](image)

**Figure 2-23: The Product View Model**

The Process View Model (Figure 2-24) is a revised version of IRMA. It is for representing AEC processes such as construction tasks with all their attributes. It also represents all of the AEC_Objects that are related to the processes. The methodology is to use the Activity as the basic representation unit for an AEC process. All of high-level objects are linked to the Activity. The related classes are described as follows:
- **Resource**: any AEC_Object Resource that is used for the Activity;
- **Participant**: any AEC_Object Participant that plays a role in the Activity;
- **Control**: any AEC_Object Control that influences the performance of the Activity;
- **Result**: anything that results from the performance of the Activity (e.g., an AEC_Object Component);
- **Method**: any technique or methodology that is used to perform the Activity.

More subtypes would be defined at lower-levels if needed to reflect the more detailed relationships to the Activity. Especially, the relationships shown in this model are consolidating relationship objects. That is, more detailed information about the relationships themselves can be defined. For example, an activity uses resources. However, information such as the amount of each resource needed for the activity should exist in between the Activity and the Resource rather than in either of the two. Then a class **ResourceUse** would be defined. As a result, the relationship of the Activity uses the Resource is “objectified” as an object of **ResourceUse**. Other places that could have this kind of objectification would be the type of role that a Participant plays in the Activity, the

![Figure 2-24: The Process_View_Model](image)
type of influence that a Control has towards the Activity, and so on. The objectification may be presented explicitly in this level's models, or in lower level's models.

In addition, multiple inheritance technique may be needed to represent more complex relationships between the classes. For example, a crew can be a Participant of the Activity but should also be considered to be a resource for the Activity. Presumably, a class like \textit{HumanResource} is defined as a subtype of both the Participant and Resource classes to represent the crew's special role in the Activity.

\textbf{Scheduling View Model}

The \textit{Scheduling View Model} (Figure 2-25) is a refinement of the \textit{Process View Model}. It represents typical construction scheduling activity information including the precedence relationships (predecessors and successors). It also depicts detailed information about resource-activity assignments for resource leveling. It is a good example of high-level layer models supporting lower-level models that can be directly implemented in a computer system, in this case construction planning system. The following explains its main classes and their relationships:

\begin{figure}[h]
\centering
\includegraphics[width=0.6\textwidth]{scheduling_view_model.png}
\caption{The Scheduling View Model}
\end{figure}
- **Plan**: a subclass of the Activity in the Process View Model. Because the Activity class is generic and allows aggregation relationships, the Plan can be defined as a collection of many individual activities.

- **Schedule Activity**: a subclass of the Activity in the Process View Model. It differs from its supertype because it has attributes specially for describing a scheduling activity such as Early Start Date. Each of the Scheduling Activities is a subpart of the Plan.

- **Resource and Resource Use**: the Resource has no differences with the one in the Process View Model, while the Resource Use is an objectification of the Activity-uses-Resource relationship in the model. This is a more detailed description of the more general relationship between the Activity and Resource; and it must be defined here since this model will support direct implementations.

- **Precedence Relation**: is an objectification of the Control-influences-Activity relation in the Process View Model. It is a sequence constraint between two activities and indicates one activity is the controlling object of the other. Thus the Schedule Activity is also a subtype of the Control class. Multiple inheritance must be used in this case.

- **The Result, Participant and Method**: are not presented in this view, because they are not typical enough for activity scheduling to be included in all basic scheduling programs. However, they can be easily applied in any application models (normally lower-level models) if required. Indeed, certain scheduling applications
do treat the activity participants or components as important factors for scheduling, and this can be critical for some types of projects such as a high-rise building that has repetitive activities.

2.6 CONCLUSION

Two vital mechanisms to achieve CIC, the information modeling and the information sharing, were discussed in this chapter. Moreover, some current AEC object information models developed by other researchers, particularly the conceptual models or the like, were outlined. Also, a good version of the AEC conceptual reference models (ACRM) that can be the core models supporting AEC computer applications was presented. As I indicated in the “Introduction” chapter, one of our research objectives is to test some of the basic ideas and the implementation capabilities of the current conceptual AEC models in a computer program. The ACRM are good and typical enough to be the tester because of its contents and modeling mechanisms. The next chapter will explore the world of the development of the computer program, StartPlan, and will explain how the ACRM are implemented in it.
Chapter 3

STARTPLAN

Chapter Abstract:

The AEC Core Reference Models (ACRM) have been discussed in the previous chapter. As mentioned in the Introduction of thesis, one of our research goals is to develop a program to test some of the basic ideas and capabilities of the models. StartPlan is such a program that has been developed based on the ACRM models. In this chapter, the StartPlan program is described in detail including its basic components and functionality. In addition, issues about how the models are implemented and what aspects of the implementation are tested in StartPlan are discussed. Moreover, some important facts about StartPlan system design and programming are reviewed.

3.1 INTRODUCTION

The need for the AEC Core Reference Models (ACRM) in Computer-integrated Construction (CIC) has gained recognition among the researchers and practitioners in the AEC industry. For the last decade, a great deal of research effort has been put into the development of such models in the form of research projects and international workshops (Björk 92, Froese 93a and 94c, Froese and Yu 94). Our research goal is to investigate the
role of these standard AEC data models in CIC. However, our experience has indicated that it is difficult to achieve this until the ACRM concepts are truly implemented in real-world projects in a computer application. We wish to evaluate the models as a general information reference by applying them to specific AEC problems, and use some version of the models to represent and manipulate real world construction information (Froese and Yu 94).

Although the existing models are still in a conceptual stage, we believe that some of the central concepts are sufficiently well developed for testing some of their implementation capabilities. Our intention is not to give final solutions to project-specified problems, but to obtain feedback information from the experience of coding an application based on the models and using it to deal with a real-world project. We hope that our practice will assist in the future development of ACRM and in a broader sense the overall development of CIC (this issue will be addressed again in the Section 3.2 later).

StartPlan (Froese and Yu 94) is such a program that attempts to deal with construction project management by creating template construction schedules. Based on a user's description of a construction project, StartPlan will create a construction schedule from a library of "good" schedule segments for similar projects and will alter it to suit the details for the working project at hand. The system will export the schedule to one of a variety of formats to be used as a "starting template" for creating a new project schedule in systems such as Microsoft Project. Not only does this save users significant effort in creating a new schedule, but it helps them take advantage of the scheduling expertise of
Also, StartPlan provides a useful test-bed for testing the implementation capabilities of the ACRM. The test will be done in three ways: 1) representing real-world construction information and integrating various type of construction management information by representing and translating it between different project management applications; 2) implementing the models using an object-oriented computer language, in this case C++, so that computer tools based on them can be developed; 3) examining the possibility that an object-oriented central AEC database based on the models can be developed. With an expert system built into the program, it will create document templates that provide users with various frameworks to start off working on their own schedules.

StartPlan is developed in C++ (Borland 4.0) and runs on a PC platform running Microsoft Windows.

This chapter describes the detailed aspects of the development of StartPlan and the implementation of the models in it.

3.2 SYSTEM OBJECTIVES

The ACRM models are standard structured conceptual models that reflect the information about real-world AEC objects and their relationships. Since researchers have not yet reached an agreement for a satisfactory perfect version of ACRM, its practical
applicability and implementation capabilities have not been proven. However, it is not necessary that experimental work must wait until a final product has been produced. The experience of applying the models to real world examples will be helpful in their development. We believe that applying some real-world projects in an ACRM-based system can expose weakness of the current versions of the ACRM. StartPlan is such a system that provides a test-bed on which a real world project data can be produced, manipulated, transferred and reported. Thus, we propose that the major objective of the StartPlan system is to test the implementation capability of the ACRM models.

In a broad sense, we believe that StartPlan will also serve as a good test-bed for data modeling research, because: 1) it must represent construction information at a detailed level and in a variety of forms; 2) the construction information is prototypical of real-world generic information in terms of scope, level of detail, and quantity; 3) it must represent a wide range of project information in a generic sense rather than only one type or one piece of specific information for one specific purpose; and 4) it must be able to translate information in from and out to a variety of existing applications. We consider not only typical scheduling software like Microsoft Project and Primavera, but also systems like REPCON that have a fundamentally different definition of a construction activity and applications like common databases or spreadsheets that have a different representation of data. In this regard, we see this application as a good practice arena for general data modeling research.
3.3 **SYSTEM FUNCTIONALITY**

StartPlan’s main function is to generate template plans. There are two types of functionality of the system. One is for normal users who need templates in order to create their own construction schedules. The other is for power users who need to modify the knowledge base documents and rule base documents for the StartPlan Expert System.

### 3.3.1 Functionality For Normal Users

The normal users are those who use the system as a tool to create construction schedules. The functionality for the normal users involves a series of user-machine interactions and a set of system procedures. The following list presents typical steps that would be involved in creating a new template schedule in StartPlan. This example presents the functionality for the normal users.

1. The user selects an option to ask questions about information on the project that you want to plan for.
3. StartPlan makes a template schedule and stores it together with the facts about the project into the Project Document.
4. StartPlan opens one of the various viewers to display the information in the Project Document.
5. The user alters the plan schedule if necessary.
6. The use selects a desired output format.
7. The translator engine does the translation.

3.3.2 Functionality For Power Users

StartPlan has the feature to allow the knowledge base documents and rule base documents to be modified whenever needed and during run-time. Because the two documents are the basis of the expert system and the project documents for the schedule templates, it must be noted that only the power users who are extremely experienced and exactly clear about what they are doing can change these documents. Any problems introduced in these files may cause undefined errors when running the system, or incorrect results when using the templates in a project.

The basic functionality for working with the rule base document is no more than modifying the rules including adding a new rule, changing or deleting an existing rule, and so on. For the reference document, StartPlan can modify general project information such as resources or products. Also, and more typically, schedule segments can be modified. Activity relationships such as precedence relationships can also be altered.

Moreover, since the reference library and rule base documents are not developed as a part of the coding but are separately stored in persistent storage using an ASCII (American Standard Code for Information Interchange) file format, it is possible to modify them outside of StartPlan by any generic ASCII editors such as DOS Text Editor. More discussion on this issue is provided in Section 3.6.
3.4 WHY USE TEMPLATES AND AN EXPERT SYSTEM

There are two reasons for using the templates and an expert system in StartPlan. One is to simplify the system for the user. The other is to allow for the system to implement the ACRM.

For the former, we intend to allow users to easily perform their tasks in StartPlan by taking advantage of the document templates and the expert system.

First, it saves a great deal of the user's time to create a complete project schedule for a relatively large project, e.g., a 15 floor condominium. For example, to create a similar activity list using conventional means, e.g., typing in the activities individually from a scratch, may take more than one hour. However, the StartPlan expert system can produce 300 activities in a 10 minute session. By executing the expert system, the system will automatically generate a template schedule from which the user has already been given the majority of the activities and their links required for the final usable schedule and other project information such as resources and products.

Second, from our experiences, we have realized that many people do not use scheduling software to its full potential. Possible reasons are: 1) they do not have the experience to design a fully detailed schedule that contains all the information needed; 2) the effort required to develop such a schedule might be too great; 3) the software they use is not capable of allowing users to create such a schedule because it is not able to store, represent or report information to the desired detailed level. One solution would be to
provide schedule examples or templates that the users can start with and work on to modify the specifics of their own projects. In fact, a good schedule should have various appropriate representation formats along with the detailed descriptions about the scheduling information. Such schedules can be easily obtained from the templates having few chances of losing important information that good schedules must have. Once the templates are prepared by the system, the users can start to modify data to suit the project details at hand by imitating the sample data descriptions and formats in the templates.

However, there are enough details of good schedules that are dependent upon project specifics that it would be difficult to develop either a small number of templates that can cover many applications, or enough typical templates to suit many types of projects. The expert system in StartPlan is intended to overcome this problem. In the expert system, good schedule segments are stored in a reference document. Based on the information about the project provided by the user, the expert system will create a new template schedule by applying the rules in a rule document to the reference. The new template schedule will be suitable for the type of the working project.

Third, using the expert system allows the user to take advantage of the others' expertise. In the reference and rule base of the expert system, human experts' knowledge is captured and organized in an appropriate way. Normally, this knowledge is considered to be perishable, scarce, vague and difficult to apply. However, through the expert system, one can easily, quickly and directly adopt this knowledge in their own practice. This can be of great help in terms of ensuring the quality of the schedules and of the
overall project control and planning.

From the stand point of achieving the objectives of StartPlan, using the templates and expert system enhances the testing ability of StartPlan to implement the ACRM models in the system. First, the templates are represented as the model-based project documents. This means that the data structures of the information in the templates are based on the definition of the models. Since the templates are supposed to give the user a good starting point for a good construction schedule, they should contain sufficient and detailed enough information about the project. Hence, they are a good test for the capability of the models to represent real world project information correctly and completely. Second, the reference document of the expert system is also based on the ACRM models. As indicated, the reference consists of good construction schedule segments from which the templates must be built. For this reason, the data stored in the reference document must be well structured and organized. Likewise, the experts' knowledge, that is again normally difficult to apply, must be well organized in order to be transferred into the reference library, and able to be represented as computer data. The data must not only maintain the original knowledge but also be refined into a form that is compact, clear and easy to apply. Thus, for the reference library is to be based on the ACRM models, it is another good test for the models to represent complete and well-organized real-world information.

To summarize, both the templates and the expert system in StartPlan are to help the users easily get started to create their own schedules. They are also helpful in the
implementation of the models to achieve the objectives of StartPlan.

### 3.5 SYSTEM COMPONENTS

As shown in Figure 3-1, StartPlan consists of a Project Document, a series of Viewers and Translators, and an Expert System. This section will first address the concepts relating to the Project Document. Then, the system viewers and translators will be described in section 3.5.2 and 3.5.3 respectively. The next section (Section 3.6) will give a full detailed description of the expert system, including the reference library document and rule base document.

![Figure 3-1: StartPlan System Components](image)
3.5.1 Project Document

The project document in StartPlan is the document that the users work on so as to make construction plans for the construction project at hand. The template schedules that StartPlan creates are contained in the project documents within which the templates can be modified. After the modification, the document files can be saved or translated out to other project management application files.

Ideally, the project document will contain full information about multiple projects and a number of different types of plans such as resource plans and activity schedule plans. Each type of plan has its focus. For example, a resource plan may focus on how to manage resources among many activities and many projects, while an activity plan normally deals with activities' starting and finishing times, etc. However, in the current StartPlan, there is only one project and a list of scheduling plans with each containing a list of activities. Concepts relating to the plan lists are one of the parts that have been developed the most in StartPlan. They have the capability to represent quite a large amount of information about the plans and activities. The project in the project document, on the other hand, holds information about the working project such as project name, location, and so on. The information can be obtained by the users through the interfaces of the expert system; or from the users manually typing in via the viewers of the project document. In addition to these facts about the project, the project document also contains a resource list, a participant list, and a product list. The instances in each list, for example, a resource, will be linked to the activities in each plan list.
The data in the project document is based on the ACRM models. It is one of the most important components because most important implementations of the ACRM entities and their relationships can be found in the document. Some examples are: Activity, Resource, Component, Result and so on. The design and the use of the project document plays a key role in the implementation of the ACRM in StartPlan. Refer to Section 3.7 for more detailed discussion about the implementation.

3.5.2 Document Viewers

The viewers are one of the fundamental components of StartPlan. They are made to display large amount of and various types of AEC information in ACRM-based documents. Thus, they are also crucial in the implementation of the models.

StartPlan document viewers are the bridge between users and documents. Through them, data in the documents is displayed to and manipulated by the users. These viewers provide a large variety of user interfaces for certain purposes, such as tabular activity breakdowns, Gantt charts, network diagrams for scheduling or resource list for estimating. While the document-based databases will be implemented independent of any specific viewers, the document viewers are largely modular components of the system and normally document type-dependent. This means that each type of the viewers is designed and can only work for certain type of the documents. For example, a viewer for a project document cannot be used for a rule base document. When the system is invoked, a series of viewers will be automatically associated with a type of document. While one viewer has only one document, there may be many viewers for one document. Once the user
modifies the document through one viewer, the system will notify the changes to all of the others.

While the document type-dependent viewers may well be the major ones in StartPlan, other types exist. These viewers are not specifically for some type of document but for some type of data. For example, a picture viewer can be used to display a picture assigned to an activity in the project document. It can also display an equipment picture in the reference document. More importantly, in the future, we will develop a generic type of viewers. Those viewers will not be document type-dependent, but should be able to present all the objects in all types of the documents in StartPlan. They may also have a generic object browser to search for all run-time objects in the system.

In spite of the above, this section will focus only on the StartPlan document type viewers, which fall into three groups: those for the project documents, those for the reference base documents and those for the rule base documents.

3.5.2.1 Viewers For Project Document

The viewers for project documents display all the information in the project documents and help the user modify the information. There are three major viewers in the current version and each focuses on a certain type of the information.

One view is called The Project View (Figure 3-2). It summarizes all the information in the project document. Basic project information such as project names can be found in this view. In addition, the resource list, product list, and participant list are
displayed. A plan summary is also shown as a plan list. If one plan in the list box is highlighted, a list of its activities is presented in the activity list box. The Project View does not show any detailed information about individual objects, e.g., attributes of an activity. Basic objects such as resources or plans can be added or deleted through this view, but no detailed modification is allowed, e.g., setting predecessors. This view is good for the user to get a quick glimpse of the contents of the project document and a general idea about it.

Another view is the Activity List (Figure 3-3). This view lists all the activities for a particular plan in rows. Each row indicates one activity with some of its attributes. Though this view is simple, it is quick to pop up and requires little memory. It is also easy to design and code. As a result, it is useful during the system developing phase. For
example, when the project document data structures are changed, it is always used to test the system and get quick results. Through this view, activities in the plan can be modified.

![Activity List Viewer](image)

**Figure 3-3: Activity List Viewer**

The Activity Information View (Figure 3-4) emphasizes all the detailed information about a particular activity. One activity must be highlighted from the activity list so that all its attributes can be shown in certain fields of the display window. Activity precedence relationships are represented in two lists, one for predecessors and the other for successors. Resource assignment, activity participants and activity products can be modified and the links are shown in the view. A number of user-interface dialogs are also designed for these modifications. The Activity View plays an important role in the implementation of ACRM, because it is capable of presenting a large amount of information about activities, which are at the center of the ACRM models.
Other than those three major viewers, there are also some minor viewers for the project document. These viewers are used to display additional information and each emphasizes a different type of information. For example, the Contractor List Viewer (Figure 3-5) displays all the information about the contractors for the working project, while the Resource List Viewer (Figure 3-6) and the Product List Viewer (Figure 3-7) show the information regarding all the resources and products for the project respectively. Objects in these viewers such as contractors, resources or products can be added, deleted or modified through their corresponding viewer. For instance, a new contractor can be added to the contractor list through the Contractor List Viewer.
Figure 3-5: Working Project Contractor List Viewer

Figure 3-6: Working Project Resource List Viewer

Figure 3-7: Working Project Product List Viewer
3.5.2.2 Viewers For Reference Document

There is currently one viewer in StartPlan to support the reference document (see Section 3.6.2.1 for detailed description of reference documents) (Figure 3-8). Similar to the project documents, a reference document contains information about a project and a list of plans. Therefore, the viewers are able to display the project information including the resource list, product list and participant list (see columns 1, 2 and 3 in the view window). The plans are displayed as a list in column 4. The list of each plan’s activities is located on the side. Through the viewer, both the project and plans can be configured such as adding, modifying and deleting a product or a plan. This viewer is mainly for the users to get an idea about what is in the reference library document. The importance of this viewer is the same as the project viewer; however, there are not many viewers needed.
at present. As for the detailed information about each object, e.g., an activity, once it is
translated into a template, the viewers for the project document can be used to display and
manipulate the detailed data.

3.5.2.3 Viewers For Rule Base Document

The viewers for the Rule Base Document (see Section 3.6.2.2 for detailed
description of rule base documents) can be rather simple. At present, it is not critical to
have many viewers for the document. As long as they can present all the information in
the document, one window of the view is enough. In StartPlan, there is one viewer for the
rule base document shown in Figure 3-9. The rules in the document are listed according
to the questions (see column 1, “Question”). All the possible answers for each question
are listed in column 2, “is”. Then for each answer, the conclusions are displayed on
conclusion fields in the “THEN” group box. The “Segment #” is the number of the
schedule segment in the reference library document according to the answer in the “is”
column. The “Repeat time” is the time that the segment should be modified by, and it is
normally one for most cases. If the repeat time will be defined by the user, for example
the number of the floors, the “Repeat time” field shows “User-defined”. In this case, the
inference engine will get the correct segment and accumulate it by the number given by the
user. Through this viewer, the users can also modify the document by adding, modifying
or deleting the rules.
3.5.3 Translators

The Translator Module is another basic component of the system (Figure 3-10). It serves to integrate different kinds of existing applications upon the basis of ACRM concepts. The information sharing is achieved by translating project information between applications such as Microsoft Project and StartPlan.
In the current StartPlan system, the document files are all in ASCII format. Therefore, the translators must be made to work based on plain text files. There are several advantages of using ASCII file format, especially at the starting stage of the system development. First, in the future development of StartPlan, standard data files will be made available to help achieve the goals of the system, for example, as a central AEC database. It is likely that international standard data exchange file formats are in ASCII type. Second, it is easy to read and trace the information in the input and output files. This makes the coding of the translators much quicker and easier because the contents of the translated files are visible and straightforward to be read and modified. The text editors are widely available on every machine. Thirdly, ASCII type is one of the generally accepted standard computer file formats. It is commonly used or available for many computer applications. Many programs have the capability to save and/or read ASCII files such as Microsoft Project and Microsoft Excel. Finally, StartPlan is currently using the ASCII type as its basic database file format (the Project Document, Reference Document and Rule Base Document files are ASCII format). Therefore, using ASCII format for the translating files makes the coding consistent.

Despite the above, we will still make all the database module and translation module available for binary file format in the future to take advantage of its efficient data storing feature.

There are two translators in the current StartPlan, one for Microsoft Project and the other for Microsoft Excel. Each comes with an output and an input engine. Taking
MS Project as the example, StartPlan input translator translates MS Project files into StartPlan project document files. First, in MS Project, a table must be created. It defines the data structure of the information in the MS Project file that StartPlan can understand. Then, the MS Project project file must be saved based on the table as ASCII type. After this, the user can start StartPlan and choose the “Run Translator” command and the “Input MS Project” option from a menu in the main window. A common file open-save dialog will then be opened. From the dialog, the user can browse and select the file saved in MS Project. Next, a StartPlan translator will be “fired” and the data in the file will be translated into ACRM-based objects. StartPlan will then create a Project Document object along with a viewer for it (normally the Activity List View). Finally, all the translated information will be stored in the object and be made visible to the user through the viewer. It is up to the user whether to alter it, translate it into another format or simply save it as a StartPlan file. The output translator works the similar way but in a reverse order. A table must also be defined in the MS Project before it attempts to open the translated file.

The translators for MS Excel work roughly the same way as those for MS Project, except that no tables must be defined because Excel columns can basically take any type of data automatically. Nonetheless, the user still must be clear about the data file structures that the Excel Input Engine can read, and must use the structures to organize data in the Excel spreadsheet. Otherwise, the Excel file will not be able to be understood by the translator. ASCII file format must also be required for the translation.
3.6 STARTPLAN EXPERT SYSTEM (SPES)

The StartPlan Expert System (SPES) consists of a User Interface, an Inference Engine, a Reference Library Document and a Rule Base Document. In this section, basic concepts about a common expert system are first explained. After that, all aspects of the SPES are described in detail.

3.6.1 Basic Concepts of An Expert System

Expert Systems, also known as Knowledge Based Systems, are a branch of artificial intelligence (AI), a sub-field of computer science (Hu 89). Other areas in AI are Robotics, Speech and Image understanding and so on. Simply, the idea behind expert systems is to capture the expertise of human experts and to represent it in computers, so that others cannot only take advantage of the expertise to solve problems, but to provide easier, quicker and better computer applications.

An expert system is typically arranged into four sections as follows:

1. *The Reference Library*, containing the facts in a specialized problem domain. It normally contains all the application-specific information. Application-specific means the problem domain that the expert system deals with. This information can be simple facts (e.g., a person's name), relationships (e.g., employer and employee), or procedural information (e.g., sequential code for printing a report or drawing a graph). The reference library is the major component of an expert system and lies at the heart of it. In most cases,
even after the development of the system is completed, its reference library must be regularly modified to reflect knowledge revision in the problem domain.

2. The Rule Base, containing the rules to be examined by the inference engine to retrieve the facts in the reference library. It often embodies the expert's expertise in the rules. The rules are conditional statements with the occurrence(s) of fact(s) as a premise or condition and with the assertion of another fact as conclusion in the form of: IF (condition) Fact A, Fact B,... and Fact C, THEN (conclusion) Fact D and Fact E.

3. The Inference Engine, containing the computer instructions to reason with the information provided by the user and stored in the reference library. It then uses a logic rule in the rule base to solve the problems. An example of an inference rule commonly used in an expert systems is: IF A is TRUE, and IF A THEN B is TRUE, then B is TRUE. Possible implications of the logic rule exist. For example, IF B is NOT TRUE, and IF A THEN B is TRUE, THEN A is NOT TRUE. The inference engine executes by searching the rules and facts in the rule base and the reference library. Backward chaining and forward chaining are the basic mechanisms of the searching.

4. A group of user interfaces, supporting the connection of the inference engine and the user, external data bases, or the programs running on other computers. The rules are normally represented as questions by the user
interfaces. The users will, via the interfaces, answer the questions and view or report the results of the concluded facts after the inference engine's execution.

### 3.6.2 StartPlan Expert System (SPES)

The StartPlan Expert System (SPES) is an important part of StartPlan. It captures experts' expertise about construction control and planning. Through SPES, the knowledge can be retrieved, manipulated and translated so as to be used by the users. It is also the vehicle to get the template schedules in StartPlan.

Following the convention of normal expert systems, SPES consists of a reference library, a rule base, an inference engine and a set of user interfaces. The reference library document is developed based on the ACRM models, while the rule base document is ACRM-aware that understands the models supporting the reference document. Thus, while the data structures of the documents are defined in the program, the contents of them are not encoded in the expert system. SPES will read the documents at run-time to allow maximum configurability. On the other hand, the inference engine and the user interfaces are a part of the coding.

#### 3.6.2.1 Reference Library Document

The reference library documents are databases of good practice that store the library of good schedule segments. In particular, each schedule segment is represented as a plan that contains a list of activities, for example, all activities for a typical floor.
A list of such plans are stored in the reference library document. At run-time, when the expert system is executed, a series of plans in the document are selected by the inference engine according to the rule base. Those plans will then be combined into one complete plan for the project and saved in the project document. The reference library document also contains information about a project, which includes a resource list, a participant list, and a product list. These objects are linked to certain activities in the document. They will also be retrieved by the inference engine and the links are retained in the project document.

The experts' knowledge and experience contained in the reference library documents is not always constant during the life time of projects. Since new ideas and solutions to the problems in the domain will be represented in the contents of the reference library, it is important that the documents must be able to be updated over time and the new versions must be able to be executed during run-time. Therefore, SPES does not encode the reference library in the program, but rather loads it from a separate reference document file at run-time. Thus the intelligence of the system can be re-configured and revised.

While a knowledge database document is project-independent since it is not restricted to any specific project, it is project type-dependent. This means that one reference library document will only suit one type of project. For example, there could be a document that contains scheduling information about a typical reinforced concrete high-rise building, and another dealing with highway construction. Different versions of the
documents can exist in the computer system as different data files. The user can choose one of these as the knowledge document at run-time to suit the project at hand. This feature provides great flexibility to deal with different types of projects for different types of users.

StartPlan allows reference library documents to be modified within the system. Nonetheless, because files are of ASCII type, it is also possible to modify them in any text editor externally such as DOS text editor or Windows Notepad.

### 3.6.2.2 Rule Base Document

A rule base document contains rules that the inference engine uses to reason. The rules are needed to retrieve appropriate data in the knowledge documents. They are problem-dependent so they only work for a specific reference library document.

There are many similarities between a rule base document and a reference library document. First, it captures experts' expertise; and for the same reasons, it must be able to be updated over time. Second, a rule base is not encoded in the program so that it will be read at run-time. StartPlan also allows the rule base documents to be modified at any time to suit the needs of the developing construction projects. Third, its file format is of ASCII type. Therefore, it can be flexibly edited by any generic text editor outside StartPlan. Although the rule base document in StartPlan is not ACRM-based (the rules are not created based on the models), it is ACRM-aware -- that is -- it understands the models that support the reference library document. The rules in the rule base must work
with the contents in the reference document.

3.6.3 Inference Engine

An Inference Engine is a software “machine” that contains a set of processing methods to query users, reason with rules and retrieve knowledge. The reference library document, rule base document and answers to the questions are the parameters of the inference engine. During user interactions, the questions created according to the rules are asked and a set of answers are retrieved. In the case of StartPlan’s current reference library document and rule base document, the inference engine applies the answers to the rules, and uses the results to identify a set of plans along with the number of repetitions of each plan. Then it gets all the plans from the reference library and accumulates each plan by the repeating time (e.g., a building has 10 typical floors). After this, it combines all the plans into one complete one and puts it in the project document. The inference engine will also retrieve the project information from the reference library and the user, and set it in the project document. It is responsible for assigning resources, products and participants to the activities.

3.6.4 User Interface

The viewers for the reference library documents and rule base documents have been described in Section 3.5.2.2 and 3.5.2.3. This section only describes the ones designed specially for retrieving questions from the rule bases and getting the answers from the users.
Since the questions are from the rules, the user interface should be able to communicate with the rule base documents. Several window type dialogs are designed together with the interface (Figure 3-11 and Figure 3-12). They are able to provide enough fields to display questions and catch and display answers. Some function menus and/or buttons to send or receive messages are also needed. Through the dialogs, the users are able to provide enough information about the working project by answering those questions.

---

**Figure 3-11: SPES User Interface**

**Figure 3-12: SPES User Interface Questioning Window**
The user interface in SPES is rule-independent. This means the interface windows do not have to be restricted to certain types of questions or answers. Any questions in the rules can be viewed and asked in one of the dialogs. It can also make all the possible answers available in the dialogs in accordance with the type of the question. Thus, the user does not have to type in the answer but should just pick up one from the answer list. This not only saves the user much time but also avoids misspelling or invalid answer strings. This is useful since the interface will suit any type of project and will not need to be upgraded when rule base documents change.

3.7 IMPLEMENTATION OF THE MODELS IN STARTPLAN

The implementation of the ACRM models in StartPlan is one of the major objectives of our research project. Computer applications are written in computer programming languages. With StartPlan, we intend not only to use the models as the base of its database documents, but also to experiment with how the models can be used as the base of computer programming for a model-based computer application. Also, we would like to examine the possibilities of using the models to develop a central AEC database that can be dynamically shared among many computer applications, and to support file exchange in CIC. We believe that this practice will be a great help for future development of ACRM.
3.7.1 Implementing ACRM In An Object-Oriented Computer Programming Language (C++)

Object-oriented analysis, design, and programming are different forms of computerized data modeling, but all with the same purpose of building computer data models about the problem domain at a low modeling level. ACRM models are top level models about AEC objects and they support the development of data modeling at the lower levels. Normally, during the high level modeling phases, the internal details of the entities are not of the interest. However, because high level models are needed to support the lower level modeling, they must be meaningful to the lower level models in matching the concepts of real world objects. Thus, it is unrealistic to invent good high level models without the experience gained from lower level modeling such as programming computer applications and using the applications in real-world examples. The appropriateness of the ACRM models cannot be fully tested until they are truly implemented in the computer programming practice.

This section discusses the issues relating to the implementation of the ACRM models in a computer language, in this case C++.

3.7.1.1 Why Use Object-Oriented Programming Technique

The features of object-oriented techniques have been discussed in Chapter 2. They are all applicable for object-oriented modeling, analysis, design and programming. In particular, object-oriented programming centers around several major concepts: abstract data types, classes, type hierarchies by subclassing, inheritance and polymorphism. Object-
oriented techniques have therefore been used in the design and programming of StartPlan. In addition, the development of the models employs object-oriented techniques. Hence, it is logical to use the same techniques to develop a computer application based on the models. In this way, the designing and programming are linked more closely to the implementation of the models. As a result, the programmers can easily handle a large number of entities in the models and complex relationships among them at the programming phase. For example, the advantages of inheritance is one of the reasons for using object-oriented technique for the modeling. An object-oriented computer language can also support this type of relationship by easily making classes as subtypes or supertypes of the others. This is extremely important since the models are changing frequently and the design and programming of the application must follow the models' development steps by being updated over time. Our experiences have clearly indicated this fact.

3.7.1.2 Why Use C++

C++ was originally developed at AT&T Bell Laboratories in the early 1980s (Eckel 89). It is a hybrid language that focuses on object-oriented functionality with the features of a structured language, C. C++ offers programmers object-oriented capability without losing run-time efficiency. Specifically, the object-oriented language features of data abstraction, encapsulation, inheritance and polymorphism are fully supported in C++. Besides, C++ classes and their objects are highly compatible with entities in the ACRM models. This is attractive to us when choosing a language as the implementation tool.
For a broad consideration of the objectives of StartPlan, we seek to consider any issues that could arise when implementing the ACRM models in an object-oriented computer language. C++ has the most features of object-oriented techniques and is one of the most popular object-oriented languages used among programmers. It goes further than other object-oriented languages such as Ada and Modula-2 in terms of supporting the object-oriented features and is faster than SmallTalk.

Hence, C++ is sufficiently representative of existing object-oriented languages and is powerful enough to use as the basic programming language for StartPlan.

We have chosen Borland C++ version 4.0 as the compiler along with the Borland ObjectWindows Library (OWL) version 2.0 (Borland 93). The reasons for the choice are: 1) it supports all C++ features; 2) it is currently regarded as one of the best C++ compilers; 3) OWL supports Multiple Document Interface (MDI), which suits our needs for handling different types of documents and viewers; 4) OWL supports a wide variety of graphical user interfaces such as windows, listboxes, buttons and menus. This provides a high degree of potential possibilities for building an application that requires rich viewers; 5) Borland International Data Structures (BIDS), the container libraries, make it easy to handle both common data structures and special user-defined classes.

3.7.1.3 How C++ Classes Work With The Models

The implementation of the models as C++ classes cannot be done for the higher-level models, e.g., at the Standard AEC Reference Models level (Figure 3-13). A C++ class of "Process_View_Model" is not very meaningful. Within the view model,
however, implementation can begin.

![Diagram of AEC Core Model and its refinements]

Figure 3-13: Layers of Standard AEC Reference Models

Looking at the contexts of the models, we identify two fundamental elements: entities and relationships. For the implementation of the models, the entities can be directly interpreted as C++ classes. The entities in the high level conceptual models normally do not contain any attributes. The classes, however, will contain detailed attributes as their instances are to be created. For example, the "Activity" entity in the AEC_Core_Model can be directly implemented as a C++ class named "Activity" (Figure 3-14 and Figure 3-15). The boundary of the class "Activity" can be the same or very similar to the entity "Activity". The C++ class implementation of the model can also be at a lower level. For example, the class "Product" is a direct interpretation of the entity "Component"; in addition, subtypes of the "Product" are created, e.g., the class "Floor" (Figure 3-21 and Figure 3-22).
In the StartPlan implementation, most of the entities in the AEC_Core_Models and other view models have been implemented as C++ classes, which include the AEC_Core_Model (Figure 3-14), Process_View_Model and Scheduling_View_Model (Figure 3-16 and Figure 3-17), Resource_View_Model (Figure 3-19), and Product_View_Model (Figure 3-21), while some less important ones are ignored at this time. The class implementations are shown underneath each model accordingly (Figure 3-15, Figure 3-18, Figure 3-20, and Figure 3-22). Since StartPlan is a scheduling application and our current research interest is mainly in the process modeling rather than product modeling, our focus of the implementation is therefore on the Process_View_Model and Scheduling_View_Model. Also, since the AEC_Core_Model is the base of the entire schema, it is also our focus. Thus, the implementation of the other view models has remained at a level as high and simple as possible.
Figure 3-16: The Process_View_Model

Figure 3-17: The Scheduling_View_Model

Figure 3-18: C++ Classes of Scheduling_View Model And Process_View_Model
Figure 3-19: The Resource_View_Model

Figure 3-20: C++ Class Implementation of The Resource_View_Model

Figure 3-21: The_Product_View_Model

Figure 3-22: C++ Class Implementation of The Product_View_Model
The second basic element of the models is the relationships between the entities. They fall into three basic categories: Aggregation type (a 'Part_of' relationship), Specialization type (a 'Type_of' relationship) and Association Relationships (any types other than these two).

A “Part_of” relationship occurs when one object is contained in another object. A “Part_of” relationship between two entities can be implemented by defining: the class of one entity to have an attribute that is a list of objects of the other class type derived from the other entity. For example, a Plan has a list of Activities (Figure 3-18). Borland C++ container library classes are often involved in representing the Part_of Relationship (a more detailed explanation about this will be given later in this chapter).

The “Type_of” relationships in the models, when implemented in C++, are often treated using the subtyping feature of C++. For instance, in Figure 3-14, entity “Resource” is a “type_of” “AEC_Object”. In Figure 3-15, the C++ class “Resource” can be easily created as the subclass of the class “Object”. In this implementation, class “Resource” and “Object” are the C++ class interpretation of the entities “Resource” and “AEC_Object” respectively. A subclass has the same attributes as its superclass plus more that the superclass does not have. Both object-oriented modeling and programming language like C++ support this inheritance. Therefore the implementation works well by matching the entity relationships in the models with their lower level implementation forms in C++.  

86
Other relationships can be quite arbitrary such as Participant "plays_a_role" in Activity, Activity "uses" Resource, etc. These types of relationships between entities are normally represented in C++ as follows: two classes with one having the attributes that are objects or pointers to objects of the other type. For example, the class Activity has a "participant" attribute, which is a pointer to a class "Participant" type object.

Pointers in C++ are variables that hold the physical memory addresses of the objects of data types or functions. Pointers are useful in many situations. The most common one is to use a pointer to tell a procedure call where the object that it points to lives in the system. If a pointer to an object is used as an attribute of a C++ class, it represents the fact that the object of the class has a relation with the object as the attribute. This keeps a link by providing a reference in the computer memory between the two objects in the system. The advantage of this utilization of pointers is that it can represent most types of relationships since the linkages in the computer memory can be understood by programmers as any relational links that are meaningful.

The use of pointers in C++ offers great flexibility in representing complex relationships between objects within the entire system. Proper use of C++ pointers provides strong capabilities for representing many different types of object relationships. It also avoids duplicated objects from existing in memory. For instance, once the system has created a list of Participant objects, the objects are stored in the memory. Then when one of them has been assigned to an Activity, which means that this Participant "plays_a_role" in this Activity, the system does not make another copy of the Participant. Instead, it lets
the Activity know the memory address of (pointer to) that Participant object. The Activity then can send messages to the Participant by calling the address. The reason for doing this is that it does not make sense to make a copy of a same person in the sense of modeling the real world practice. One person could be assigned to many jobs (activities), but he is always one person. Also, the time to make a copy of an object and the space to hold it in the computer are both considered unnecessary. Thus, high system efficiency is to be expected. This would be critical in guaranteeing that the system will run satisfactorily while maintaining the capability of handling a large amount of data.

3.7.2 Support For Shared AEC Models

We would like to examine the possibility that central or shared AEC Models based on the ACRM can be developed and used. The Central AEC Models provide the definitions of many basic and key concepts about AEC domain such as Activity, Resource, Participant, Control, Product, and so on. These concepts are defined in the entities with proper contexts in the ACRM models. It is possible that these models are used as Central AEC models, or some Central AEC models can be developed based on the ACRM models. Either way, the central models will be shared or adopted by AEC applications. This will be a great help to CIC as a whole.

Conceptual models are described in computer modeling languages, and the descriptions are saved in computer files. Although models can be shared by letting the files be accessed by multiple applications, this is not an easy task. One reason is that conceptual models do not provide detailed information about the attributes of the entities.
Thus, actual objects of the entities are not intended to be created this way. Also, the objects of the entities cannot be directly used in the applications so that translators must be provided either generically for all types of applications or specifically for an individual one. Moreover, the descriptions are written in modeling languages such as EXPRESS or EXPRESS-G, so they are not coded as part of the application program. Hence, the implementation of the models have to be made dynamically at run time. This technique is called late binding, which is always tougher in terms of programming than early binding where the models are implemented as lower level languages.

One solution is to use dynamic-link libraries (DLL) (Borland 93). A DLL is an executable library module. It is obtained by a C++ compiler compiling C++ class files. The instances of the classes in the module thus can be created and their methods can be called at run-time. When a DLL is loaded by the operating system, it can be shared among multiple applications; one loaded copy of the DLL is all that is necessary.

As discussed above, the ACRM models are implemented in the form of C++ classes in which data structures are defined according to the contexts of the entities in the models. In order to be made in a central model and shared by many applications at run-times, the classes must be compiled into a DLL. We have created such a DLL and the following C++ classes shown in Table 3.1 implemented from the ACRM models have been built in it.
Several aspects of using the DLL can be noted. First, the classes in DLL can be used by the applications either directly or through translators. The DLL never needs to be re-compiled. The applications or translators that use the DLL, however, must know the import records of the DLL (the DLL contains the corresponding export records). The records are contained in a LIB file which should be included when compiling the applications or translators. Second, the DLL should normally be located in the same directory as the application executables. However, if there are many applications running and using the same DLL, it should then be located in a common directory, such as c:\windows; but no more than one DLL is needed because it can be shared by all the applications. Consequently, the ACRM models are shared by the applications by simply being implemented as C++ classes and built in the DLL.

Other than offering the possibility of sharing a single model in many applications, using DLLs has other advantages. First, since the DLL and the applications are developed (compiled) separately, it is not necessary to rebuild the DLL when updating the application, or even to rebuild the application when updating the DLL. Also, it can reduce...
the size of the applications executables and reduce the time needed to compile the applications. Moreover, it can conserve system memory because only one loaded copy of it is needed.

Weaknesses arising from use of DLL exist. For example, using DLLs requires an early binding mechanism, which means the applications decide how to implement the models (classes) at compile time because they understand only the C++ classes. While early binding has the advantage of execution speed, it lacks the benefits of flexibility and a high level problem abstraction that late binding can offer.

Nevertheless, from our practice of developing and using the DLL in StartPlan, we have found that using DLLs is a good approach to achieve sharing ACRM models. Comparing to directly sharing the ACRM models at conceptual level, which requires late binding, it is quick and easy to apply, and requires less effort.

3.7.3 Support For A Central AEC database

One way to achieve the goals of CIC is to allow many AEC computer applications to share one central AEC database. We would like to examine the possibility that such a central AEC database based on the ACRM can be developed.

The central AEC databases store AEC computer objects in computer memory. Unlike sharing the models, sharing database is an implementation at the Information Models level of data abstraction because the objects in the database are in fact the information models. It is objects in the databases that play the key role in the
computer applications. Therefore, the implementation at this level is the quickest and most direct way to share AEC information.

Because they will be transferred, translated, and eventually used by many AEC applications, the data structures of the objects in a central database must be generic enough to represent, handle, manipulate and manage all types of AEC information. Also, since the data structures are supported by conceptual models, the models must be generic. The ACRM provides such models for AEC. Thus, an AEC central database can be developed based on the ACRM models. We would like to use some version of the ACRM to develop a database and see how it can be used by many AEC applications as a central AEC database.

There are basically two mechanisms of sharing a central database -- one is direct sharing and the other indirect sharing. Direct sharing means that applications understand the data structures of the central database so they can directly use it. Unlike direct sharing, applications using the indirect sharing mechanism cannot directly use the central database because they only understand their own types of data structures. Therefore, indirect sharing requires translators that can translate the data from the central database to the forms that the applications understand. We intend to examine both mechanisms with using the ACRM as the base of the central AEC database.

StartPlan is the program that not only creates but also uses the ACRM-based databases. Two types of its document files can be used as AEC central databases -- one is the reference document and the other is the project document. The reason is that
they are based on the ACRM, and so have the capability to represent generic AEC objects information. This capability is just what a central AEC database needs. After one of the two documents has been created from StartPlan, it is ready to be used by AEC applications. Since the StartPlan program directly adopts the concepts of the data structures defined in the ACRM models, it understands the data objects in the documents. Therefore, StartPlan itself is a good testing program that uses the databases with the direct sharing method. In particular, StartPlan uses the reference documents as the reference library for its expert system, and the project documents as its working documents.

For the indirect sharing, Microsoft Project version 3.0 has been chosen as the test program that uses the central AEC database. Similarly, reference documents and project documents created from StartPlan can be the central databases. However, unlike the above case, in order to let MS Project use the databases, a translator is needed because MS Project does not understand the data structures used in the two documents. Such a translator has been developed in StartPlan (it is possible that it could be developed outside of StartPlan). The way that the translator works has been demonstrated in Section 3.5.3. The key point of this approach, however, is to use the project document as the central database. Because it is based on the ACRM models, it can store all the information that MS Project needs to represent. In other words, ACRM can support a central AEC database that is able to represent, store and provide all types of AEC object information that is needed by common AEC computer applications such as MS Project. Some other interesting issues were also revealed. One is that data translation between StartPlan and MS Project illustrates some aspects of data exchange between an object-oriented database
and a relational one. The project document is an object-oriented database based on object-oriented AEC models, while tables defined in MS project are based on the relational database concepts. In the future, we will do more tests for other popular packages including common database and spreadsheet software.

3.7.4 Support For File Exchange

Information sharing at the application level can be achieved through various mechanisms: file exchange, shared database and application programming interface (API). This section discusses how the ACRM models support file exchange and what the role of StartPlan is in this implementation.

There are two basic types of mapping to exchange files between applications as shown in Figure 3-23 and Figure 3-24. One is Direct Mapping and the other is Indirect Mapping. The former means that application files are translated directly from one type into another through translators, while the latter signifies that application files are translated into ACRM-based files and then into another type. The indirect mapping has the advantages over the other as listed below:

![Diagram of Direct Mapping]

**Figure 3-23: Direct Mapping**
Only one translator is needed between an application file and the ACRM file. Direct mapping requires one application to provide many translators for many other applications. For example, if a spreadsheet file needs to be transferred into MS Project, SuperProject, Primavera Planner file, three translators must be provided for the spreadsheet program; and each of the other three may need three translators as well for the same reason. However, for Indirect mapping, the spreadsheet program only needs to provide one translator for ACRM-based file, and so does each of the other three.

No file data will be lost. In direct mapping, the translation capability of the translator between two applications is application-dependent. Some data in one application file may not be able to be translated into the other application simply because it does not support that type of information. In contrast, the indirect mapping uses ACRM-based files as a medium to hold all the information in the application files. Because the ACRM files are able to represent all types of information about AEC objects, no data should be lost when translations occur through them.

Indirect Mapping takes advantage of the ACRM-defined data structures. Again, the ACRM files are able to represent all types of information. Hence, if application files
are translated into the ACRM file formats, although some of the data fields might be empty due to the incapability of the application to represent these types of information, they provide opportunities for other applications to fill the fields after the files have been translated. In the indirect mapping, some information in some applications still is not translated into programs that do not understand it, but it is in the ACRM files for other applications to use.

• Only one translator is needed for each application. This means that if every application provides only one translator to read in and write to ACRM files, all applications files are transferable among themselves. In contrast, it will not be possible for an application to provide all types of translators for all types of other applications. For example, a CAD system may not always offer a translator to translate its graphical data into the form of a spreadsheet file. However, if it has only one translator for ACRM files, the problem can be solved.

• Indirect Mapping takes advantage of international data exchange standards. It is likely that the ACRM files will eventually be replaced by a standard data structure format, e.g., Part 21 of ISO 10303, Exchange Structure Format For Product Data (ISO 92). Because this kind of formats is intended to be specially suitable for file exchange, their implementation will make file translation quick, easy, and accurate.

The StartPlan project document files can be used as a vehicle to carry the data between two applications. We have used two translators to do the test: one to translate Excel files to StartPlan project document files, the other to translate StartPlan
project document files to MS Project files. Because the StartPlan project document files are based on the ACRM models, they are able to represent all the information in the Excel files. Also, as it can represent all relationships between the objects, the links between the activities and resources are created automatically in the StartPlan files. After they have been translated into MS Project files, both objects' information and their relationships are transferred.

3.8 SYSTEM DESIGN AND PROGRAMMING

While parts of the sections in this chapter have already covered the concepts underlying the system design, the complete details of the system design and programming are beyond the scope of this thesis. Nevertheless, this section will address some of the important issues raised during the design and programming phase.

3.8.1 Run-time Objects In The Project Document

StartPlan objects are run-time objects created by the StartPlan application. Since the classes that represent the corresponding objects' types are ACRM-based, the relationships between objects reflect the relationships between the entities in the models.

As shown in Figure 3-25, the objects of projects, resources, products and participants can exist within the AEC-wide scope. However, plans or activities always exist as objects within a specific project. Therefore, although they are also of AEC-Object type, they only exist during the life time of the project. They are valid only under the
project scope. On the other hand, during the project stage, some of the AEC objects such as resources, products and participants are temporarily assigned to a particular project. For example, a piece of equipment might be at one time used for one project and at another time for another project; but at other times, it could be idle.

![AEC Objects Diagram]

Figure 3-25: AEC Objects And Their Scope Multiple Document

While recognizing these facts, we did not extend them AEC-wide. This means that although we know that some of the AEC objects are within the extended AEC scope during their life time, we do not use them as a direct AEC object. We focus on the scope of projects for the sake of simplicity.

The project document object stands at the center of the StartPlan program. It contains the major information about the working project at hand. As shown in Figure 3-
26, the objects that are considered to be associated with AEC projects only such as Plans are stored directly under the ProjectDocument object. The others, that for their entire life time belong to AEC scope but for the project time belong to the project, are stored under the Project object in the document. This indicates that they are presently in the project. We are not concerned with periods when they are out of the current project. Relationships between objects are represented as either part of or pointer to the other objects. For example, a Plan has activities; and an activity has a pointer to a product type object. Figure 3-26, gives a general view of how the StartPlan objects are organized and stored in the project document at run-time.

3.8.2 Interface (MDI) And The Doc/View Model

Many types of file documents need to be created and managed in StartPlan. The contents of the documents also need to be modified, and the modification will be done through a series of viewers. In addition, one document sometimes needs to be viewed from different viewers for different purposes by different project participants. For example, a project document might be viewed as activity breakdowns by estimators, or as Gantt charts by schedulers. To meet these requirements in StartPlan, we have chosen Borland ObjectWindows Library (OWL) Version 2.0 and its Doc/View Models to develop the StartPlan documents and viewers.
Figure 3-26: Objects In Project Document
3.8.2.1 Doc/View Models

Borland OWL (Borland 93) provides a Doc/View model to contain and manipulate computer information (Figure 3-27). The Doc/View model consists of three parts:

- Document objects that contain many different types of data and provide methods to access the data.

- View objects that form an interface between a document object and the user. They control how the data is displayed and how the user can interact with the data.

Figure 3-27: Doc/View Models
An application-wide document manager that maintains and coordinates document objects and the corresponding view objects.

The Doc/View model frees the programmer and the users from worrying about the type of data a file contains and how that data is presented on the screen.

### 3.8.2.2 How The Doc/View Model Works

The Doc/View Model (Borland 93) associates a document class with a view class. The document manager keeps a list of the associations. Each association is represented as a document template (not to be confused with the template schedules in StartPlan). A document class handles data storage and manipulation. It contains the information that is displayed on the screen. A document object, however, controls changes to the data, and when and how the data is transferred to computer memory such as the hard drives or RAM. When the user opens a document, it is displayed using an associated view class.

The view models manage how the data is displayed and how the user interacts with the data on the screen. In fact, the viewers form an interface between the display windows and the documents. Some document types might have only one associated view class, while others might have several. This is the Multiple Document Interface (MDI) mechanism.

Consequently, it has been shown that the advantageous characteristics of the OWL suit our needs and are not too difficult to implement.
3.8.2.3 The Doc/View Model in StartPlan

Figure 3-28 depicts the document classes, viewer classes, association templates classes and their relationships used in StartPlan.

![Diagram of Doc/View Model Classes in StartPlan](image)

**Figure 3-28: StartPlan Doc/View Model Classes**

There are three document classes to support three different types of file documents in StartPlan: ProjectDocument, ReferenceDocument and RuleBaseDocument. All three are derived from a Borland C++ Library class TFileDocument, which offers basic functionality of manipulating file data. All view classes must be derived from the Borland TView class and two types of viewers have been used in StartPlan, TListBox and TDialog. For the viewers derived from the TDialog, an additional class must be defined.
that manipulates the controls in the dialog. Examples are ProjectDialog, ActyInfoDialog, ReferenceDialog and RuleBaseDialog. The document classes must be associated with one or more view classes by association template class(es). For instance, ActyInfoTemplate associates ProjectDocument with ActyInfoView. At run-time, an object of class TDocManager will be created when StartPlan starts. Then a list of association template objects will be created and controlled by a Document Manager object. When an object of a document is required to be opened, the document manager will create an appropriate viewer for the type of the document according to the option selected by the user. Multiple viewers for one document can be opened at a same time.

We have found that from the perspective of programming, the use of Borland OWL Doc/View models provides great convenience; and the MDI supports the capability of representing large amounts and all types of AEC information in StartPlan.

3.8.3 Borland C++ Container Classes

The Borland C++ container library is known as the Borland International Data Structures (BIDS) (Borland 93). Containers are objects that implement common data structures. They provide processing methods to manipulate the data elements in the container, such as adding or deleting a container member. Containers can store common type of data like integers, reals, strings, also user-defined types such as classes. The advantage of using the containers is that they hide the details of data processing from the user so that they can concentrate on the problem of interest. This mechanism also increases the programmers' effectiveness and efficiency of coding.
The containers are implemented using templates (not to be confused with the template schedules in StartPlan). A container class that will hold the objects is created by declaring a template passing the object’s type as the parameter. There are many types of containers available from Borland and they are classified into two categories: Fundamental Data Structures (FDS), e.g., BTree, Hashtable, Linked list and Vector; and Abstract Data Types (ADT), e.g., Array, Set, and Stack. For example, an array container that holds class Activity type objects, with a vector as the underlying FDS, can be instantiated as follows:

```
TArrayAsVector<class Activity> ActivityArray(100);
```

Some containers can keep their contents in sorted order. If the above is written as follows:

```
TSArrayAsVector<class Activity> ActivityArray(100);
```

it means that the objects of type Activity in the container ActivityArray are sorted according to some criteria such as activity Ids. Containers can directly hold the copies of the objects in it (direct containers) or pointers to objects (indirect containers). The above two examples are direct containers, while the following is an indirect container:

```
TISArrayAsVector<class Activity> ActivityArray(100);
```

One reason to use an indirect container is that it avoids duplicated copies of the objects if they are needed at different places. More importantly, using templatized
containers is helpful in data exchanging of the objects in the containers. For example, a list of activities could be stored in a central database using Vectors as the FDS. When an application needs to transfer the objects into it and use the objects in its way, it can store the objects in a means suitable for its specific purpose. For instance, a Double-linked list type might be more suitable for a construction scheduling application than a Vector. The change of the implementation can often be done easily with a single change to a typedef keyword. The following example shows how this is done:

```
TSBagAsVector<class Activity> Activities; //this is for the central database

TISBagAsDoubleList<class Activity> Activities; //this is for the application
```

There are many places in StartPlan where the container classes are used. As shown in Figure 3-29, all the implementations of object lists are implemented as TArrayAsVector type either directly or indirectly for different cases. The reason is for the simplicity of the programming. Also, doing this does not affect the outcomes we expect from StartPlan, because its main functionality for now is to represent data rather than to provide full functionality to manipulate the data in a way that a complete scheduling program would do.

From our study, we have found that using Borland BIDS gives us a great deal of advantageous features as mentioned above, though there are still some drawbacks needed to be overcome in the future.
First, the BIDS libraries do not have all possible combinations of ADT-FDS pairs, as shown in Table 3-2. Hence, it is not yet possible to handle complex databases needed for all types of applications. It is expected that newer version of the Borland C++ compiler package will dramatically improve in this sense.

Table 3-2: Borland BIDS Library (Borland 93)

<table>
<thead>
<tr>
<th>FDS</th>
<th>Stack</th>
<th>Queue</th>
<th>Dequeue</th>
<th>Bag</th>
<th>Set</th>
<th>Sorted Array</th>
<th>Array</th>
<th>Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>DoubleList</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hashtable</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>BinaryTree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-29: Container Classes In StartPlan
Secondly, it is true that the indirect containers have the advantages as discussed above and proper use of them can be extremely important for an object-oriented database. However, because the indirect containers contain physical computer Random Access Memory (RAM) addresses as the references to its objects, these references are only temporarily valid in the memory. When these references need to be saved on to persistent memory like a hard drive for the object that holds them, there is no automatic way to do so. Because the second time they are loaded in the system, the objects’ addresses will be different. As a result, the memory addresses of objects cannot be used as the permanent references to the objects. The references in this sense become useless. One solution to this problem, in StartPlan, is to use their Ids, e.g., integer numbers, as the references to the objects instead of memory addresses. An example can be found in the code of class Activity to deal with Linkers. We would not regard this as a perfect idea, but hope that better solutions will be developed in the future.

3.8.4 Design of The StartPlan Expert System (SPES)

The StartPlan Expert System (SPES) is designed and developed separately from the StartPlan main application modules. Both, however, are compiled at the same time. The fundamental functionality of SPES is to catch answers from the user and produce a new ProjectDocument object — the template schedule. Four classes are created for SPES. They are:

- ExpertSystem;
- UserInterface;
• UserInterfaceDialog;
• InferenceEngine.

The ExpertSystem class is the core of SPES. It uses a pointer to ProjectDocument as its initialization parameter. It consists of an InferenceEngine, a UserInterface, a pointer to ReferenceDocument, a pointer to a RuleBaseDocument, and a pointer to a ProjectDocument. It will be started by the system and will perform the following tasks: invoke the User-interface, run the inference engine and finally return the Project Document object to the system.

An instance of the UserInterface class will be created by an ExpertSystem object. It gets questions from the rule base document and holds the answers from the user. It also returns the set of answers to ExpertSystem object.

For the user interface, a windows-type dialog is "drawn" in using the Borland Resource Workshop, which contains information about the questions and allows the user to control certain actions through typical windows command methods like buttons. The UserInterfaceDialog class is used for controlling this dialog and manipulating the data held in the dialog.

After getting the answers from the user-interface, an object of the InferenceEngine is made by the ExpertSystem. Then it gets the schedule segments structured as plans from the reference document in accordance with the answers and rules in the rule base document. Next, it makes all the plans into one plan and assigns project information to
each activity. Precedence relationships of the activities are created. Finally, all the project information including that particularly for the project at hand and the new plan is set in the project document. The ExpertSystem will at last return the project document object to the system. At this point, the job of the SPES is finished. It is the StartPlan application object's responsibility to delete the ExpertSystem object from the memory and present the project document in one of its viewers to the user.

Table 3-3 briefly depicts how the StartPlan Expert System objects function at run-time.

3.9 FUTURE DEVELOPMENT

There is significant opportunity for future development of StartPlan.

First, as mentioned in the thesis introduction, one of our research approaches is to use the implementation and testing results from StartPlan to help develop newer and better models. These models will then be re-implemented in the StartPlan program. Thus, StartPlan must be updated repeatedly based on the newer models. Once the models are changed, the C++ classes that implement them must also be changed accordingly. The changes may include the hierarchy structures and the attribute relationships of the classes.
<table>
<thead>
<tr>
<th>Sequence No.</th>
<th>Object Owner</th>
<th>Object</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Operating System Object</td>
<td>StartPlan Application Object</td>
<td>1. creates an object of ExpertSystem</td>
</tr>
</tbody>
</table>
| 2           | StartPlan Application Object | ExpertSystem Object | 1. reads reference document  
2. reads rule base document  
3. creates an object of UserInterface |
| 3           | ExpertSystem Object | UserInterface Object | 1. gets questions from rule base document  
2. pops up UserInterfaceDialog  
3. asks questions to the users upon request  
4. stores the answers in the memory  
5. clears the UserInterfaceDialog object from the memory  
6. returns the answers to the ExpertSystem object |
| 4           | StartPlan Application Object | ExpertSystem Object | 1. catches the answers  
2. clears the UserInterface object from the memory  
3. creates an object of InferenceEngine |
| 5           | ExpertSystem Object | InferenceEngine Object | 1. gets the Plans Ids and their repeated times according to the answers and rule base document  
2. gets the plans and project from the reference document  
3. combines all the plans into one and assigns project objects (e.g. resources) to the activities  
4. sets the project and the plan into the project document |
| 6           | StartPlan Application Object | ExpertSystem Object | 1. clears the InferenceEngine object from the memory  
2. returns the project document object to the StartPlan application object |
| 7           | Operating System Object | StartPlan Application Object | 1. clears the ExpertSystem object from the memory  
2. presents the reference document in a viewer as a reference for the user  
3. presents the rule base document in a viewer as a reference for the user  
4. presents the project in a viewer as the draft template schedule |
Secondly, more detailed implementations are sometimes needed. In fact, we have experienced this during the entire development of StartPlan. For example, initially, the entity "Component" in the models was simply implemented as a C++ class "Product". We did not develop a detailed implementation of it because our research focus is on process modeling. However, when even a simple high-rise building project example was taken into account, it was found that the simple implementation of the product models might not be enough, for the activities have interrelations with locations that are represented as Product types. These relationships are extremely important, especially when dealing with repetitive activities such as those in the high-rise building. Consequently, we must subtype the Product class as "Floor", "FloorPrototypes" and so on. From this perspective, since product modeling and process modeling are highly inter-related, changes on either side may require significant amount of work in the implementation. In spite of the this, we will still focus more on the process model implementation in StartPlan to match our primary research goal -- process modeling. In this sense, a truly generic "Activity" will be developed, and its implementation tests in StartPlan will be continued.

Thirdly, although the current StartPlan has all the basic functions required to work with simple tests and perform a simple real-world example, it is not yet good enough for more complex and comprehensive implementation. There are plenty of areas where further improvement is needed.

- The data structures of "ProjectDocument" need to be refined. In particular, multiple projects will be handled in one document. In the future, not only are there one type of project document, but also other types will be available such as estimates, contracts,
organizational management and so on. Secondly, upon the development of the project documents, a larger variety of viewers will emerge. They will be specially designed to serve different project participants for their unique purposes, e.g., Gantt chart views for scheduling, spreadsheet views for estimating and so on, and for representing larger amount of AEC information for the implementation purposes of the test-bed. Thirdly, translators are always important in sharing information. In StartPlan, more translators for more different types of applications will be developed. As the data structures of many types of classes in StartPlan will change upon the updating of the ACRM models, the translators need to follow accordingly. Moreover, a translator for exporting and importing a standard file format suitable for international file exchange is required.

- The StartPlan Expert System will also be redesigned and much more coding needs to be done. For the same reasons as the project document, the data structures of the rule base document and reference library document may need to be changed. The class "RuleSet" should be much better and more useful for a more complex expert system to handle more complicated examples. In addition, the inference engine is not powerful enough to create a template schedule that has the full information needed. For example, when schedule segments are retrieved from the reference library document and made as one bigger plan in the project document, the precedence links among the activities between the segments are not established automatically.

- The need for good computer applications that create construction document templates is not only in scheduling practice. Others are estimating, budgeting, resource planing,
daily project monitoring and so on. The benefits of using templates to create construction schedules are also available for other applications. Therefore, it is interesting and useful for StartPlan to be able to create many types of templates in a similar manner. Also, as all the templates and their final outcomes are to be created within one system, an integrated construction application environment is likely to be built. Easier data sharing, file translating and faster module processing will be expected.

3.10 CONCLUSION

This chapter has described an ACRM-based computer program -- StartPlan. While no widely accepted standard AEC conceptual reference models yet exist, the development of the current models has allowed some experimental practice for real-world examples in a model-based application. StartPlan is such a program that provides a test-bed for testing some of the basic ideas and implementation capabilities of the models. It uses an expert system to create template construction schedules and to assist in the implementation tests. Object-oriented techniques are used in both the implementation and the design and programming of StartPlan. We believe that StartPlan will be a great help in the future development of the ACRM models and further to that of CIC. The next chapter will describe a real-world construction project example used in StartPlan to accomplish the goals of the implementation.
Chapter 4

An Example

Chapter Abstract:

The implementation capabilities of the ACRM models and the workability of StartPlan cannot be tested and proven until real-world project information is represented in StartPlan. This chapter describes such a project and shows how the project is translated from REPCON and how it is used in StartPlan. At the end of the chapter, the results of this test will be summarized. These results will be useful in supporting the development of the ACRM models.

4.1 INTRODUCTION

As mentioned in the previous chapters, one of our research objectives is to use a real-world example to test the implementation capabilities of the AEC Core Reference Models (ACRM) used in StartPlan. Also, we would like to determine, through the use of the example, which concepts of the models are properly defined and where there is room for improvement. In particular, we would like to see if a wide range of construction project information can be represented in StartPlan and translated to other project management computer systems such as Microsoft Project.
We have decided to use a virtual construction project that was originally developed in REPCON (Russell 95). Although it is not a specific real-world project, it can be treated as a typical project because it has been tested in many real-world situations and proven to contain most of the information that a similar real-world project would have. Initially, the project information contained in the REPCON project file must be translated into StartPlan. Then the basic project information (particularly the scheduling information) must be used by the StartPlan Expert System to create new projects, and translated into other programs such as Microsoft Project (MSPProject 94) or Excel (MSExcel 93).

This chapter initially describes the basic characteristics of the project. Then it explains how the original project in REPCON is translated into the StartPlan project and how the StartPlan project can be used to test the capabilities of the ACRM models used in StartPlan. Next, the results from the test will be summarized. These results can be treated as the feedback information used in the research methodology to support the refinement of the AEC conceptual models. Finally, the impact of the results to the development of the AEC core models will be briefly discussed.

4.2 DESCRIPTION OF THE EXAMPLE PROJECT

4.2.1 Description of The Project

The example project used in StartPlan is a virtual mid-rise residential type building. The term "virtual" here means that the project is fictitious. A plan for the project was
initially developed in REPCON. The original purposes of making up the project were for testing the workability of REPCON and for use in educational courses and research programs as a standard construction project schedule. Table 4-1 depicts a description of the basic information about the project.

### Table 4-1: Description of The Example

<table>
<thead>
<tr>
<th>Project Type</th>
<th>Mid-rise Residential Building</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure Type</td>
<td>Reinforced Concrete</td>
</tr>
<tr>
<td>Basic Construction Method</td>
<td>Cast-In-Place Concrete</td>
</tr>
<tr>
<td>No. of Main Floors</td>
<td>9</td>
</tr>
<tr>
<td>No. of Typical Floors</td>
<td>1</td>
</tr>
<tr>
<td>No. of Underground Parkades</td>
<td>2</td>
</tr>
<tr>
<td>No. of Top Level Parkades</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 4.2.2 Description of The Plan

**Schedule Information**

Any plan in REPCON must define locations that will be linked to certain activities to do the project scheduling. Table 4-2 shows the list of the locations defined in REPCON for the project. Each location comes with a *Code* and a *Description*. In total, there are 26 locations that are to be linked to the activities in the plan. REPCON permits locations to be defined that are not actual physical spacing descriptions in the building, such as "P: Call for quotes". These kinds of locations are used to describe the stage of the linked activities.
Table 4-2: REPCON Locations of The Example Project

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPRJ</td>
<td>GLOBAL PROJECT LOCATION</td>
</tr>
<tr>
<td>APPR</td>
<td>APPROVALS &amp; PERMITS</td>
</tr>
<tr>
<td>OS</td>
<td>OFFSITE WORK</td>
</tr>
<tr>
<td>SW</td>
<td>SITWORK</td>
</tr>
<tr>
<td>FDN</td>
<td>FOUNDATIONS</td>
</tr>
<tr>
<td>P2</td>
<td>SECOND PARKADE LEVEL</td>
</tr>
<tr>
<td>P1</td>
<td>FIRST PARKADE LEVEL</td>
</tr>
<tr>
<td>MAIN</td>
<td>MAIN FLOOR</td>
</tr>
<tr>
<td>2</td>
<td>2ND FLOOR</td>
</tr>
<tr>
<td>3</td>
<td>3RD FLOOR</td>
</tr>
<tr>
<td>4</td>
<td>4TH FLOOR</td>
</tr>
<tr>
<td>5</td>
<td>5TH FLOOR</td>
</tr>
<tr>
<td>6</td>
<td>6TH FLOOR</td>
</tr>
<tr>
<td>7</td>
<td>7TH FLOOR</td>
</tr>
<tr>
<td>8</td>
<td>8TH FLOOR</td>
</tr>
<tr>
<td>9</td>
<td>9TH FLOOR</td>
</tr>
<tr>
<td>10</td>
<td>10TH FLOOR</td>
</tr>
<tr>
<td>ROOF</td>
<td>ROOF</td>
</tr>
<tr>
<td>PNTH</td>
<td>PENTHOUSE</td>
</tr>
<tr>
<td>QTES</td>
<td>P:Call for quotes</td>
</tr>
<tr>
<td>AWRD</td>
<td>P:Supply award</td>
</tr>
<tr>
<td>SAPR</td>
<td>P:Sample approvals</td>
</tr>
<tr>
<td>SDWG</td>
<td>P:Shop drawing preparation</td>
</tr>
<tr>
<td>SDRV</td>
<td>P:Shop drwg review &amp; revision</td>
</tr>
<tr>
<td>FABR</td>
<td>P:Fabrication</td>
</tr>
<tr>
<td>DEL</td>
<td>P:Delivery</td>
</tr>
</tbody>
</table>

There are a total of 66 activities in the plan including milestones. These activities are listed in Table 4-3, and each of them is described with an Activity Code, a Description, and a Location Range linked to the activity (other scheduling information such as activity precedence relationships is ignored in this table). This would correspond to 350 activities in a traditional CPM system that treats the work at each location as a separate activity.

**Contractors Information**

Table 4-4 is the list of the contractors in the example project. Each contractor has a Responsible Code, which is normally represented as the first one or two digits of the Activity Code of the activities that the contractor is responsible for. For example, the Code of the activity DEMOLISH EXISTING STRUCTURE is G00300 in which the letter “G” represents the contractor UBC CONSTRUCTION MANAGEMENT LAB.
Table 4-3: REPCON Activities of The Example Project

<table>
<thead>
<tr>
<th>Activity Code</th>
<th>Description</th>
<th>Location Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>040900</td>
<td>P: Precast panel metal inserts</td>
<td>SDWG, SDRV, FABR, DEL</td>
</tr>
<tr>
<td>100200</td>
<td>P: Precast panels - first 2 - 5</td>
<td>SAPR, SDWG, SDRV, FABR, DEL</td>
</tr>
<tr>
<td>100400</td>
<td>P: Main floor column covers</td>
<td>SDWG, SDRV, FABR, DEL</td>
</tr>
<tr>
<td>110200</td>
<td>P: Upper floor windows (2 - 10)</td>
<td>SAPR, SDWG, SDRV, FABR, DEL</td>
</tr>
<tr>
<td>110300</td>
<td>P: Main floor glazing</td>
<td>SAPR, SDWG, SDRV, FABR, DEL</td>
</tr>
<tr>
<td>110400</td>
<td>P: Main floor entrance doors</td>
<td>SAPR, SDWG, SDRV, FABR, DEL</td>
</tr>
<tr>
<td>160700</td>
<td>P: Elect finishing fixtures</td>
<td>QTES, AWRD, SAPR, FABR, DEL</td>
</tr>
<tr>
<td>G01500</td>
<td>PROJECT START</td>
<td>GPRJ</td>
</tr>
<tr>
<td>G02700</td>
<td>P: Fabricate Sysforms</td>
<td>AWRD, SDWG, SDRV, FABR</td>
</tr>
<tr>
<td>G02800</td>
<td>P: Plyform hardware</td>
<td>SAPR, SDWG, SDRV, FABR, DEL</td>
</tr>
<tr>
<td>100300</td>
<td>P: Precast panels - first 6 - 10</td>
<td>FABR, DEL</td>
</tr>
<tr>
<td>G00200</td>
<td>OBTAIN DEMOLITION PERMIT</td>
<td>APPR</td>
</tr>
<tr>
<td>G00300</td>
<td>OBTAIN EXCAVATION PERMIT</td>
<td>APPR</td>
</tr>
<tr>
<td>010200</td>
<td>CLEAR SITE</td>
<td>SW</td>
</tr>
<tr>
<td>G00500</td>
<td>MOBILIZE SITE</td>
<td>SW</td>
</tr>
<tr>
<td>G01600</td>
<td>DEMOLITION COMPLETE</td>
<td>SW</td>
</tr>
<tr>
<td>020100</td>
<td>EXCAVATION</td>
<td>P1 - P2, P2 - P2, FDN - FDN</td>
</tr>
<tr>
<td>160400</td>
<td>INSTALL TEMPORARY POWER</td>
<td>SW</td>
</tr>
<tr>
<td>030100</td>
<td>MESH, TIEBACKS &amp; SHOTCRETE</td>
<td>P1 - P2, P2 - P2</td>
</tr>
<tr>
<td>040200</td>
<td>REINF FOUNDATIONS</td>
<td>FDN</td>
</tr>
<tr>
<td>G00600</td>
<td>FORM &amp; PLACE FOUNDATIONS</td>
<td>FDN</td>
</tr>
<tr>
<td>G00700</td>
<td>ERECT CRANE</td>
<td>SW</td>
</tr>
<tr>
<td>G04000</td>
<td>REINF P2 WALLS &amp; COLUMNS</td>
<td>P2</td>
</tr>
<tr>
<td>G01000</td>
<td>FORM &amp; PLACE P2 WALLS/COLS</td>
<td>P2</td>
</tr>
<tr>
<td>G01900</td>
<td>CRANE INSTALLED</td>
<td>SW</td>
</tr>
<tr>
<td>G02500</td>
<td>CRANE USAGE</td>
<td>SW</td>
</tr>
<tr>
<td>G00900</td>
<td>BACKFILL FTGS &amp; GRADE ROG</td>
<td>FDN</td>
</tr>
<tr>
<td>040300</td>
<td>REINF SLAB ON GRADE</td>
<td>P2</td>
</tr>
<tr>
<td>G01100</td>
<td>FORM &amp; PLACE P2 SLAB ON GRADE</td>
<td>P2</td>
</tr>
<tr>
<td>G01200</td>
<td>FORM &amp; PLACE P1 FLOOR SLAB</td>
<td>P1</td>
</tr>
<tr>
<td>040500</td>
<td>REINF P1 FLOOR SLAB</td>
<td>P1</td>
</tr>
<tr>
<td>040600</td>
<td>REINF P1 WALLS &amp; COLUMNS</td>
<td>P1</td>
</tr>
<tr>
<td>G01300</td>
<td>FORM &amp; PLACE P1 WALLS/COLS</td>
<td>P1, 3 - 10, ROOF, PNTH - PNTH</td>
</tr>
<tr>
<td>G00100</td>
<td>STRIP/FORM SUPERSTRUCT FLOOR SLABS</td>
<td>MAIN, 2 - 10, ROOF, PNTH - PNTH</td>
</tr>
<tr>
<td>040100</td>
<td>REINF SUPERSTRUCT FLOOR SLABS</td>
<td>MAIN - MAIN, 2 - 10, ROOF - ROOF, PNTH - PNTH</td>
</tr>
<tr>
<td>150500</td>
<td>MICH SLVSRG-SSTRUCT FLOOR SLABS</td>
<td>MAIN, 2 - 10, ROOF - PNTH</td>
</tr>
<tr>
<td>160100</td>
<td>ELECT CONDUIT SSTRUCT SLABS</td>
<td>MAIN, 2 - 10, ROOF, PNTH - PNTH</td>
</tr>
<tr>
<td>G02600</td>
<td>CONSTRUCT SUPERSTRUCTURE</td>
<td>MAIN - PNTH</td>
</tr>
<tr>
<td>G02400</td>
<td>PLACE SSTRUCT SLAB CONCRETE</td>
<td>MAIN - PNTH</td>
</tr>
<tr>
<td>G02000</td>
<td>FORM/PLACE SSTRUCT WALLS/COLS</td>
<td>MAIN, 2 - 10, PNTH - PNTH</td>
</tr>
<tr>
<td>040800</td>
<td>REINF SUPERSTRUCT WALLS/COLS</td>
<td>MAIN, 2 - 10, PNTH - PNTH</td>
</tr>
<tr>
<td>150400</td>
<td>MICH SLVSRG-SSTRUCT WALLS/COLS</td>
<td>MAIN, 2 - 10, PNTH</td>
</tr>
<tr>
<td>160600</td>
<td>ELECT CONDUIT SSTRUCT WALLS/COLS</td>
<td>MAIN, 2 - 10, PNTH - PNTH</td>
</tr>
<tr>
<td>G05000</td>
<td>STRIP SSTRUCT WALLS/COLS</td>
<td>MAIN - 10, PNTH - PNTH</td>
</tr>
<tr>
<td>150200</td>
<td>ROUGH-IN VENTILATION SYSTEM</td>
<td>P2 - P1, MAIN, 2 - 10, PNTH</td>
</tr>
<tr>
<td>151000</td>
<td>ROUGH-IN SPRINKLER SYSTEM</td>
<td>P2 - P1, MAIN, 2 - 10, PNTH</td>
</tr>
<tr>
<td>160500</td>
<td>INSTALL ELECTRICAL ROOM</td>
<td>P2</td>
</tr>
<tr>
<td>090100</td>
<td>INSTALL ROOFING</td>
<td>PNTH, ROOF</td>
</tr>
<tr>
<td>100100</td>
<td>INSTALL PRECAST PANELS</td>
<td>2-10, MAIN</td>
</tr>
<tr>
<td>110100</td>
<td>INSTALL WINDOWS &amp; DOORS</td>
<td>2-10, MAIN</td>
</tr>
<tr>
<td>020100</td>
<td>ROOF COMPLETED</td>
<td>ROOF</td>
</tr>
<tr>
<td>120100</td>
<td>INSTALL STEEL STUDDING</td>
<td>2 - 10, MAIN</td>
</tr>
<tr>
<td>160200</td>
<td>ELECTRICAL ROUGH-IN</td>
<td>P2 - P1, 2 - 10, PNTH - PNTH, MAIN</td>
</tr>
<tr>
<td>G01400</td>
<td>FORM &amp; PLACE PLANTERS/WALLS</td>
<td>SW</td>
</tr>
<tr>
<td>G01700</td>
<td>STRUCTURE FINISHED</td>
<td>PNTH - PNTH</td>
</tr>
<tr>
<td>040700</td>
<td>REINF PLANTERS &amp; WALLS</td>
<td>SW</td>
</tr>
<tr>
<td>120200</td>
<td>INSTALL DRYWALL</td>
<td>2 - 10, MAIN</td>
</tr>
<tr>
<td>090200</td>
<td>INSTL EX WATERPROOF MEMBRANE</td>
<td>SW</td>
</tr>
<tr>
<td>120300</td>
<td>TAPE &amp; FILL DRYWALL</td>
<td>2 - 10, MAIN</td>
</tr>
<tr>
<td>000800</td>
<td>DISMANTLE CRANE</td>
<td>SW</td>
</tr>
<tr>
<td>050100</td>
<td>LANDSCAPING</td>
<td>SW</td>
</tr>
<tr>
<td>150300</td>
<td>FINISH VENT/SPRINKLER SYSTEM</td>
<td>P2 - P1, 2 - 10, PNTH - PNTH, MAIN</td>
</tr>
<tr>
<td>160300</td>
<td>ELECTRICAL FINISHING</td>
<td>P2 - P1, 2 - 10, PNTH - PNTH, MAIN</td>
</tr>
<tr>
<td>G02000</td>
<td>CRANE DISMANTLED</td>
<td>SW</td>
</tr>
<tr>
<td>G01800</td>
<td>SUBSTANTIAL COMPLETION</td>
<td>GPRJ</td>
</tr>
</tbody>
</table>
Table 4-4: Contractors In The Example Project

<table>
<thead>
<tr>
<th>Responsible Code</th>
<th>Contractor Name</th>
<th>Contractor Trade</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>UBC CONSTRUCTION MANAGEMENT LAB</td>
<td>General Contractor</td>
</tr>
<tr>
<td>1</td>
<td>OUR BEST CLIENT</td>
<td>Owner</td>
</tr>
<tr>
<td>2</td>
<td>JOE'S EXCAVATING</td>
<td>Excavation</td>
</tr>
<tr>
<td>3</td>
<td>SHOTCRETE INC.</td>
<td>Shotcrete &amp; Tiebacks</td>
</tr>
<tr>
<td>4</td>
<td>ART'S REINFORCING</td>
<td>Reinforcing Steel</td>
</tr>
<tr>
<td>5</td>
<td>GOOD TASTE LANDSCAPING</td>
<td>Landscaping</td>
</tr>
<tr>
<td>9</td>
<td>WATER-TITE INC.</td>
<td>Roofing &amp; Waterproofing</td>
</tr>
<tr>
<td>10</td>
<td>CONFORCE PRECASTERS LTD.</td>
<td>Precast Concrete</td>
</tr>
<tr>
<td>11</td>
<td>CLEAR SIGHT INC.</td>
<td>Windows &amp; Exterior Doors</td>
</tr>
<tr>
<td>12</td>
<td>RELIABLE DRYWALL</td>
<td>Drywall</td>
</tr>
<tr>
<td>15</td>
<td>SMITH &amp; JONES MECHANICAL</td>
<td>Mechanical, Plumbing &amp; Ventilation</td>
</tr>
<tr>
<td>16</td>
<td>LIGHTNING ELECTICAL LTD.</td>
<td>Electrical</td>
</tr>
</tbody>
</table>

Resource Information

Because the example project is a virtual project, the resource information is simple.

Table 4-5 shows the only three resources in the project.

Table 4-5: Resources In The Example Project

<table>
<thead>
<tr>
<th>Resource Code</th>
<th>Resource Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Labour (all types)</td>
</tr>
<tr>
<td>2</td>
<td>Tower Crane</td>
</tr>
<tr>
<td>3</td>
<td>Hydraulic Excavators</td>
</tr>
</tbody>
</table>

4.3 WHY THIS EXAMPLE WAS CHOSEN

There are several reasons why this example is chosen for the test:

- **representativeness**: mid-rise buildings are typical construction projects. They contain most of the typical construction activities involving typical construction
methods conducted by many different types of participants such as the owners, contractors or suppliers. A plan for a mid-rise building can be easily extended to a high-rise project with minor alternations. This characteristic makes it suitable for use it in the StartPlan Expert System.

- quality: the plan created from REPCON has been thoroughly tested for years. It has been used as a standard project plan in REPCON in both education and research, and has proven satisfactory.

- non-traditional scheduling information: the plan has most types of construction activities including repetitive ones. Specifically, it uses REPCON Activities those have different definitions compared to that in normal project management software packages. This leads to different ways of representing the relationships between activities and products. It is challenging and interesting to see whether the generic activities in StartPlan are capable of fully representing the information of REPCON activities.

- suitability for comparison: the project plan itself is used in REPCON as a template plan based on which new and more complex plans can be built. Therefore, it is useful to see the differences between the original template and the ones created from StartPlan based on the original.

- workability: its size is reasonable for conducting the test. It is not so small that it does not contain enough information; and it is not too big to deal with its detailed information.
4.4 TRANSLATION OF THE EXAMPLE

4.4.1 About REPCON

REPCON is a computer-based information system developed by Alan D. Russell (Russell 95). It is a construction project planning and scheduling software aimed at providing managerial solutions for all project participants including general contractors, construction management firms and developers. REPCON specializes in dealing with repetitive activities such as those in high-rise buildings or highway projects. A core implementation in REPCON is the definition of activities and their relationships. Especially, REPCON emphasizes that an activity may occur at a series of locations (e.g., at the sequential floors in a high-rise); thereby, a single REPCON activity is specified with a number of locations as its attributes. Some important concepts about the REPCON activities are briefly explained as follows (Russell 95):

- *Activity*: Unlike the traditional activity defined in common software systems, a REPCON activity is stipulated as any event or operation that takes place at a unique location or at a series of locations. For either case, a single activity is associated with the specified location(s).

- *Activity Types*: REPCON activities are classified into five types to further represent the specific relationships between them both in terms of time and locations. They are: ordered, continuous, shadow, hammock and milestone.

- *Ordered Activities*: An ordered activity must be done at the sequence of its locations specified by the user. Within the activity, traditional precedence
relationships between the locations exist. They are: finish-start, finish-finish, start-
start, and start-finish relationships. In some cases, work at a location is not only
dependent on the completion status of the work at another location, but also on
that at some location(s) of its successor activities. For example, the start of
activity A, at the 1st floor, depends on its completion at the main floor and the
start of activity B at the 1st floor. In this situation, two constraints are imposed to
a work at a location.

- **Continuous Activities:** An Order Activity becomes a continuous activity when the
  following additional constraint is imposed: the work at all its locations must
  proceed continuously from location to location, without interruption, and
  consistent with the activity's internal logic. This means that the work at the first
  location in the sequence cannot be started until the above condition is met. As a
  result, the work at all the locations of a continuous activity must move as a single
  unit in accordance with the precedence constraints imposed by other activities in a
  schedule.

- **Shadow Activities:** A shadow activity has no internal logic relationships between
  its locations. It can be started once its predecessor constraints allow it regardless
  of any completion status of the work at preceding locations. Shadow and ordered
  activities then stipulate the same type of work that occurs at different locations
  where no continuity constraints can or need to be maintained.

- **Hammock Activities:** The duration at each location of a hammock activity is
determined by its predecessors and/or successors. Such activities cannot be in a critical path. The current version of REPCON does not have resource assignments to any of the hammock activities.

- **Milestone**: A milestone represents an activity with its duration equal to zero. It is normally used to indicate the beginning or end of a sub-project.

- **Typical Predecessor**: An activity that precedes another at all locations where both activities occur. Only activities having multiple locations can have typical predecessors.

- **Non-typical predecessor**: The activity that precedes the other at only one location.

- **Multiple Precedence Relationships**: REPCON allows compound precedence relationships between two activities in the case of Start-Start and Finish-Finish relationships (Figure 4-1). The relationship that determines the timing of the successor is the governing constraint.

![Figure 4-1: SS Governing Relationship](image)

- **Lags and Offsets**: Intervals of either time or space must be defined between the precedence relationships. A time interval is called a lag, while an offset is a space
(location) interval. The lags exist in both the typical and non-typical relationships. For example, defining the precedence relationship between two activities as FS with the lag equal to 2 means that the successor must commence after 2 days of the finish of the predecessor. Unlike the lags, the offsets are only applicable for typical relationships that are between the activities having multiple locations. If one defines an FS = 0 relationship with an offset of 2, it means that when the predecessor is completed at location #3, the successor may start at location #1.

4.4.2 About Microsoft Project

Microsoft Project Version 4.0 (MSProject 94) is developed by Microsoft with at least 20 companies offering add-ons to it. The package covers a wide range of project management capabilities including sophisticated charting, multi-project organization, project planning and development, statistical simulation, risk analysis, and comprehensive e-mail solutions and so on. The following list depicts some concepts of the terms used in Microsoft Project, and Figure 4-2 shows the terms as entities and their relationships.

Figure 4-2: Some Terms Used In MSProject And Their Relationships
• **Task**: A job that has a beginning and an ending. The completion of a task is important to the project's completion. Projects are made up of tasks.

• **SummaryTask**: A task that is made up of subtasks and that also summarizes those subtasks. The summary task information, such as duration, cost, and so on, is determined by the information from the subtasks.

• **Subtask**: A task that is part of a summary task. The subtask information is consolidated into the summary task.

• **Project**: A Project contains the most current information about a project, including actual dates and durations for tasks that have started and the latest projected dates and durations for tasks that have not yet started.

• **SubProject**: A project used in another project, where it is represented as a single task. The subprojects are used to break projects into manageable units and to reduce memory usage.

• **Master Project**: A project containing one or more tasks that represent and are linked to subprojects.

• **Combined Project**: A project that is used in another project and contains other subprojects.
4.4.3 Translation

Translation of REPCON Locations

REPCON locations are translated into StartPlan as StartPlan Product objects containing the same information since REPCON locations typically refer to the physical location or part of the product being worked on. In particular, all the characteristics of a REPCON location should be represented as the attribute information of a Product object in StartPlan. In most cases, REPCON locations such as floors can be perfectly matched to StartPlan products. However, in some other cases, they do not represent typical building locations that identify spatial relationships between physical building components. For example, "Shop drawing preparation" can be a location in REPCON. In fact, REPCON locations are quite loosely defined to represent a large range of information related to REPCON activities. In this situation, the "Shop drawing preparation" is related to the activity in the same way as other locations in REPCON. It may not be suitable to represent such REPCON locations as a StartPlan product and relate it to the StartPlan activity in the same way as the other products. For the example project, we still use the StartPlan products to represent this type of REPCON information since it does not affect the results when only simple tasks are performed in the program. Nevertheless, we will do more research work on this problem in the future.

Figure 4-3 shows a list of StartPlan products used in the example project. Most of them are translated from REPCON locations and the other locations, such as the FLOORs over the 10th floor, are created to be used in the StartPlan Expert System. There are a
total of 46 products in the reference file with each represented by an id number and name.

... 
46
501 GLOBAL PROJECT LOCATION
502 APPROVALS & PERMITS
503 OFFSITE WORK
400 SITWORK
200 FOUNDATIONS
101 SECOND PARKADE LEVEL
100 FIRST PARKADE LEVEL
1 MAIN FLOOR
2 2ND FLOOR
3 3RD FLOOR
4 4TH FLOOR
5 5TH FLOOR
...
201 ROOF
202 PENTHOUSE
300 P:Call for quotes
301 P:Supply award
302 P:Sample approvals
303 P:Shop drawing preparation
304 P:Shop drwg review & revision
305 P:Fabrication
306 P:Delivery
...

Figure 4-3: A List of Locations In The Reference Document File

Translation of REPCON Activities

Typical REPCON activities that have multiple locations are translated into a number of StartPlan activities. That is, the work at one location is translated into a StartPlan activity having an attribute of a pointer to the StartPlan product object representing the location. Therefore, the number of the activities is the same as the number of the locations. Figure 4-4 shows the mechanism of this translation. In addition, the REPCON precedence relationships need to be broken down to be represented in
between the StartPlan activities. All the StartPlan activities having a same product will then be grouped as a plan in the plan list object of a reference document file that will be used in the StartPlan Expert System. A sample plan used in the reference document representing the work for a concrete footing foundation is shown in Figure 4-5.

Figure 4-4: Translation from REPCON Activity To StartPlan Activity

Figure 4-5: A Sample Plan In The Reference Document File
Non-typical REPCON activities can be translated directly without being split. Only time intervals are considered.

*Translation of Contractors*

Though the definition of a StartPlan Participant is slightly different from that of a REPCON contractor, the former comprises the latter. Therefore, all the information regarding the REPCON project contractors can be translated into the StartPlan project information. In particular, each contractor is represented as a participant object in StartPlan. The list of participants is translated into the StartPlan reference document file. Figure 4-6 is a portion of a reference document file depicting a list of the participant objects translated from the REPCON contractors. In total, there are 12 participants and each is represented with an id number and a name.

```
... 
12
100 UBC CONSTRUCTION MANAGEMENT LAB
1 OUR BEST CLIENT(owner)
2 JOE'S EXCAVATING(Excavation)
3 SHOTCRETE INC.(Shotcrete & Tiebacks)
4 ART'S REINFORCING(Reinforcing Steel)
5 GOOD TASTE LANDSCAPING(Landscaping)
9 WATER-TITE INC.(Roofing & Waterproofing)
10 CONFORCE PRECASTERS LTD.(Precast Concrete)
11 CLEAR SIGHT INC.(Windows & Exterior Doors)
12 RELIABLE DRYWALL(Drywall)
15 SMITH & JONES MCHANICAL (Mechanical, Plumbing&Ventilation)
16 LIGHTNING ELECTICAL LTD.(Electrical)
... 
```

*Figure 4-6: Participant List In The Reference Document File*
Translation of Resources

The definition of a StartPlan Resource is rather simple at this stage; however, it is
good enough to represent the basic information of a REPCON resource. A StartPlan
Resource object can be directly used for representing a REPCON resource. Figure 4-7
shows a portion of the reference file representing the three resource objects translated
from REPCON resources in the sample project. Similarly, each object is represented with
its id number and name.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Labour (all types)</td>
</tr>
<tr>
<td>2</td>
<td>Tower Crane</td>
</tr>
<tr>
<td>3</td>
<td>Hydraulic Excavators</td>
</tr>
</tbody>
</table>

Figure 4-7: Resource List In The Reference Document File

Procedures of Translation

The Procedures of translating the original REPCON example project plan into
StartPlan objects are shown below:

1. Define a report table in REPCON to print the activity list. Make sure all of
   the information about the activities is included in the table.

2. Output the activity list to a printer file, which is in ASCII format.
3. Import the file into MS Excel.

4. Define some MS Excel macros to reorganize the format of data in the file including parsing the data, splitting the activities, calculating the precedence relationships between the new activities, creating new columns to hold new information such as predecessors, and regrouping the activities according to the locations (products).

5. All activities at one location are grouped and copied to another spreadsheet or another file. They will be treated as a single plan.

6. Each of the plans is manually translated into a StartPlan reference document file.

4.5 USING THE EXAMPLE IN STARTPLAN

As presented in the previous chapter, the example project is originally created in REPCON and translated into a reference document. The document is used in the StartPlan Expert System as the knowledge base. In particular, a project object is created in the reference document object in the system. The information contained in the project includes a resource list, a participant list, and a product list. Each schedule segment is represented as a Plan object in the reference document and all of the segments are listed as a plan list in the document. When required by the user to create a new similar project, e.g., a 15-floor high-rise building, StartPlan initiates its expert system. The reference document is opened and the information is retrieved and assigned to the project document
object. Consequently, a new project plan template based on the reference document is created. The user can then either use the new template to create a complete plan, or translate it into other computer programs such as Microsoft Project or Excel.

We have been able to create new plan templates successfully for similar projects (up to 25 floors) in StartPlan using the reference document that contains the example project information translated from REPCON. In order to do this, the user has to run the SPES (StartPlan Expert System) and to answer several questions through the SPES user interface. The questions will provide the basic information about the project at hand, such as project type, building structure type, foundation type and number of typical floors, etc. This information will be useful for the SPES to query the reference document through the rule base document. The resulted plan will contain all the activities for the project and the total number of activities will be approximately 250 for a 10 floor concrete building. We have also translated the new projects into Microsoft Project and Excel using the translators that we developed for StartPlan. For the former, in order to open the file the Microsoft Project, a table is required to be defined so that the name and data type of each column of the table can be matched to that in the translated file. For the latter, there is no need to do so because Excel can basically accept any common data types in its columns. It should be noted that for the current version of StartPlan, all the translated files are in ASCII file format. Therefore, when the files are to be opened in Microsoft Project and Excel, it must be in ASCII open mode.
4.6 RESULTS

As stated before, the StartPlan program is supported by the ACRM models. The use of the real-world project in StartPlan is to test the implementation capabilities of the models. Also, the results of the test can reflect the advantages and disadvantages of the models and can be used as feedback information to support the development of the models. The results of the test are summarized as follows:

- The basic concepts of the entities in ACRM such as Activity, Resource, Participant and Product, and their relationships are appropriately defined. First, they are able to describe the real-world situation accurately, completely, and concisely. Second, they hold the promise to support more detailed or lower level models in computer applications. In our case, the The_Scheduling_View_Model is used as the application models in StartPlan to deal with construction scheduling problems.

- The implementation capabilities of ACRM are at least as good as that of other similar type software, such as Microsoft Project, because StartPlan can basically describe the same type and amount of information as the other software. The test shows that all the project and scheduling information in REPCON and Microsoft Project can be successfully represented in StartPlan. In some situations, the test indicates that the models used in StartPlan are better than those in other software. For example, StartPlan recognizes the importance of building products using the entity Product and emphasizes the relationship between it and the activities. However, Microsoft Project does not have a distinctive definition of a building product and therefore is unable to
take the relationship between a product and a task into account. Also, the definition of a product in StartPlan is more generic than a location in REPCON. StartPlan emphasizes the nature of building products instead of only the physical spacing relationships of them. Thus, a product object in an application like StartPlan can be used not only for scheduling, but also for other purposes such as architectural design, structural design or estimating.

- The entity Activity, which is the core of ACRM, is generic enough to describe the nature of a unit of construction work in a computer information form and to support different views of it (in our case, the scheduling activity view). It can also support the translation of different types of activities. In the test, we have successfully translated a REPCON activity into Microsoft Project tasks through StartPlan activities.

Despite the above advantages of ACRM, the test also reveals some points where improvements are needed to the model.

- Although the Activity entity is generic enough to represent any type of work in a construction project, its implementation in a computer application still needs to be detailed enough to suit the purpose of the application. For example, the Scheduling_View_Model is the lower level view of the AEC_Core_Model. We use the Scheduling_View_Model as the application model of StartPlan. Therefore, the Activity in the Scheduling_View_Model is a scheduling view of that in the AEC_Core_Model and is also a generic entity. Nevertheless, subtypes of the Activity are needed to suit the scheduling purpose. For example, a kind of summary activity
might be needed to represent a group of activities that have the same characteristics so that the tasks of construction scheduling can become quicker and easier. The examples of similar approaches found in Microsoft Project are Summary Task and MasterProject. Also, a typical REPCON activity is actually a kind of summary activity representing a group of the same type of tasks occurring at a series of locations.

• Although the ACRM stresses the relationship between Activity and Product, the test clearly indicates that in order to fully take advantage of this approach, more detailed implementation of the Product View Model is essential. For instance, when the learning curve effect needs to be considered in construction planning, the products associated with a group of a same type of activities must also be of the same type, e.g., same type of columns. In this case, subtypes of the Product entity are useful to classify the building components. In other words, good process models must be developed based on good product models (or product view of the AEC_Core_Model), though each has its own focus. While StartPlan uses the Process View Model as the application model because it deals with construction activity plans, other applications may apply the Product View Model. Nevertheless, an AEC central database must be based on the combination of both models.

4.7 CONCLUSION

This chapter describes the example project used in StartPlan to test the implementation capabilities of the ACRM models. It also explains the testing
methods, in particular, the approaches of translating the project from REPCON to StartPlan and Microsoft Project. The results of the test are summarized. They will be useful in the future development of AEC computer information models.
Chapter 5

Conclusions

Chapter Abstract:

To conclude this thesis, this chapter first summarizes the two key terms of the entire research and development: ACRM and StartPlan. It then lists the contributions of this research work and finally proposes some potential future research and development topics and directions.

5.1 SUMMARY

This thesis is based on the theme that Computer-integrated Construction (CIC) relies on the AEC project information being shared and exchanged among various AEC computer applications used by many different project participants; in turn, this information sharing is based on the AEC information models (Froese and Yu 94). While the current development of the models is still in a conceptual stage, we have been investigating such models, and believe that some basic central concepts are sufficiently well developed for testing some of their implementation capabilities in a computer program using a real-world example (Froese and Yu 94). Throughout our research, we have proposed a version of AEC information models -- the AEC Core Reference Models (ACRM) -- combining the good features of many other models (Froese 94c). We have also developed a computer application program, StartPlan, based on the ACRM and used it as a test-bed for these
experiments. Therefore, ACRM and StartPlan are the central terms in the thesis.

- **ACRM**: A set of AEC conceptual models that serves as the AEC core or shared models. It is developed based on our investigation of current AEC information models developed by other researchers, and on our own opinions of the characteristics of the models. A model-layering technique is used in ACRM to reflect different perspectives to the AEC information. The highest level is the AEC_Core_Model, which is aimed at identifying the major entities in the AEC industry such as activities, building components and resources, etc. The second level is the refinement of the AEC_Core_Model, but focuses on different types of AEC applications. For example, the Process_View_Model focuses on the construction process. The lower levels will be AEC application models that directly support various AEC computer applications.

- **StartPlan**: A computer program that has been developed based on the ACRM. It provides a test-bed to test the implementation capabilities of the ACRM. StartPlan is a program that creates “template” construction plans. Based on a user’s description of a construction project, StartPlan will assemble a construction schedule from a library of “good” schedule segments using an expert system developed within the system. It will then alter the schedule to suit the details of the project at hand. It will also export the plan to one of a variety of formats to be used as a starting template for creating a new project plan in other systems such as Microsoft Project. StartPlan can represent a wide range of AEC project information and allow the information to be shared and translated between different types of AEC computer applications.
The two key concepts in this research, ACRM and StartPlan, have been tested using a real-world example and the results of the tests have been used to help the development of the AEC information models.

5.2 CONTRIBUTIONS

The following items are considered to be the primary contributions of this research project to the AEC domain. Each of them will have the potential to help the future development of CIC:

1. *Overall approach*, of building a specific computer application that is based on an explicit representation of generic project models, and of using this as a test-bed to test the basic implementation capabilities of the models, as presented in Chapter 1 and illustrated throughout the thesis.

2. *AEC Core Reference Models (ACRM)*, a version of AEC conceptual models, that combines good features of many currently developed models, plus our opinions about the models, as described in Chapter 2.

3. *StartPlan*, a computer program for testing the basic ideas and implementation capabilities of the AEC information models, as described in Chapter 3.

4. *Results from the tests*, that reveal the advantages and disadvantages of the ACRM (outlined below), that provide a useful and significant impact on the
advantages:

- The basic concepts of the entities in the ACRM such as Activity, Resource, Participant and Product, and their relationships are appropriately defined.

- The implementation capabilities of the ACRM are as good as those of other similar types of software such as Microsoft Project, since StartPlan can describe the same type and amount of AEC information as the other software.

- The core of the ACRM, the Activity entity, is generic enough to represent the nature of a unit of real-work construction work and to support the translation between different types of activities between computer applications.

disadvantages:

- The implementation of the entity Activity is not detailed enough to support many types of computer applications.

- More detailed implementation of the Product_View_Model needs to be developed to assist in the implementation of the Process_View_Model.
The following items are also considered to be the contributions of the thesis:

1. An introduction to Computer-integrated Construction (CIC) as described in Chapter 2.

2. An introduction to computer information models and information sharing as explained in Chapter 2.

3. A review of current research efforts on developing AEC information models and of the current models as described in Chapter 2.

4. Experiences on implementing the models in a computer application as shown in Chapter 3.

5. Issues raised from programming and executing the application as discussed in Chapter 3.

6. A library of C++ class representation of ACRM, that will be useful for other implementation work, as presented in Chapter 3.

7. A run-time library of ACRM models, that will be useful for other testing computer applications, as mentioned in Chapter 3.

8. Experiences in coding a computer program to create “template” schedules for construction projects and on developing a knowledge-based system for creating such a template, as described in Chapter 3.
Through these contributions, this research could provide a useful reference for future developments of AEC information models and implementation practice for the models. However, none of these contributions is considered to be the final or definitive solution to the problem domain. Further research and development work is needed as described in the following section.

5.3 FUTURE WORK

The research work in this thesis has pointed out that further research and development are required to fully achieve the goals of CIC, including the following:

- The research and development of the AEC information models must continue because the current version of the ACRM is imperfect. As described in the “Results” section of Chapter 4, the drawbacks of the ACRM must be overcome to suit the need of computer applications based on it. In particular, although the core of the ACRM -- the entity “Activity” -- is generic enough in the high-level reference models, its implementation in the lower level information models, such as an application model, is not detailed enough to represent all the information in many forms. Thus, a more generic or more detailed “Activity” must be further developed in future AEC conceptual models. We have proposed a version of “Activity” using a technology called “Summary_Activity” to represent the generic activity. Figure 5-1 shows a portion of such models. A “Summary_Activity” represents all the activities that have one attribute in common. For example, the “Time_Summary_Activity” represents all the activities that occur during the same time period, while the
"Component_Summary_Activity" represents all the activities that take place on the same building component such as column #2. This technology is only one method in attempting to achieve a generic and detailed "Activity". However, this research is only at its initial stage, and much more work will be done in the future. Also, there may be other better technologies yet to be discovered and developed.

Figure 5-1: Summary_Activity Models
• StartPlan must continue to be developed or refined based on the generation of any newer version of the ACRM. This work includes many tasks such as refinement of the C++ classes that represent the models, representing more project information based on the new classes, translating the new project information between other applications, and so on. Also, more and other types of real-world examples should be used in StartPlan to do the similar tests for the newly implemented models, and the results should be continually collected to support further development of the models.

StartPlan should be expanded to produce other construction document templates in a similar manner. Along with this work, other efforts will also need to be made such as developing a complete and more comprehensive expert system based on the current one, and creating more useful viewers to present more types of documents, etc. StartPlan should also be enhanced to handle multiple projects. Such projects should also be able to either share a pool of resources or a group of contractors, or different resources or contractors for different projects.
## List of References


Appendix I

Attributes And Methods of The ACRM Objects

Object:

Attributes:
- ID;
- Name;
- Description;
- ClassName;

Methods:
- SetID;
- setName;
- setDescription;
- GetID;
- GetName;
- GetDescription;
- GetClassName;
- Write;
- Read;

Activity: Parent Class: Object

Attributes:
- ActivityCode;
- Duration;
- Participant;
- Project;
- PictureFileName;
- PredArray;
- SuccArray;
- Resources;

Methods:
- SetCode;
- SetDuration;
- SetPredecessor;
- SetSuccessor;
- SetResource;
- SetResUse;
- SetParticipant;
- SetProduct;
- SetPictureFileName;
Linker:

Attributes:
- LinkActy;
- Lag;
- Type {SS, FS, SF, FF} type;

Methods:
- SetLinkActy;
- QueryLinkActy;
- SetLag;
- QueryLag;
- SetType;
- QueryType;
- WriteToMSProjectASCII;

Parent class: Object

Product:

Methods:
- WriteToMSProjectASCII;

FloorPrototype:

Attributes:
- Size;
- Shape;

Methods:
- SetSize;
- QuerySize;
- SetShape;

Parent class: Product
Floor: 

Attributes: 
FloorType;

Methods: 
SetFloorType; 
QueryFloorType; 
WriteToMSProjectASCII;

HighriseBuilding: 

Attributes: 
FloorTypeList; 
floors; 
NumberOfFloors; 
FoundationType;

Methods: 
SetFloorType; 
QueryFloorType; 
SetFloors; 
QueryFloors; 
SetNumberOfFloors; 
QueryNumberOfFloors; 
AddFloorType; 
QueryFloorType; 
SetFloor; 
QueryFloor; 
SetFoundationType; 
QueryFoundation; 
WriteToMSProjectASCII;

Plan: 

Attributes: 
ActyList;

Methods: 
SetPlanActyList; 
QueryActyList; 
GetNmbOfActys; 
AddActy(& acty); 
QueryActy; 
ModifyActy; 
DeleteActy; 
SetPredecessors; 
ClearActivities;
WriteToMSProjectASCII;

Participant:  
**Parent class:** Object

**Methods:**
WriteToMSProjectASCII;

ProjectInfo:

**Attributes:**
ProjectName;
Proj_Type;
Foundation_Type;
Structure_Type;
Numb_Of_TypicalFloors;

Project:

**Parent class:** Object

**Attributes:**
projectInfo;
ResList;
ProList;
PartList;

**Methods:**
SetProjectInfo;
GetProjectInfo;
SetProjectName;
GetProjectName;
SetProj_Type;
GetProjectType;
SetFoundation_Type;
GetFoundation_Type;
SetStructure_Type;
GetStructure_Type;
SetNumb_Of_TypicalFloors;
GetNumb_Of_TypicalFloors;
SetResList;
QueryResList;
GetNmbOfReses;
SetProList;
QueryProList;
GetNmbOfPros;
SetPartList;
QueryPartList;
GetNmbOfParts;
AddRes;
QueryRes;
QueryResource;
AddPro;
QueryPro;
QueryProduct;
AddPart;
QueryPart;
QueryParticipant;
ModifyRes;
DeleteRes;
ModifyPro;
DeletePro;
ModifyPart;
DeletePart;
ClearResources;
ClearsProducts;
ClearsParticipants;

Resource: Parent class: Object
Methods:
WriteToMSProjectASCII;

ResourceUse: Parent class: Object
Attributes:
LinkRes;
Conversion;
Price;
Quantity;
ResHolder;

Methods:
SetResUse;
SetRes;
QueryLinkRes;
SetConversion;
QueryConversion;
SetPrice(price);
QueryPrice;
SetQuantity;
QueryQuantity;
WriteToMSProjectASCII;
Appendix II

C++ Classes of ACRM

class Object
{
    public:
        Object();
        Object(const Object& object);
        ~Object();
        void SetID(unsigned id);
        void SetName(string name);
        void SetDescription(string description);
        unsigned GetID() const;
        unsigned QueryID();
        string GetName() const;
        string QueryName();
        string GetDescription() const;
        string QueryDescription();
        const string GetClassName();
        BOOL IsOld();
          int operator == (const Object& object) const;
          int operator < (const Object& object) const;
    friend ostream& operator « (ostream& os, const Object& object);
    friend istream& operator »(istream& is, Object& object);
    friend ofstream& operator «(ofstream& os, const Object& object);

    protected:
        unsigned ID;
        string Name;
        string Description;
        string ClassName;
        BOOL isNew;
};

//end of class Object
class Activity: public Object
{
    public:
        // different constructors for different initializations
        Activity();
        Activity(const Activity& acty);
        ~Activity();
    void SetCode(string code);
    void SetDuration(unsigned duration);
    void SetPredecessor(Activity& predr, int type, int lag); // set a predecessor
    void SetSuccessor(Activity& succr, int type, int lag); // set a successor
    void SetResource(Resource& res, float con, float pri, float quaty);
    void SetResUse(ResourceUse& resUse);
    void SetParticipant(Participant& partcpnt);
    void SetProduct(Product& prod);
    void SetPictureFileName(char* pictureName);
    string QueryCodeO const; // get the activity code
    unsigned QueryDurationO const;
    class Linker* QueryPredecessor(int predno) const; // get one of the predecessors
    class Linker* QuerySuccessor(int succno) const; // get one of the successors
    ResourceUse* QueryResourceUse(int index) const;
    Participant* QueryParticipantO const;
    Product* QueryProductO const;
    const char* QueryPictureFileNameO const;
    LinkerList QueryPredArrayO const; // get the pred list
    LinkerList QuerySuccArrayO const;
    ResourceUseList QueryResourceUseListO const;
    ResourceUseList GetResourceUseListO();
    ResourceUseO GetResourceUseO(int index);
    unsigned int GetTotalResUsesO();
    unsigned QueryPredecessorNumberO const;
    void ClearPredArrayO; // clears all the preds in the PredArray
    void ClearSuccArrayO;
    void ClearResourceUseListO();
    void SetLinkersHolder(int nmb);
    void SetPredHolder(int* pholder);
    void SetPredTypeHolder(int* typeholder);
    void SetPredLagHolder(int* lagholder);
    void SetResHolder(int Res_holder);
    void SetResID_holder(int* resIDholder);
    void SetResCon_holder(float* conholder);
    void SetResPrice_holder(float* priceholder);
    void SetResQuntty_holder(float* qnttyholder);
    void SetProdholder(int Prod_holder);
    void SetPartHolder(int Part_holder);
    int GetLinkersHolderO();
    int* QueryPredholderO const;
    int* QueryPredTypeHolderO const;
    int* QueryPredLagHolderO const;
    int GetResHolderO();
    unsigned* QueryResID_holderO const;
protected:
    
    float* QueryResCon_Holder() const;
    float* QueryResPrice_Holder() const;
    float* QueryResQuentity_Holder() const;
    int GetProdHolder();
    int GetPartHolder();
    int operator == (const Activity& act) const;
    int operator < (const Activity& act) const;
    friend ostream& operator << (ostream& os, const Activity& act);
    friend istream& operator >> (istream& is, Activity& act);
    friend ofstream& operator << (ofstream& os, const Activity* act);
    void WriteToMSProjectASCII(ofstream& out) const;

    string ActivityCode;
    unsigned Duration;
    Participant* participant;
    Product* product;
    char PictureFileName[MAXPATH];
    void InitPictureFileName();
    int linkersHolder; //temporarily holds the number of the linkers
    int* Pred_Holder; //temporarily holds predes
    int* *PredType_Holder, *PredLag_Holder; //holds pred types and lags
    unsigned* resID_Holder;
    float* *resCon_Holder, *resPrice_Holder, *resQuentity_Holder;
    int ProdHolder, PartHolder, ResHolder;
    LinkerList PredArray; //list of predecessors by pointers to list of Linkers
    LinkerList SuccArray; //list of successors by pointers to list of Linkers
    ResourceUseList resources;

};

//end of class Activity

class Linker: public Object
{
public:
    
    Linker():Object();
    Linker(const class Activity* linkAc, int tyPe, int lag):Object();
    ~Linker();
    //sets predecessor or successor with the lag, type
    void SetLinkActy(const Activity& predr, int tYpe, int lag);
    const Activity* QueryLinkActy() const; //retrieves the predecessor or successor
    void SetLag(int lag); //sets lag
    int QueryLag() const; //gets lag
    int GetLag(); //gets lag
    void SetType(int anytype); //sets type
    int QueryType() const; //gets type
    int GetType();
    void WriteToMSProjectASCII(ofstream& os) const;
    int operator<(const Linker& linker) const;
    int operator==(const Linker& linker) const;
    friend ofstream& operator<<(ostream& os, const Linker& link);

private:
    
    const Activity* LinkActy; //points to an activity that could be pred. or successor
    int Lag; //holds link lag
class Product: public Object
{
    public:
        // different constructors for different initializations
        Product(): Object();
        Product(const Product& prod): Object();
        ~Product();
        int operator == (const Product& prod) const;
        int operator < (const Product& prod) const;

        friend ostream& operator << (ostream& os, const Product& prod);
        friend istream& operator >> (istream& is, Product& prod);
        friend ofstream& operator << (ofstream& os, const Product& prod);
        void WriteToMSProjectASCII(ofstream& out) const;

    protected:
};

class FloorPrototype: public Product
{
    public:
        FloorPrototype(): Product();
        FloorPrototype(FloorPrototype& flrPrototype): Product();
        ~FloorPrototype();
        void SetSize(float size);
        float GetSizeO const;
        float QuerySizeO;
        void SetShape(string shape);
        string GetShapeO const;
        string QueryShapeO;
        int operator == (const FloorPrototype& flrType) const;
        int operator < (const FloorPrototype& flrType) const;

        friend ostream& operator << (ostream& os, const FloorPrototype& floorType);
        friend istream& operator >> (istream& is, FloorPrototype& floorType);
        friend ofstream& operator << (ofstream& os, const FloorPrototype& floorType);
        void WriteToMSProjectASCII(ofstream& out) const;

    protected:
        float Size;
        string Shape;
};

enum Type {SS, FS, SF, FF} type; // holds link types
// end of class Linker
class Floor: public Product
{
    public:
    Floor(): Product();
    Floor(Floor& flr): Product();
    ~Floor();
    void SetFloorType(FloorPrototype* type);
    FloorPrototype* GetFloorType() const;
    FloorPrototype* QueryFloorType();
    int operator == (const Floor& floor) const;
    int operator < (const Floor& floor) const;
    friend ostream& operator << (ostream& os, const Floor& floor);
    friend istream& operator >> (istream& is, Floor& floor);
    friend ofstream& operator << (ofstream& os, const Floor& floor);
    void WriteToMSProjectASCII(ofstream& out) const;

    protected:
    FloorPrototype* FloorType;
    // end of class Floor

class HighriseBuilding: public Product
{
    public:
    HighriseBuilding(): Product(), FloorTypeList(3,0,1), floors(5,0,1);
    HighriseBuilding(HighriseBuilding& High-rise):
        Product(), FloorTypeList(3,0,1), floors(5,0,1);
    ~HighriseBuilding();
    void SetFloorTypeList(FloorPrototypeList& flrtypelist);
    const FloorPrototypeList* GetFloorTypeList() const;
    FloorPrototypeList* QueryFloorTypeList();
    void SetFloors(Floors& flrs);
    const Floors* GetFloors() const;
    Floors* QueryFloors();
    void SetNumberOfFloors(unsigned Numb);
    unsigned GetNumberOfFloors() const;
    unsigned QueryNumberOfFloors();
    void AddFloorType(FloorPrototype& flr_type);
    FloorPrototype* GetFloorType(unsigned int index) const;
    FloorPrototype* QueryFloorType(unsigned int index);
    void SetFloor(Floor& floor);
    Floor* GetFloor(unsigned floorID) const;
    Floor* QueryFloor(unsigned floorID);
    void SetFoundationType(string* fdtn_type);
    string* GetFoundationType() const;
    string* QueryFoundation();
    friend ostream& operator << (ostream& os, const HighriseBuilding& highrise);
protected:

friend istream& operator >> (istream& is, HighriseBuilding& highrise);
friend ofstream& operator << (ofstream& os, const HighriseBuilding& highrise);
void WriteToMSProjectASCII(ofstream& out) const;

FloorPrototypeList FloorPrototypeList,
Floors floors;
unsigned NumberOfFloors;
string* FoundationType;

}; // end of class HighriseBuilding

class Plan: public Object
{
   public:
      Plan():Object(), ActyList(30,0,30);
      Plan(const Plan& plan):Object(), ActyList(30,0,30);
      ~Plan();
      void SetPlanActyList(ActivityList* Actylist);
      const ActivityList* GetActyList() const;
      ActivityList* QueryActyList();
      unsigned int GetNmbOfActys();
      void AddActy(Activity& acty);
      const class Activity* GetActy(unsigned int index) const;
      class Activity* QueryActy(unsigned int index);
      void ModifyActy(Activity& acty, unsigned int index);
      void DeleteActy(unsigned int index);
      void SetPredecessors(unsigned int SuccrIndex, int totalprednumb, int* predNumbArray, int* predTypeArray, int* predLagArray);
      void ClearActivities();
      int operator == (const Plan& pln) const;
      int operator < (const Plan& pln) const;
      friend ostream& operator << (ostream& os, const Plan& pi);
      friend istream& operator >> (istream& is, Plan& pi);
      friend ofstream& operator << (ofstream& os, const Plan& pin);
      void WriteToMSProjectASCII(ofstream& os);

   protected:
      ActivityList ActyList;

}; // end of class Plan

class Participant: public Object
{
   public:
      // different constructors for different initializations
      Participant():Object();
      Participant(const Participant& prtcpnt):Object();
      ~Participant();
      int operator == (const Participant& prtcpnt) const;
      int operator < (const Participant& prtcpnt) const;
friend ostream& operator << (ostream& os, const Participant& prtcpnt);
friend istream& operator >> (istream& is, Participant& prtcpnt);
friend ofstream& operator << (ofstream& os, const Participant& prtcpnt);

protected:

}; // end of class Participant

class ProjectInfo
{
    public:

        string ProjectName;
        string Proj_Type;
        string Foundation_Type;
        string Structure_Type;
        unsigned Numb_Of_TypicalFloors;

}; // end of class ProjectInfo

class Project: public Object
{
    public:

        Project(); Object(); ResList(10,0,1), ProList(10,0,1), PartList(10,0,1);
        Project(const Project& proj)
            : Object(), ResList(10,0,1), ProList(10,0,1), PartList(10,0,1);
        ~Project();
        void SetProjectInfo(struct ProjectInfo projinfo);
        const ProjectInfo GetProjectInfo() const;
        void SetProjectName(string projnm);
        const string GetProjectName() const;
        void SetProj_Type(string projtype);
        const string GetProj_Type() const;
        void SetFoundation_Type(string foundtype);
        const string GetFoundation_Type() const;
        void SetStructure_Type(string strcttype);
        const string GetStructure_Type() const;
        void SetNumb_Of_TypicalFloors(unsigned flrs);
        const unsigned GetNumb_Of_TypicalFloors() const;
        void SetResList(ResourceList* Reslist);
        const ResourceList* GetResList() const;
        ResourceList* QueryResList();
        unsigned int GetNmbOfReses();
        void SetProList(ProductList* Prolist);
        const ProductList* GetProList() const;
        ProductList* QueryProList();
        unsigned int GetNmbOfPros();

}; // end of class Project
class Resource: public Object
{
public:
    Resource(): Object();
    Resource(const Resource& res): Object();
    ~Resource();
    int operator==(const Resource& res) const;
    int operator<(const Resource& res) const;
};
friend ostream& operator << (ostream& os, const Resource& res);
friend istream& operator >> (istream& is, Resource& res);
friend ofstream& operator << (ofstream& os, const Resource& res);
protected:
void WriteToMSProjectASCII(ofstream& out) const;
}; // end of class Resource

class ResourceUse: public Object
{
public:
    ResourceUse():Object();
    ResourceUse(const class Resource* linkRes, float conv, float price,
                float quantity, int resholder):Object();
    ~ResourceUse();
    void SetResUse(Resource& lr, float c, float p, float q);
    void SetRes(Resource& res);
    const Resource* QueryLinkRes() const;
    void SetConversion(float conv);
    float QueryConversion() const;
    float GetConversion();
    void SetPrice(float price);
    float QueryPrice() const;
    float GetPrice();
    void SetQuantity(float quatity);
    float QueryQuantity() const;
    float GetQuantity();
    void SetResHolder(int res_holder);
    int GetResHolder();
    void WriteToMSProjectASCII(ofstream& os) const;
    int operator<(const ResourceUse& linker) const;
    int operator==(const ResourceUse& linker) const;

friend ostream& operator << (ostream& os, const ResourceUse& ru);
friend istream& operator >> (istream& is, ResourceUse& ru);
friend ofstream& operator << (ofstream& os, const ResourceUse& link);
protected:
    const Resource* LinkRes;
    float Conversion;
    float Price;
    float Quantity;
    int ResHolder;
}; // end of class ResourceUse