INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 5" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600
AN INFORMATION MODEL
FOR MANAGING DESIGN CHANGES
IN A COLLABORATIVE MULTI-DISCIPLINARY
DESIGN ENVIRONMENT

Ahmed H.M. Mokhtar

A Thesis
in
School for Building
Faculty of Engineering and Computer Science

Presented in Partial Fulfilment of the Requirements
for the Degree of
Ph.D. in Engineering (Building)
at Concordia University
Montreal, Quebec, Canada

© Ahmed H.M. Mokhtar, 1998
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
Abstract

AN INFORMATION MODEL FOR MANAGING DESIGN CHANGES IN A COLLABORATIVE MULTI-DISCIPLINARY DESIGN ENVIRONMENT

Ahmed H.M. Mokhtar

The presence of incompatibility errors in construction technical documents is a major problem for the construction industry. An analysis of the production process of these documents reveals that managing design changes constitutes a main source for incompatibilities. More specifically, failure to propagate design changes among the design team is a principal cause of problems. The large amount of design data that is generated within a multi-disciplinary design environment makes this task very complex, especially when considering the fact that the involved design disciplines are, in most cases, separated both spatially and educationally. This thesis presents the development of a model that uses information technology to address the problem. The core concept of the model is a central database that functions as a repository of active building components. Each building component in this database not only carries its design data, but is also capable of recognizing the design disciplines that are affected by any change in these data and automatically send them messages. The active building components are able to perform this task because they are equipped with the necessary linking knowledge. Such knowledge is necessary to propagate the effect of a design change by one discipline on
other disciplines involved in the design of the same building. The linking knowledge is acquired from the designers and is implemented in the form of rules. A management database is also developed as part of the model's central database. This management database makes the model easily adaptable to any building configuration, an essential requirement due to the uniqueness of every building project. It also provides the model with the capability for not only tracking past design changes but also planning and scheduling future design changes as well. The model has been successfully implemented on a client-server network environment and validated using both hypothetical and real design cases.
Acknowledgements

Thanks to GOD for every thing I was able to achieve and for everything I tried but failed to achieve.

I also wish to thank all the individuals whose assistance and encouragement were instrumental in the realization of this work. I am deeply grateful to my supervisors, Dr. Claude Bédard and Dr. Paul Fazio for their guidance, constructive advice, thoughtful recommendations, and financial support. I would like also to thank the examination committee for their valuable criticism.

The continuous reassuring and assistance from both Dr. Osama Moselhi and Dr. Sabah Alkass were inspiring in continuing this work successfully. The help of Mr. Sylvain Belanger was very facilitating during the computer implementation part of this research.

I am very grateful to all my colleagues specially, Dr. Adel Abdou, Dr. Khalid El-Rayes, and Mohammad Al-Housein for their useful comments, helpful advice, and solid support.

Very special thanks to my parents for their unreplaceable giving and consistent motivation.

My deepest appreciation for my wife and for my children for their love, patience, and encouragement.
# Table of Contents

List of Figures ........................................................................................................ IX  
Nomenclatures ....................................................................................................... XIV  
Abbreviations ........................................................................................................ XIV  
Legend for User Interface Diagrams ...................................................................... XV

## CHAPTER ONE

**INTRODUCTION**

1.1 Motivation and Problem Definition ........................................................................ 1  
1.1.1 Background ....................................................................................................... 1  
1.1.2 Overview of the Detailed Design Stage ............................................................ 2  
1.1.3 Diagnosis of the Main Problems ...................................................................... 8  
1.1.3.1 Managing Design Changes ........................................................................ 8  
1.1.3.2 Communication Media .............................................................................. 9  
1.1.3.3 Source of Final Documents ...................................................................... 9  
1.1.4 Conclusion ....................................................................................................... 10  
1.2 Research Scope and Objectives ........................................................................... 10  
1.3 Outline of the Thesis .......................................................................................... 12

## CHAPTER TWO

**INFORMATION TECHNOLOGY IN DESIGN: A LITERATURE REVIEW**

2.1 Introduction ......................................................................................................... 13  
2.2 Collaborative Design Approaches ......................................................................... 14  
2.2.1 Agent-Based Federated Architecture Approach ............................................. 14  
2.2.2 Knowledge-based Approach ......................................................................... 15  
2.2.3 Constraints Management Approach ............................................................... 18  
2.2.4 Circle Approach ........................................................................................... 19  
2.2.5 Discussion ..................................................................................................... 19  
2.3 Product Data Modeling ......................................................................................... 21  
2.3.1 On Definitions and Implementation ............................................................... 21  
2.3.2 Analysis of Some Building Product Data Models ........................................... 24  
2.3.2.1 The AEC Building Reference Model .................................................... 25  
2.3.2.2 The COMBINE IDM Model .................................................................. 26  
2.3.2.3 The RATAS Model ............................................................................... 26  
2.3.2.4 Other Models ....................................................................................... 27
CHAPTER THREE

PROPOSED INFORMATION MODEL:
A CONCEPTUAL PERSPECTIVE

3.1 Objectives and Model Characteristics ......................................................34

3.2 The Model Architecture .............................................................................35
  3.2.1 Basic Parts .........................................................................................35
  3.2.2 The Concept of Active Building Components .......................................39
  3.2.3 Propagating Current Design Changes ................................................43
  3.2.4 Tracking Past Design Changes ............................................................45
  3.2.5 Planning and Scheduling Future Design Changes ...............................48

3.3 Discussing the Model Advantages and Limitations .....................................52

3.4 Summary ....................................................................................................55

CHAPTER FOUR

THE CENTRAL PROJECT-DATABASE:
A DETAILED PERSPECTIVE

4.1 Introduction ..................................................................................................56

4.2 Flexible repository of active building components .......................................57
  4.2.1 Providing Flexibility ............................................................................57
  4.2.2 Making the Building Components Active ...........................................62

4.3 Assembly and Configuration .......................................................................64
  4.3.1 Assembly of the Building-Components-Database of a New Project .......65
    4.3.1.1 The General-Database ..................................................................66
    4.3.1.2 The Assembly Process .................................................................70
  4.3.2 Configuring the Project-Database of an Existing Project .....................72
    4.3.2.1 Configuring Building Components .............................................72
    4.3.2.2 Configuring Prebuilt Rules ..........................................................74
    4.3.2.3 Configuring the Project Menu .....................................................74

4.4 Summary ....................................................................................................79
CHAPTER FIVE

MANAGING DESIGN CHANGES:
A DETAILED PERSPECTIVE

5.1 Introduction
5.2 Propagating Current Design Changes
  5.2.1 Objectives
  5.2.2 Methodology
    5.2.2.1 Acquire the linking knowledge
    5.2.2.2 React to Design Changes
    5.2.2.3 Deliver Design Change Messages
  5.2.3 User Interface
5.3 Tracking Past Design Changes
  5.3.1 Objectives
  5.3.2 Methodology
  5.3.3 User Interface
5.4 Planning and Scheduling Future Design Changes
  5.4.1 Objectives
  5.4.2 Methodology
    5.4.2.1 Collect Data
    5.4.2.2 Organize Changes
    5.4.2.3 Schedule Changes
    5.4.2.4 Calculate Costs and Durations
  5.4.3 User Interface
5.5 Summary

CHAPTER SIX

VALIDATION OF THE MODEL

6.1 Introduction
6.2 Computing Environment
6.3 Validation
  6.3.1 Approach
  6.3.2 First Level of Validation
  6.3.3 Second Level of Validation
    6.3.3.1 Assembly and Population of the Project-Database
    6.3.3.2 Propagating Current Design Changes
    6.3.3.3 Tracking Past Design Changes
    6.3.3.4 Planning and Scheduling of Future Changes
  6.3.4 Third Level of Validation
6.4 Summary and Conclusions

Ahmed Mokhtar, Ph.D Thesis .... VI
CHAPTER SEVEN

SUMMARY, CONTRIBUTIONS, AND RECOMMENDATIONS

7.1 Summary ........................................................................................................... 159
7.2 Contributions ..................................................................................................... 161
7.3 Recommendations for future work ................................................................. 164

REFERENCES ........................................................................................................ 166

APPENDIX A

DATABASE STRUCTURE IN MS ACCESS

A.1 Management-database ...................................................................................... 173
    A.1.1 Conceptual Data-Structure ....................................................................... 173
    A.1.2 Structure of Data-Tables ......................................................................... 174

A.2 General-Database ............................................................................................ 177
    A.2.1 Conceptual Data-Structure ....................................................................... 177
    A.2.2 Structure of Data-Tables ......................................................................... 178

APPENDIX B

SUBROUTINES FOR PLANNING AND SCHEDULING DESIGN CHANGES

B.1 Subroutine to generate the various change-paths. ........................................... 179
    B.1.1 Initializations ............................................................................................ 179
    B.1.2 Generate the first change-path .................................................................. 179
    B.1.3 Generate the other change-paths .............................................................. 181

B.2 Subroutine to generate the logical sequence of design changes .................. 182
    B.2.1 Initializations ............................................................................................ 184
    B.2.2 Process the initial design change .............................................................. 184
    B.2.3 Process the rest of the design changes in the change-path ...................... 184
    B.2.4 Generate the dummy design changes ...................................................... 186

APPENDIX C

LISTINGS OF LEVEL5 OBJECT MODULES

C.1 Configuration Module ...................................................................................... 190
C.2 Data Manipulation Module ................................................................. 208
C.3 Message Module .................................................................................. 226
C.4 Tracking Module .................................................................................. 231
C.5 Virtual Change Module ....................................................................... 239
C.6 Planning and Scheduling Module ........................................................ 250
List of Figures

Figure 1-1 Production process of construction technical documents during detailed design stage. ................................................................. 4

Figure 2-1 Organization of design agents in federation architecture. (Khedro et al. 1994, redrawn) ........................................................................... 16

Figure 3-1 Architecture of proposed information model. .................................................. 36

Figure 3-2 Interface for data manipulation module. .............................................................. 38

Figure 3-3 Configuration of a building component using configuration module. ................. 40

Figure 3-4 Assembly of building components in building-components-database from building systems available in general-database ................................................................. 40

Figure 3-5 Concept of active building components. ............................................................. 41

Figure 3-6 Message received by envelope designer informing of design change made by architect ................................................................. 44

Figure 3-7 Message received by structural engineer informing of design change made by envelope designer. ................................................................. 45

Figure 3-8 Interface for tracking module. ............................................................................ 47

Figure 3-9 Results of request made by design manager to track design changes made to a building component ................................................................. 47

Figure 3-10 Results of request made by design manager to track automated messages that are sent because of a design change. ................................................................. 48

Figure 3-11 Scenario for interrelated design changes ............................................................. 49

Figure 3-12 Alternate change-paths for scenario in Figure 3-11 ............................................. 51

Figure 3-13 Bar chart for alternate change-path (a) in Figure 3-12 ........................................ 52

Figure 4-1 Sample of data in data-table “Window” which describes instances of the building component “Window”. ................................................................. 59

Figure 4-2 Configuration of the building component “Circular_Window” using configuration module. ........................................................................... 61

Figure 4-3 Sample of data in Data-Table 4-3 that describes data-structure of the
building components “Window” and “Circular_Window”........................62

Figure 4-4 Combination of relational database and object-oriented technologies........63

Figure 4-5 Sample of data in Data-Table 4-4 that describes data-structure of variety
of building components........................................................................68

Figure 4-6 Sample of data in Data-Table 4-5 that defines building components that
belong to a building system......................................................................68

Figure 4-7 Sample of data in Data-Table 4-6 that records current projects............69

Figure 4-8 Sample of data in Data-Table 4-7 that describes some prebuilt rules........70

Figure 4-9 Sample of data in Data-Table 4-8 that describes configuration of project
menu..........................................................................................................75

Figure 4-10 Configuration of a prebuilt rule using configuration module.................76

Figure 4-11 Project menu as it functions in data manipulation module....................77

Figure 4-12 Interface to configure project menu to allow access to available
building components................................................................................77

Figure 4-13 Diagram to illustrate user interface in configuration module................78

Figure 5-1 Sample of data in Data-Table 5-1 that describes some prebuilt rules........84

Figure 5-2 Sample of data in Data-Table 5-2 that describes some dynamically built
rules........................................................................................................84

Figure 5-3 Sample of warning message about design change that results in major
design modifications.............................................................................85

Figure 5-4 Acquisition of dynamically built rule using relation-acquiring sub-
module...................................................................................................87

Figure 5-5 Sample of data in Data-Table 5-3 that describes some message elements..90

Figure 5-6 Diagram to illustrate user interface in data manipulation module...........96

Figure 5-7 Diagram to illustrate user interface in message module.......................97

Figure 5-8 Sample of data in Data-Table 5-4 that describes some design changes
and enables their tracking.......................................................................99
Figure 5-9 Diagram to illustrate user interface in tracking module. ..........................102
Figure 5-10 Scenario for interrelated design changes. ...........................................105
Figure 5-11 Starting a new case in virtual change module. .....................................107
Figure 5-12 Data provided by designer for proposed design change .......................108
Figure 5-13 Response to virtual change message in virtual change module. ..........110
Figure 5-14 Data provided by designer for proposed design change .......................111
Figure 5-15 Sample of data in Data-Table 5-6 that defines sequence of changes. ....111
Figure 5-16 Sample of data in Data-Table 5-7 which is collected from designers through virtual change module .................................................................112
Figure 5-17 Sample of data in Data-Table 5-8 that defines alternative change-paths. 113
Figure 5-18 Alternate change-paths for scenario in Figure 5-10 ...............................114
Figure 5-19 Bar chart for alternate change-path (b) in Figure 5-18 .........................116
Figure 5-20 Partial data from Data-Table 5-7 that is used for calculating costs and durations. ..................................................................................................118
Figure 5-21 Diagram to illustrate user interface in virtual change module. ..........121
Figure 5-22 Interface to show historical data about certain design change ..........122
Figure 5-23 Diagram to illustrate user interface in planning and scheduling module.122
Figure 5-24 Interface to show detailed data of a design change .............................123
Figure 5-25 Interface to show results of all alternative change-paths so as to enable comparing them .................................................................123
Figure 6-1 Sample of statements recorded by “history” tool in LEVEL5 OBJECT. ....127
Figure 6-2 Assembly of building components in building-components-database from building systems available in general-database ..........................131
Figure 6-3 Adding building component from general-database to building- component-database .................................................................132
Figure 6-4 Configuration of a building component using configuration module. ....132
Figure 6-5 Input data to structural design translated to prebuilt rule...........................................133
Figure 6-6 Input data to structural design translated to prebuilt rule...........................................133
Figure 6-7 Input data to HVAC design translated to prebuilt rule..................................................134
Figure 6-8 Input data to envelope design translated to prebuilt rule. ...........................................134
Figure 6-9 Dynamically built rule acquired from structural designer defining one of
the conditions used to design depth of Open_Web_Steel_Joist.............................135
Figure 6-10 Message received by structural designer informing of design change
made by architect............................................................................................................137
Figure 6-11 Message received by HVAC designer informing of design change
made by architect............................................................................................................137
Figure 6-12 Message received by envelop designer informing of design change
made by architect............................................................................................................138
Figure 6-13 Message received by structural designer informing of design change
made by envelope designer. .................................................................138
Figure 6-14 Message received by structural designer informing of design change
made by HVAC designer..............................................................................................139
Figure 6-15 Results of request made by design manager to track design changes
made to a building component.........................................................................................140
Figure 6-16 Results of request made by design manager to track automated
messages that are sent because of certain design change..............................140
Figure 6-17 Scenario for extended design changes with several alternatives..........143
Figure 6-18 Bar chart and data for alternate change-path (1) for the scenario in
Figure 6-17 ....................................................................................................................144
Figure 6-19 Bar chart and data for alternate change-path (2) for the scenario in
Figure 6-17 ....................................................................................................................144
Figure 6-20 Bar chart and data for alternate change-path (3) for the scenario in
Figure 6-17 ....................................................................................................................145
Figure 6-21 Bar chart and data for alternate change-path (4) for the scenario in
Figure 6-17 ....................................................................................................................145
Figure 6-22 Bar chart and data for alternate change-path (5) for the scenario in Figure 6-17 ............................................................. 146

Figure 6-23 Bar chart and data for alternate change-path (6) for the scenario in Figure 6-17 ............................................................. 146

Figure 6-24 Comparison among various alternate change-paths for the scenario in Figure 6-17 ............................................................. 147

Figure 6-25 Plan view of an exterior column and the surrounding components as designed (Fazio 1990 reproduced) ............................................. 154

Figure 6-26 Plan view of the exterior column and the surrounding components as constructed (Fazio 1990 reproduced) ............................................. 155

Figure 6-27 Prebuilt rule to capture link between construction tolerance and design of envelope and encasement ............................................. 156

Figure 6-28 Dynamically built rule to capture link between width of the gypsum board encasement and fireproofing thickness ............................. 157

Figure 6-29 Automated message to inform architect about change in thickness of fireproofing ................................................................. 158

Figure A-1 Conceptual data-structure for management-database presented in EXPRESS-G ............................................................................. 173

Figure A-2 Conceptual data-structure for general-database presented in EXPRESS-G ............................................................................. 177

Figure B-1 Result of applying steps Sub1-D till Sub1-H on example given in Figure 5-10 ............................................................................. 181

Figure B-2 Result of applying steps Sub1-I and Sub1-J on example given in Figure 5-10 ............................................................................. 182

Figure B-3 Matrix to represent start nodes and end nodes for change-path in Figure 5-18b ............................................................................. 183

Figure B-4 Flowchart for sub-routine developed to generate various change-paths ................................................................. 188

Figure B-5 Flow chart for sub-routine developed to define sequence of performing design changes so as to eliminate redundancy ............................................. 189
Nomenclatures

\[ n \] Total number of design changes in a change path.
\[ x \] A design change.
\[ C_r \] Expected impact of a change-path on the building construction costs.
\[ C_x \] Value of attribute "DIFFERENCE_INBUILD_COST" in Data-Table 5-7.
\[ D_r \] Design cost related to a change-path.
\[ h \] Cost of man hour in the design firm.
\[ M_x \] Value of attribute "REQUIRED_MAN_HOURS" in Data-Table 5-7.
\[ R_r \] Level of recommendation for a change-path.
\[ R_x \] Value of attribute "RECOMMENDATION_LEVEL" in Data-Table 5-7.
\[ T_r \] Additional design time required for a change-path.
\[ T_x \] Value of attribute "DURATION" in Data-Table 5-7.

Abbreviations

AEC Architecture/Engineering/Construction.
AI Artificial Intelligence.
AAACE American Association of Cost Engineers.
ASCE American Society of Civil Engineers.
CADD Computer Aided Design and Drafting.
CIFE Centre for Integrated Facility Engineering.
CMU Concrete Masonary Units.
CPM Critical Path Method.
DBMS Data Base Management System.
DICE Distributed and Integrated environment for Computer aided Engineering.
DIS Draft International Standards.
DXF Drawing interchange file.
EDM Engineering Data Model.
HVAC Heating, Ventilation, Air Conditioning.
IAI Industrial Alliance for Interoperability.
IBDE Integrated Building Design Environment.
IFC Industry Foundation Class.
IGES The Initial Graphics Exchange Specifications.
ISO International Organization for Standardization.
IT Information Technology.
MIT Massachusetts Institute of Technology.
MS MicroSoft.
NIST National Institute of Standards and Technology.
OO Object-Oriented.
SEED Software Environment to support Early phase in building Design.
SQL Structured Query Language.
STEP STandards for the Exchange of Product data.
## Legend for User Interface Diagrams

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define tracking constraints</td>
<td>New screen</td>
</tr>
<tr>
<td>Show building component under modification, its instance, and its modified attribute.</td>
<td>Note</td>
</tr>
<tr>
<td><img src="image" alt="Decision node symbol" /></td>
<td>Decision node</td>
</tr>
<tr>
<td>Display related messages</td>
<td>Model output</td>
</tr>
<tr>
<td>Input the data of the design change</td>
<td>Link to a floating screen</td>
</tr>
<tr>
<td>Choose a project</td>
<td>Link to a screen that replaces the current screen</td>
</tr>
<tr>
<td>Choose a related attribute</td>
<td>User input in the current screen</td>
</tr>
</tbody>
</table>
CHAPTER ONE
INTRODUCTION

1.1 MOTIVATION AND PROBLEM DEFINITION

1.1.1 BACKGROUND

Construction teams erect buildings on the basis of construction technical documents. Incompatibility errors in these documents result in change orders, contractual disputes, cost overruns, time delays, compromise to quality, frustration and client dissatisfaction. Common types of errors are:

- *Inconsistency in design information.* For example, the location of a specific column is not identical when comparing the architectural and the structural drawings.

- *Mismatch between connected components.* For example, HVAC ducts dimensions, which are given in the mechanical drawings, do not match the dimensions of the related pass-holes in the structural beams which are given in the structural drawings.

- *Components malfunction.* For example, electric supply in a room is designed to serve a classroom activity while architectural drawings indicate that the same room has been re-designed as a computer lab.

Unfortunately, such errors have become very common in the construction industry (Tilley and Barton 1997). Some contractors even depend on these errors for generating their
profit margin. Owners, on the other hand, are either discouraged to invest in buildings or try to compensate the effect of these problems in the pricing of services or products realized within the constructed facility. This may negatively affect not only the building industry, but also eventually the national economy.

Construction technical documents are produced by the design team during the detailed design stage of a project life cycle. These documents are the result of collective efforts from specialists who belong to various design disciplines. These specialists, who are usually geographically separated, make autonomous design decisions with respect to their own discipline. These decisions, nevertheless, are inter-dependent and need to be coordinated so as to maintain compatibility among the various systems and components in the building under design. The coordination process which occurs during the detailed design stage is very complicated. This is due to the generation of an enormous amount of design data from the various specialized disciplines. Unfortunately, with escalating complexity in buildings, the increasing specialization in design teams, and the tightening of financial and time resources available to designers, the coordination task is rapidly increasing in complexity. The building industry is in urgent need for research work that addresses the problem and contributes to finding practical solutions. This research is a step towards satisfying that need.

1.1.2 Overview of the Detailed Design Stage

During the detailed design stage, abstract concepts presented in the preliminary design of
a building are transferred into technical documents. These documents should contain sufficient design information to enable the erection of the building. The production process of these documents, which appear in the form of working drawings and specifications, is an elaborate one. It requires members of the design team to satisfy many constraints such as owner demands, code requirements and limitations of the project budget. In addition, it requires from them to ensure the compatibility of design information presented in the technical documents. This research is primarily concerned with the common failure of designers to satisfy this last demand. An analytical overview of the production process of construction technical documents is therefore presented. The overview concentrates on current techniques for cross-disciplinary coordination of design information, hence revealing the main problems that cause the process to fail.

Figure 1-1 illustrates the results of the analysis. The figure shows that the production process of technical documents starts by extracting the data available in preliminary design documents. From these documents, the architecture design team develops the main features of the building in the form of drawings (Point 1, Figure 1-1). These main features usually include geometric details of the structural elements, walls, partitions, doors, windows, stairs, elevators, and the function of the various spaces. Several copies are made of these drawings and distributed to other disciplines in the design team (Point 2, Figure 1-1).
Figure 1-1 Production process of construction technical documents during detailed design stage.
Next, each discipline (including architecture) starts to make design decisions for the detailing of a specific building system (Points 3, Figure 1-1). These decisions aim to satisfy various design requirements and are restricted by codes and regulations. More important to our study, these decisions depend also on design decisions that are made by other disciplines. For example, to design an HVAC component in a certain room, designer “A” searches for the data needed as input to the HVAC design process, e.g. space function, dimensions, heat produced from lighting fixtures, and thermal resistance of external walls. Such input data is actually output data of other disciplines. Both the space function and dimensions are data that is defined by the architect; the lighting fixtures are defined by the illumination engineer; the thermal resistance of the external walls is defined by the envelope designer. Designer A tries to find this data in the drawings that are provided by the architect. Most probably, only the data for the space function and the space dimensions are available in these drawings. To get the rest, designer A needs to communicate with the illumination engineer and the envelope designer (Point 4, Figure 1-1). Once all the input data is obtained, designer A can make design decisions on the HVAC components of the room. As long as all elements of input data remain unchanged, the HVAC component will remain compatible with the other components in the room.

Unfortunately, design changes - inevitably and continuously - occur for many reasons. These include the need to satisfy new or modified requirements by the owner, to reduce construction or maintenance costs, to rectify design mistakes, or to improve the design.
For example, the architect may change the function of the space from an office to a computer lab. Such a change is usually performed by the architecture discipline through renaming the function of the space on the floor plan drawing. Once that “simple” change is performed, the HVAC component - designed by A - instantly becomes incompatible with the architectural design. It remains as such till designer A becomes aware of the design change and modifies the HVAC component using modified input data.

The main problem therefore is how to make designers of any discipline aware of all changes that are performed by other disciplines and that affect their own design. This is a difficult problem to solve because of the lack, in our judgment, of the knowledge that “links” the output of one design discipline to the input of another design discipline. This lack of “linking” knowledge is primarily due to the education and training of professionals in the building industry. It is difficult, for example, for an architect to know the input data for an HVAC engineer. The architect might not know that space function is a critical input for the HVAC system design. Consequently, s/he may change the space function without informing anyone because it seems unnecessary to do so. As a result, the HVAC engineer has no way to recognize that a design change affecting HVAC has been made.

From this simple example, one can foresee the difficulty which design teams face when dealing with a project that has thousands of design data that are mostly interrelated. Not surprisingly, some professionals describe design changes in a multi-disciplinary environment as a “mess”. Nonetheless, techniques are used by professionals in order to
cope with the problem (Point 5, Figure 1-1). Some of these techniques are:

a) *Regular coordination meetings where designers from various disciplines group together for the purpose of finding out about the important changes that each discipline has done.* Each discipline usually reviews the drawings of the other disciplines to look for modified data. Drawings from different disciplines are sometimes put together for comparison. Recently, with development in computer technology, such meetings become less frequent. Reference drawing files are increasingly used to show the latest drawing version of various disciplines instantly. However, this technique of reviewing the documents of other disciplines has a serious deficiency. It depends on designers to discover the changes that are relevant to their own design, mainly through looking at drawings. As a result, many design changes which do not clearly appear on drawings (e.g. thermal resistance of a wall or a small change in a dimension) are usually missed. Since large projects may include hundreds of drawings, it is impractical to review such a large amount of data regularly.

b) *The designer who makes a change informs every other designer in the design team.* This technique may work fine in small projects, yet in larger or more complex projects, it becomes impractical. Every designer has to issue too many design change memos and will also receive as many. Most of these memos are irrelevant to many designers and soon lose credibility. Designers become discouraged to spend their time writing and reviewing them.
c) A common technique is to use check lists to verify the compatibility of various components in a project. The main problem with this technique is that the design modifications that are required when incompatibility errors are discovered may generate in turn other incompatibilities. The technique cannot continuously monitor the effect of design changes. In addition, the check lists are rarely customized so as to adapt to every building project. Many incompatibilities are therefore skipped by this technique.

The production process of construction technical documents (points 3, 4, and 5 in Figure 1-1) is cyclically repeated till deadline for submission is reached. Each discipline then finalizes its own documents (Point 6, Figure 1-1) and all final documents are gathered from the various disciplines and given to the general contractor (Point 7, Figure 1-1).

1.1.3 Diagnosis of the Main Problems

From the above analysis of the process, three major problems can be diagnosed as the causes for incompatibility errors.

1.1.3.1 Managing Design Changes

Design changes continuously occur within all disciplines. The current coordination techniques provide weak mechanisms to manage design changes and to accommodate their effect on the design information across participating disciplines. More specifically, there is a lack in the ability of designers for propagating and communicating design changes effectively. Others also support this finding. Hegazy and Khalifa (1996)
conducted a survey among 12 leading design firms in Canada and showed that disseminating project information and administering design changes are two important aspects of multi-disciplinary coordination which cause problems in building design. Cornick (1990) states also that "the problems caused in modern buildings seem much more due to deficiencies in managing communication during the design process than to merely technological factors".

1.1.3.2 Communication Media

Drawings are used as the principal media to exchange and document design information. Drawings are excellent for describing shape, geometry, proportions and other visual characteristics of the design. Yet, drawings are very poor to show other characteristics such as heat resistance, load capacity, and price. Such non-visual characteristics are as essential as the visual ones for interdisciplinary exchange of information. Problems with the drawing-centered environment are also reported by others (Voeller 1996).

1.1.3.3 Source of Final Documents

The final construction technical documents are collected from a variety of sources. This can lead to inconsistency in information when the same building element is described in different ways in several documents. For example, the location of a specific column is not identical when comparing the architectural and the structural drawings.

Recently, CADD software developers have tackled the second and the third problems. Software such as Bentley Microstation™, ArchT™ and Pro-Reflex™ provide a data-
centric environment and central single model for the designed building. Technical
drawings are then generated from such models, ensuring consistency among various
drawings. Nevertheless, other types of incompatibilities (as given in section 1.1.1 on page
1) continue to exist. These incompatibilities are attributed to the failure to address the
first diagnosed problem: managing design changes.

1.1.4 Conclusion

The analytical overview of the detailed design stage shows that a main challenge in
maintaining compatible design information resides in the accommodation of design
changes. More specifically, propagation of design changes to the affected disciplines
constitutes the critical point. The various techniques and software that are currently used
by designers to coordinate their design information appear inadequate. There is a clear
need for an approach that targets this important problem.

1.2 Research Scope and Objectives

The main objective of this research is to assist the building industry in eliminating
incompatibility errors that commonly exist in design information of building projects, a
pervasive problem that is very costly to the industry. The research focuses on the design
changes during the detailed design stage as these changes remain the main cause of
incompatibility errors.

The research aims to develop a computer-based model that uses Information Technology
to provide the building industry with a practical solution to the problem. The developed model however is not limited to the propagation of current design changes, rather it is extended to enable the tracking of past design changes and the planning and scheduling of future ones.

The research also aims to implement the model in a client-server computing environment which is similar to that regularly used by building design professionals. Furthermore, it aims to validate not only that the model is able to provide capability to manage design changes but also that the model can actually improve current practices with these capabilities.

The research however does not intend to provide a fully functional information system which should include complete software, hardware specifications, operating procedures, training process for operators, and full collection of the needed data.

The information model developed in this research does not intend to cover all types of information necessary for the design of a building. Codes, regulations, and design constraints are among the design information that are excluded. The model focuses on information needed for design changes. The model is also intended for sequential type of project delivery systems. It is not intended for other types of delivery systems such as fast track system. Designers who will use the model are expected to collaborate using client-server networks and therefore they should be located in the same building or nearby buildings. Other types of networks such as wide area networks, intranets, or internet are

Ahmed Mokhtar, Ph.D Thesis .....11
beyond the scope of this model.

1.3 OUTLINE OF THE THESIS

This thesis documents the development of an information system model for managing design changes in a collaborative multi-disciplinary building design environment. Chapter one presents the motivation for this research and defines the specific problem. Chapter two reviews research elsewhere in the use of information technology for the design of constructed facilities. Chapter three introduces a conceptual perspective for the proposed information model and discusses the advantages and limitations of this model. Chapters four and five present the detailed development of the main components of the model and show how the model concepts are realized with these components. Chapter four focuses on the central project-database and how it is developed to adapt easily to various building configurations. Chapter five provides the detailed methodologies that make the model capable of managing design changes. Chapter six explains the validation process for the model. Chapter seven summarizes the research and highlights its contributions to the construction industry. It also provides recommendations for future research work.
2.1 Introduction

In chapter one, three major problems are diagnosed as the causes for incompatibility errors in construction technical documents. These problems are: managing design changes, communication media, and source of final documents (see section 1.1.3 on page 8). As a first step to find an appropriate solution that overcomes these three problems, this chapter reviews available research work that is concerned with the use of information technology in the design of constructed facilities. This area of research is relatively new and increasingly captures the attention of researchers in many parts of the world. This attention can be attributed to the growing familiarity of the construction industry with CADD (Computer Aided Design and Drafting) systems. It can be also attributed to the potential of CADD systems to address many problems in such a scattered, multidisciplinary, and information-rich industry. This review is organized into three sections:

- The first section is concerned with approaches that use computer technology to enhance collaborative design environments. Each of the reviewed approaches is evaluated against its ability to provide solution to the previously diagnosed problems.

- The second section focuses on efforts to establish data-structures (data models) that can support design data in a central database. Such a central database is considered to be necessary for ensuring a single source of design information. The literature review
in this group aims to investigate the availability of standards or at least sufficiently advanced models that can be utilized in this research to develop a central database for building data.

- The third section surveys research projects that deal with managing the building design process, an area to which this research belongs, so as to review utilized concepts and techniques.

2.2 COLLABORATIVE DESIGN APPROACHES

The use of computer technology to enhance collaborative design environments is the focus of several research works. The following summarizes the main approaches found in literature, along with discussions regarding their ability to solve the three main problems diagnosed in section 1.1.3.

2.2.1 AGENT-BASED FEDERATED ARCHITECTURE APPROACH

Developed at Stanford University, it connects two or more independent engineering software that need to share information about the current status of a design (Khedro et al. 1994, 1995 and 1996, McGraw et al. 1996, Jones and Riley 1995, Khedro 1994, Huyn et al. 1993, Cutkosky et al. 1993). The system architecture includes several design agents (Figure 2-1). Each design agent is composed of a human designer and a software agent (design software plus communication capabilities). Every software agent interacts with other software agents through facilitators that coordinate the exchange of design information among the software agents. When a designer finishes a design task, s/he
sends the design information to the facilitator that s/he is connected to. The facilitator is able to recognize other interested users in each element of that design. It groups the design elements and forwards them to the relevant users. These users receive the information, check it, evaluate its effect on their design, and start negotiation with each other in real time to solve conflicts in design. In view of the diagnosed problems in section 1.1.3, this approach has a fundamental drawback which is the lack of a single source of final design information. However, when it comes to managing design changes, the approach has the advantage of reducing the amount of information received by every user as only the relevant information is sent to the various users. This is an important feature that has been incorporated in the information model proposed in this thesis. The federated architecture approach nevertheless depends on the designers to recognize design changes. This leaves a lot of room for error and is similar to current practice (section 1.1.2). The approach also requires a significant deployment of computing resources and communication links across separate locations, a requirement that would set such an approach beyond the reach of most building designers.

2.2.2 Knowledge-based Approach

The Knowledge-based approach has been used to help integrate software that are used in engineering design. Fenves et al. (1990, 1994) introduced a prototype system called the Integrated Building Design Environment (IBDE) at Carnegie Mellon University. The system intends to vertically integrate the various software that are used in the various stages of the life cycle of a constructed facility. Its demonstration and test domain,
however, is limited to architectural planning, structural design and construction planning of high-rise office buildings. All the involved software use knowledge-bases that support their functions. Project data is maintained in a project database whereas communication between the different software relies on a message blackboard.

Figure 2-1 Organization of design agents in federation architecture. (Khedro et al. 1994, redrawn)

Another system called DICE (Distributed and Integrated environment for Computer aided Engineering) has been under development at the MIT (Massachusetts Institute of Technology) (Sriram et al. 1989, 1990, and 1994, Sriram and Logcher 1993, Ahmed et al.)
1992, Wong and Sriram 1993). The focus of the project is on preliminary and detailed structural design. DICE consists of a blackboard, knowledge modules, and control mechanism. DICE blackboard is implemented on a central object-oriented database. This central database does not contain all the product data but is limited to the data used for communication among the knowledge modules.

A more recent model is SEED (Software Environment to support Early phase in building Design) (Flemming and Woodbury 1995) also from Carnegie Mellon University. SEED aims at providing computational support for the early design phase. It addresses specifically architectural programming, schematic layout design and the generation of a fully three dimensional configuration of physical building components like structure and enclosure. SEED contains an object database to store and retrieve different design versions and design alternatives. Designers can specify and modify dynamically and interactively design requirements. Given explicit requirements, SEED can be asked not only to propagate design changes after some interactive modifications by the designer, but also to show other designers feasible ways of solving a design problem using case-based reasoning.

In the knowledge-based approach, collaboration among software is achieved by the use of precoded coordination knowledge. The coordination knowledge is based on the various software meta-knowledge (knowledge about the composition and the computational process of the software) and hence it needs to be updated to reflect any upgrading in the involved software. In order to manage design changes, the knowledge needs to
accommodate various possible relations that may exist among different building components, a requirement that is almost impossible to fulfill due to the unique nature of every building project. Such a nature makes these relationships vary from one project to the other. When this is combined with different composition of design teams in different projects, it seems that this approach is not suitable for the complex environment of building design.

2.2.3 Constraints Management Approach

This approach has been proposed as a method to share design knowledge without linking together design software (El-Bibany and Paulson 1994, El-Bibany 1992 and 1996, Bowen and Bahler 1993). A constraint is taken as a relation which states what should be true about one or more design entities. Managing these constraints is achieved by creating and maintaining data-dependency structures that reflect any changes in the design knowledge and identify specific knowledge items that cause conflicts. The users have the ability to create relations dynamically between entities. Yet, the entities themselves are fixed in the system. We view the ability of this approach to dynamically create relationships between entities as essential to adhere to the fact that every building project is unique in these relationships. However, it is also important to be able to easily change the building entities to reflect the requirements of a specific project. In case of a design change that violates any of the constraints which were previously built, the system sends warning messages to involved parties informing of source of violation. This is also an important feature that is used by the model proposed in this thesis for managing design changes.
The constraints management approach however requires each discipline to provide all necessary equations and functions that define and constrain the design of every building component in order to built the data dependency structures. This requirement results in duplicating most of the equations and the constraints that are already used by the design software. The process as such becomes very complicated when considering the number of disciplines and the number of components involved in the design of a building.

2.2.4 CIRCLE APPROACH

Aiming to integrate some AEC (Architecture/Engineering/Construction) software that are used by participants in a project, the circle approach (Fischer and Kunz 1995) has each software linked to exactly one predecessor and one successor software. Thus, changes made to the design are propagated automatically to all other software. It is expected that participants will have private copies of all software on the circle for a particular project. The circle approach does not use any form of centralized repository for design data. As such, it cannot satisfy the need for having a single source for the final design documents which can insure the consistency of building information. It propagates new versions of design, but depends on users to recognize any change in the design that might affect them which is similar to current practice.

2.2.5 DISCUSSION

The approaches introduced above for the use of computers in collaborative design environment are not mutually exclusive but rather, share common features. All
approaches use data as the means to exchange information among participants. This supports the notion that drawings should not be - in the computer age - the media for recording design information. Both the agent-based approach and the circle approach do not use any form of centralized repository for design data, nor for data coordination knowledge. As such, these approaches cannot insure the consistency of building information. Both the agent-based approach and the circle approach propagate new versions of design, but depend on users to recognize any change in the design that might affect them. The knowledge-based approach coordinates design information using detailed coordination knowledge linked with the used software. As discussed above, this approach seems impractical for the complexity of building design environment. The constraints management approach requires duplicating most of the equations and the constraints that are already used by the design software, which makes its use time-consuming and redundant.

None of the reported approaches appears suitable to provide remedy for all the problems diagnosed in section 1.1.3. However, some reported capabilities are useful to this research such as the capability of sending information to the interested designers only and the dynamic capturing of coordination knowledge from designers instead of exclusively using precoded knowledge. The information model proposed in this thesis include these capabilities.
2.3 **PRODUCT DATA MODELING**

Product data modeling may also be viewed as one of the approaches to facilitate design collaboration. Because of the wide interest by researchers to develop product data models and the need to have a central database to provide a single source for design information, product data modeling is reviewed in more detail. The purpose is to investigate the availability of standards or at least sufficiently advanced models that can be utilized by this research in developing a central database for building data. The review starts by defining product data models and overviews its successful implementation in some research areas. A description of main product data models follows in order to explore the existence of any common or necessary features for developing these models. Finally, discussion and highlight of finding is presented.

2.3.1 **ON DEFINITIONS AND IMPLEMENTATION**

A model is a simplified representation of a part of the real world. It captures some, but not all, of the characteristics of that part (Lave and March 1975). For example, an architectural perspective of a building, which can be referred to as a graphical model, represents visual relationships between the various elements of the building envelope. Meanwhile, a group of equations represents the transfer of heat through that envelope and is called a mathematical model. One type of model that has emerged with the development of computers is Data Models which describe a part of the reality in form of its data elements. Data models are divided into three types (Elmasri and Navathe 1994): conceptual models, which tell what kind of information is used to describe some aspect of
reality and how such information is internally structured (Björk 1989); implementation models, which are translations of conceptual models for a specific database management system; physical models, which represent how the data is placed on a computer hardware system.

Product Data Models are a distinct category of the conceptual type of data models. These are tailored to represent data of engineering products. There is no single definition that is agreed upon for a product data model. The COMBINE project (Augenbroe 1993) defines it as "a complete conceptual description of a product, capable of structuring all the information necessary for the design, manufacture, and use of the product." Björk and Penttilä (1989) define it as a "conceptual structure specifying what kind of information is used to describe a building (product) and how such information is structured." Tolman, Kuiper, and Luiten (1990) define it as an "information model of a product, describing the 'reality' of a product in its different life cycle stages." Eastman (1992a) defines the product data model as "the database model supporting the design, fabrication, operation and other uses of some type of product." From these definitions, common denominators can be extracted: product data models are conceptual models; these represent all the product data; this representation spans throughout the product life cycle. This conclusion can be confirmed by the International Organization for Standardization (ISO) definition of product data (ISO 1989) as "The totality of data elements required to completely define a product; this includes geometry, topology, relationship, tolerance, attributes, and features necessary to completely define a component, part or an assembly of parts for the
purpose of design, analysis, manufacture, test and inspection." Product data models are, therefore, perceived to be good tools for both the exchange and integration of product information.

Because of the potential of product data models to facilitate the integration and exchange of product information, researchers in various parts of the world have adopted them to represent the Building as a product. The AEC Building Systems Model in the USA (Turner 1990), the RATAS model in Finland (Björk 1989), and the COMBINE IDM model in Europe (Dubois and Parand 1993) are examples of that utilization.

Other research work has been conducted so as to create an environment and a set of concepts that link building data models to various aspects of the building delivery process. The available research was in areas like:

- Early design cost control (Tsou 1992),
- Computer-aided architectural design (Turner 1992),
- Conformance with regulations (DeWaard 1992),
- Evaluating building performance (Augenbroe 1992),
- Maintenance information management (Svensson 1993),
- Renovation design (Vahala 1995),
- Design of building envelope (Rivard et al. 1995),
- Design of precast concrete facades (Karhu 1997), and
- Structural synthesis and evaluation (Bakkeren and Tolman 1995, Enseleit et al.
1995).

Another notable effort in using database to record building design data is through the International Alliance of Interoperability (IAI). The IAI is an alliance of groups in the building industry that aims to integrate the AEC/FM (Architecture, Engineering, Construction / Facility Management) industry by specifying Industry Foundation Classes (IFCs) as a universal language to improve the communication, productivity, delivery time, cost, and quality throughout the design, construction, operation and maintenance life cycle of buildings (IAI 1996). IFCs constitute a library of commonly defined objects that create "intelligent" project data such as the properties, behavior, and graphical representation of building components (Herold 1997).

Similar effort is done by ISO which is in the process of establishing a universal standard for the representation and exchange of product data (ISO 1989). This standard is usually referred to as STEP (STandards for the Exchange of Product data), formally ISO 10303 (Froese 1996). Its objective is to provide a mechanism that is capable of describing product data throughout the life of a product independent of any particular computer system. A comparison among current standards to communicate data [e.g. DXF (1995) and IGES (Reed et al. 1990)] and those suggested by IAI and STEP is presented by Arnold and Teicholz (1996).

2.3.2 ANALYSIS OF SOME BUILDING PRODUCT DATA MODELS

The following is a review of some of the major building product data models developed
in research. The purpose is to understand the various concepts and approaches that categorize and organize the data that exists in a building.

2.3.2.1 The AEC Building Reference Model

This is a high level conceptual product data model for AEC products that concentrates on Buildings (Turner 1990). It defines a building project as a unique object made of a building and a site. A building is an object with one or more properties such as type, primary activity, and secondary activity. It decomposes into a number of building systems. Building system examples are structural, electrical, circulation, plumbing, heating, and lighting. A site is also an object with one or more properties such as humidity ratio, temperature, and view. It decomposes into site systems which can be electrical, fresh water and disposal, electric, and gas.

The AEC Building Reference Model adopts a hierarchy of four levels: system, system component, system component port, system component port joint. A system exists to satisfy a human or natural need; an example is the building structural system. A system is a collection of system components. A system component such as a wall may belong to various systems, such as the enclosure system, the acoustical system, and the structural system. The wall component has different functions in each of these systems. System component ports are used to connect components in the same systems or in different systems. A door is a port that connects a room component to a corridor component. Each system component port is joined with another system component port by a system component port joint. The AEC model provides description for only the spatial system,
the enclosure system, and the structural system of a building.

2.3.2.2 The COMBINE IDM Model

COMBINE IDM stands for "CComputer Models for the Building INdustry in Europe - Integrated Data Modeling" (Dubois 1992, Dubois and Parand 1993). It is part of the COMBINE project which has a primary focus on the energy aspect of buildings. The root object in this model is the Construction Project which may contain one or more Buildings. The Building is seen as an assembly of systems. The systems are: spatial system, fabric system, technical system, functional system, and external environment system. The overall space is divided into a set of zones, and each zone has some homogeneous behavior. A zone is created from the specific view of an expert. Due to the focus of the model (energy analysis), the fabric system main element is the building enclosure, and the HVAC (Heating, Ventilation, Air Conditioning) system is the only one available as a technical system.

2.3.2.3 The RATAS Model

RATAS is the basic model for the building products in Finland. It stands for "Computer Aided Design of Buildings" in the Finish language (Björk 1989). It aims to cover the design, production and maintenance stages of the building project. Concepts such as objects, attributes, relationships are used in the model. Objects are collections of data about a "thing", whether physical or abstract. To each object, a number of attributes can be associated that describe the properties of that object. Attribute types in RATAS are:
numeric, text, pictures, codes, and lists which make it capable of containing all kinds of
data describing a building. Five functional levels for objects are distinguished in RATAS:
built, system, subsystem, part, and detail. There is only one building level object per
building. Its attributes are about the site, climate, type of building, construction cost, total
size etc. Systems can be spaces, load-bearing, mechanical, electrical, or heating.
Subsystems (or Groups) divide systems into functional parts such as floor or hospital
ward. Parts constitute the vast majority of objects and are usually tangible entities such as
building elements or technical devices. Their typical attributes are location and shape.
Detail level includes data of objects that are subdivisions of parts like the different parts
of a window. Usually these data will be included in general access databases. The model
uses two types of relations: 'Part of' and 'Connected to'. The 'Part of' relation links objects
from different levels while the 'Connected to' is more typical in part and detail levels, and
usually connects objects at the same level.

2.3.2.4 Other Models

Another product data model is the Construction Project Reference Model (Rezgi and
Depras 1995) where the building is described according to five complementary systems:
structure, work, space, technical and separation systems. The purpose of this model is to
enable the generation of construction documents from a single source of information. A
multi-dimensional model of buildings (3P Model) which integrates the process, product,
and participants is proposed by Bédard and Rivard (1995). It classifies buildings into
structure, envelope, services, and interior systems.
2.3.3 Discussion

The review of the available models shows that classification of building data varies considerably from one model to another. It depends mainly on the purpose of each model. The common feature in the models is the division of a building into group of systems. However, these divisions vary from one model to the other. Each model concentrates on a specific stage on the building delivery process and details a limited number of systems to fit its purposes. Therefore, no standards are currently available to follow nor is there any model that is sufficiently developed to be used in this research. Large groups of researchers have developed some of the models, yet the models still have limited capabilities. This shows the extreme difficulty to represent buildings as data models due to the unique nature of building projects. Therefore, we believe that any data model that represents building projects mainly needs to be flexible and oriented to be suitable for the project at hand. The use of IFCs might achieve this need, however, little information is publicly available now in order to consider their use in this research.

2.4 Design Process Management

Research in the use of information technology for managing the design of a building project, which this research belongs to, is still uncommon. However with growing interest in both the use of information technology in collaborative design and the use of product data modeling in the construction industry, we believe that interest in using information system for managing design will also grow. The following reviews available research in three issues that are important to the design process management.
2.4.1 Design Changes

Design changes, which are the main issue addressed by this research, has not yet received much attention in the literature. A project by Krishnamurthy and Law (1995a, 1995b, and 1996 and Krishnamurthy 1996) proposes a three-layered model of versions, assemblies, and configuration. A version is a description of a primitive entity, an assembly describes a composite entity that belongs to a single discipline, while a configuration describes the overall project. The model monitors independent design activities by systematically tracking components descriptions in individual disciplines. It uses the concept of equivalent operations to compute changes. The advantages of this approach is its ability to compute changes between different versions of the design. Yet, it remains for the designer who performs the changes to know whether or not others are affected by the change. Also, designers themselves need to recognize any inconsistencies by reading drawings of the latest design version.

The EDM (Engineering Data Model) which has been developed by Eastman intends to support integrity management among various intelligent design software packages that are connected to the model (Eastman et al. 1995, Eastman 1992b, 1994). The model includes constraints that check the status of rules and specific knowledge embedded in the software. When a design change occurs to an instance, all design instances that use that data of the changed instance are flagged for rechecking. However, no information is given to the user on what data element has changed and why a data element is flagged for rechecking.
2.4.2 Design Information

Management of design information was also the focus of some research work. Platt (1996) focused on civil engineering projects and tried to manage their design information through process modeling rather than product modeling. Process modeling has information on how a product is transformed from its initial state to the final deliverables. Hence, the project information recorded on computer contains not only data about the project component but also about the process used to record this data. Rezgui has developed an approach to tackle the problem of integrity and consistency in the production of construction documents (Rezgi and Depras 1995). It is based on building a construction project reference model and a document reference model. Both models are linked with an association model that indexes building components to documentary items. The research focuses on how to extract construction documents from a project reference model and tries to ensure the integrity of these documents. It does not focus on how to ensure the integrity of the design information itself. The International Organization for Standardization (ISO) is also in the process of establishing a standard layering structure to be used in building design to facilitate the transfer and management of information (Björk et al. 1997).

Exchanging design information among designers has been investigated by Vries and Somers (1995) who developed a process model for that purpose. They suggested that a protocol to exchange information should ensure that both senders and receivers of information should interpret it the same way. Also all the required and correct
information should be exchanged. These suggestions are important and are incorporated in the information model proposed in this thesis. The use of the internet to exchange and manage design information has been also investigated. Guenster (1996) discussed the benefits, costs, and lessons from using the internet with a design workgroup. Goodman and Chinowsky (1996) discussed the requirements for managing design information through an inter-disciplinary team when the internet is utilized. Rojas (1997) described a process for developing a web-centric system that enables cooperative engineering.

2.4.3 DESIGN RATIONALE

Several research projects have addressed the issue of acquiring the rationale behind a design decision. Peña-Mora and others (1995) studied the representation, use and communication of design rationale for conflict mitigation. They developed a model that is capable of representing design knowledge in terms of the reasoning process used by the designer. The model also provides computer support for capturing designer’s reasoning process. This capability appears useful to easily extract required knowledge from designers. The proposed information model in this thesis therefore includes similar capabilities. De la Garza and Alcantara (1997) used parametric dependency networks to represent design rationale. These networks can show how one particular design decision affects other decisions. The use of this type of network requires significant effort and training from the design team. In general, the aim of research work in capturing design rationale is to define the motives for a design decision. The acquired knowledge is different from that needed to propagate design changes. The latter requires knowledge
that defines the consequences of a design decision.

2.5 SUMMARY OF FINDINGS

From the literature review, it is evident that the building industry is still at the exploration stage on how to use information technology in design and design management. Several approaches have been tested for collaboration in a multi-disciplinary environment. Each approach is oriented to solve a specific problem. None of them is capable to offer a solution for all problems diagnosed in section 1.1.3. Building product data models are successfully used to solve several research problems. However, different researchers present very different compositions for these data model. No standard is available nor is there sufficiently developed model that can be used. A flexible data model that can be configured for a specific building project appears to be the practical answer. Literature review on managing design process reveals that fewer research work studied how to deal with design changes. As shown in section 2.4.1, these researches study the problem from a viewpoint which differs from this thesis. Krishnamurthy and Law (1995a, 1995b, and 1996 and Krishnamurthy 1996) focus on comparing various design versions and define embedded differences whereas Eastman (1992b, 1994, 1995) is concerned with how to propagate design changes among intelligent design software. Throughout the literature review, some capabilities appeared useful and will be incorporated in developing the proposed information model. These capabilities are:

- Dynamic capturing of knowledge from designers during the design development.
Such capturing capability should be supported by appropriate computer interface.

- Acquiring and storing some information about the design process itself and not limiting the information used in the model to that describing the building components.
- Propagated design information needs to reach only the interested designers not all the designers.
- Exchange of design information has to be performed in a standard fashion that can be understood clearly.

The following chapters present the proposed information model in both conceptual and detailed perspectives.
CHAPTER THREE

PROPOSED INFORMATION MODEL:
A CONCEPTUAL PERSPECTIVE

3.1 OBJECTIVES AND MODEL CHARACTERISTICS

The analysis for the production process of construction technical documents diagnosed three main problems that result in the inclusion of incompatibility errors. The analysis also revealed that design changes remain as the principal challenge. The information model developed in this research aims to overcome this challenge. The model also needs to provide solution to the other diagnosed problems (see section 1.1.3 on page 8). The information model therefore needs to have the following characteristics:

1. It is capable of managing design changes. This capability is seen not to be limited to help propagation of design changes but it is also expected to cover the need for tracking past design changes and for providing the capacity to plan and schedule future design changes.

2. It uses data rather than drawings as the main media for storing and communicating design information.

3. It has a single source for design information that carries a unique description for every component in a building project.

Through the following sections, the architecture of the model is described in a conceptual perspective so as to reveal the main ideas rapidly (see also Mokhtar, Bédard, and Fazio
1998). The description is supported by simple examples that demonstrate the capabilities of the model. A detailed perspective is presented in the next two chapters.

3.2 **The Model Architecture**

3.2.1 **Basic Parts**

The heart of the model, as shown in Figure 3-1, is a central *Project-Database*. Unlike other models, this project-database is uniquely composed of two parts, the *Building-Components-Database* and the *Management-Database*. The building-components-database functions as a repository of all the design data that is necessary to describe every building component in a certain project. Such data is detailed enough to produce construction technical documents that are suitable for erecting the building. The management-database contains data that is required to manage the various functions of the information model. The role of this database will be incrementally revealed throughout the rest of the thesis. To generate the data that populates the building-components-database, each *Designer* is linked to the central project-database by a group of modules called *Designer System*. The *Design Manager*, who is responsible for managing the production of the technical documents, is also linked to the central project-database through a group of modules called *Design Manager System*. 

*Ahmed Mokhtar, Ph.D Thesis .... 35*
Figure 3-1 Architecture of proposed information model.
When the building components database is fully populated, construction technical documents can be produced with the use of the *Document Generation Module* which can create customizable documents about the project through its link to the project database (Mokhtar and Bédard 1994). This module, shown in dotted lines in Figure 3-1, resides beyond the scope of this thesis.

The detailed design stage starts with the architectural team translating ideas in preliminary design documents into elementary building components such as walls, columns, doors, and windows. Most of the data that describes these components is textual in nature. Therefore, its value can be keyed in the building-components-database. Some of the data is geometric in nature, such as the dimensions, and is usually documented through a computer-aided drafting tool. The function of that tool is to acquire, through its interface, the geometric data and to translate it into alphanumeric characters. The model currently does not include such a tool and the designers therefore are expected to directly key in the geometric data directly in textual format.

Designers interact with design data in the database in order to view, modify, or add values. This interaction is materialized through the *Data Manipulation Module*. As Figure 3-2 illustrates, a designer can - through this module - create a new instance or manipulate the value of any attribute of any instance of any building component. The type of manipulation (view and/or modify) should depend on the authority that a designer has over a specific building component. The data saved in the building-components-database reflects decisions taken by the design team members.
Yet, to enable the designers to record data about the various building components of a certain project, the building-components-database needs to have a suitable "data-structure" (or "product data model"). This data-structure should echo the type of building components that exist in the project. Because buildings are unique products, every building project contains different combination of building components. Hence, every building project needs a building-components-database with a unique data-structure. The building-components-database therefore should be flexible so as to adapt to the requirement of every project. Such flexibility is realized in the model through the use of the management-database. More details are given in chapter four and also in Mokhtar,

---

1 All linear measurements are in mm units.
Bédard and Fazio (1997b). The data-structure of the building-components-database can be manipulated through the *Configuration Module* which is part of the design manager system. Figure 3-3 illustrates part of that module where an attribute (vapor barrier type) is added to the building component “CMU_backed_brick_veneer_wall” as part of creating that component in the building-components-database. Such manipulation, which is carried out by the design manager, is facilitated through the use of a *General-Database*. This general-database functions as a source of data-structure for a large variety of building components. Figure 3-4 illustrates the use of the general-database to retrieve the data-structure of the building components that belong to an HVAC system into the building-components-database. The configuration of the data-structure for the building-components-database needs to be performed before beginning the development of the detailed design. Yet, it can also be updated during that development.

### 3.2.2 THE CONCEPT OF ACTIVE BUILDING COMPONENTS

As recognized before from analyzing the production process of construction technical document, design changes constitute a major source of incompatibility errors. The crux of the design change problem is the failure on the design team’s part to propagate design changes. As found from the analysis of the detailed design (section 1.1.2), this failure is due to the lack of the “linking” knowledge that is necessary to perform this task. Also, it is due to the large amount of data that is generated and needs to be monitored and communicated during the development of detailed design.
Figure 3-3 Configuration of a building component using configuration module.

Figure 3-4 Assembly of building components in building-components-database from building systems available in general-database.

Ahmed Mokhtar, Ph.D Thesis .... 40
The model introduces a concept that directly tackles the crux of the problem. It assigns the responsibility of propagating design changes to the building components themselves. As such, when a designer modifies any attribute that describes a property of a building component, the component itself becomes responsible to find out who in the design team is affected by that design change. The component becomes also responsible to send messages to the affected designers. These messages inform them about the design change and advise each one of them on how such a change may have an impact on his/her design (Figure 3-5).

![Diagram of Building Components]

Figure 3-5 Concept of active building components.

For example, if the envelope designer decides to increase the height of a wall, the wall itself notifies the structural engineer of this decision and advises him/her that such a decision may affect the design of the girder that carries this wall. By depending on the
building components themselves rather than the designers to propagate design changes, we are eliminating the possibility that the envelope designer, who initiates the change, may forget to propagate the change, think that it is too insignificant to mention, or not be aware that another discipline is affected. We also compensate for the ability of the structural designer to discover such a change when reviewing the wall detail drawings.

On the basis of the above example where only one element of data has changed, it is evident that the proposed concept of having building components propagate design changes becomes highly advantageous in real cases where hundreds of such changes occur daily within the thousands of data elements that exist while developing the detailed design of a building project.

With such concept, the building-components-database is no longer a mere repository of idle data, but rather a repository of "active" building components that help the design team coordinate the building design information.

To make building components active in conducting their assigned task, they are connected with methods (groups of procedures). These methods make a building component active as soon as a design change occurs to any of its attributes. The components are also equipped with the necessary "linking" knowledge which identifies the disciplines that are affected by a specific design change and how they are affected.

The model is developed so as to acquire this knowledge in the form of rules. These rules,
like the building components, are customizable so as to reflect the requirements of a specific building project. The model considers two types of rules, prebuilt rules and dynamically built rules. Prebuilt rules are acquired before the beginning of the detailed design stage while configuring the model. The dynamically built rules are acquired “on the fly” during the detailed design stage (i.e. during the use of the model). For example, a prebuilt rule would reflect the fact that a change in the height of a wall affects the structural system design. An example of a dynamically built rule would be the design decision to make the height of a wall in a one-story building equals the clear height plus 1000mm.

3.2.3 **Propagating Current Design Changes**

When a design change occurs to a building component attribute, the component becomes active and checks all relevant prebuilt and dynamically built rules. If any of these rules applies to the design change, the component automatically sends messages to the affected designers.

Figure 3-6 gives an example of a message that is automatically sent to the envelope designer as a result of a design change made by the architect. The designer receives the message through the *Message Module* (see Figure 3-1 on page 36). This module allows the designer to review and modify the rules that cause messages to be sent, thus providing the model with the flexibility to adopt new relations that designers may develop among building components. When the envelope designer in turn takes a corrective action,
another message (Figure 3-7) is automatically sent to any other affected designer.

With the concept of active building components, we are now in a position to propose a solution to the main source of problems when design changes occur. Naturally, we expect that the involved designers will assume the responsibility to take the necessary actions as soon as they are informed of design changes.

![Figure 3-6 Message received by envelope designer informing of design change made by architect.](image)
3.2.4 TRACKING PAST DESIGN CHANGES

For design managers to manage the production of construction technical documents at the detailed design stage better, the history of design changes in a project represents an important source of information. The tracking of such history enables design managers to analyze the development of the production process. They can then evaluate the performance of the various designers in a project (e.g. designers who make poor design decisions which need to be later modified, costing the design team time and money), and can also determine responsibilities for design incompatibilities. With this in mind, the model is developed to provide the following capabilities through its Tracking Module.
(see Figure 3-1 on page 36):

- Retrieval of design changes that are:
  
  made by a discipline (e.g. structure),
  
  made to a building component (e.g. concrete beams),
  
  made to an instance of a building component (e.g. concrete beam001),
  
  made to a single attribute of a specific instance of a building component (e.g. depth of concrete beam001).

  The retrieved design changes can also be limited over a period of time.

- The model can show the design manager all automated messages that have been sent by a specific building component in response to the design changes affecting it. The model also indicates which discipline has received each message and whether designers of this discipline have read the message and taken corrective action.

Figure 3-8 shows the design manager interface to the tracking module while Figure 3-9 gives the result of a request to monitor the design of a building component. Figure 3-10 shows one of the messages which were automatically sent to a member of the design team (as in Figure 3-7) because of a specific design change. The color of the message determines whether the designer who received the message read it and whether a corrective action was taken or not.
Figure 3-8 Interface for tracking module.

Figure 3-9 Results of request made by design manager to track design changes made to a building component.
3.2.5 PLANNING AND SCHEDULING FUTURE DESIGN CHANGES

Many design changes impact only a limited number of building components. Yet, some changes result in a sequence of design changes to related building components. For example, in a specific building project, the owner needs to change the function of a certain space from an executive office to a computer lab. Such a change (indicated as A1 in Figure 3-11) which is to be performed by the architect leads to several other changes that need to be performed by other disciplines. The lighting designer needs to modify the type of luminaires that are used in the space (L1). The interior designer needs to change the wall finishing (I1). The HVAC designer needs to increase the size of the supply duct

Ahmed Mokhtar, Ph.D Thesis .....48
to the space. S/he has the option to do that by increasing the width of the duct (H1) or the depth of the duct (H2).

**Figure 3-11 Scenario for interrelated design changes**

When these design changes (L1, I1, and H1 or H2) are performed, each of them will result in the need for further design changes. As the wall finishing changes (I1), the type of luminaires (L1) needs to be reevaluated and may be changed. If the width of the duct is changed (H1), then the location of the sprinkler outlet needs to be changed (F5) and the width of the hole in the structural beam, where the duct passes, needs to be changes (S1). However, if the depth of the duct is changed (H2), different changes are needed. The
height of the hole in the structural beam will be changed (S2) as well as the depth of this beam (S3). The later change will result in lowering the suspended ceiling level and reducing the clear height (A2). Due to the lowered clear height, the type of luminaires (L1) needs to be reevaluated and may be changed.

This scenario illustrates the “domino” or interrelated effect that a design change in one discipline can have on several other disciplines. For simplicity, changes that recursively affect each other in closed loop fashion are omitted. Such types of design changes represent a challenge for the design team, specially the design manager, in many aspects. Among these aspects is how to predict the interrelated design changes that result from an initial design change such as (A1). The lack of the linking knowledge that enables such prediction makes this task very difficult. Another aspect is the possibility to have more than one solution to accommodate a design change as in the case of the HVAC designer who has the alternative to follow change (H1) or change (H2). Each alternative results in different “group of design changes” (or change-path). It is difficult not only to determine all the possible change-paths to accommodate an initial design change but also to select which one of them is the best to follow. Another aspect of the challenge is the possibility that some design changes such as (L1) may be repeated several times. It is hard to know when such a change should be performed in order to avoid the repetition.

To assist the design team in facing the challenges of interrelated design changes, the capability of planning and scheduling future design changes is built in the model. A methodology has been developed and implemented for this purpose using both the
Virtual Change Module and the Planning and Scheduling Module shown in Figure 3-1 on page 36. Figure 3-12 shows the result of applying the methodology on the example given in Figure 3-11. The figure illustrates that the methodology identifies the possible change-paths and properly locates design changes so as to avoid repetitions. Figure 3-13 shows the same result as it is displayed to the design manager for one of the possible change-paths. The detailed development of the methodology is presented in chapter five and in Mokhtar, Bédard and Fazio (1997a).

![Diagram](image)

Figure 3-12 Alternate change-paths for scenario in Figure 3-11
3.3 **DISCUSSING THE MODEL ADVANTAGES AND LIMITATIONS**

The architecture of the model provides a data-centric rather than a drawing-centric environment. It also provides a single repository of the design data. As such, the model architecture solves two of the diagnosed problems (section 1.1.3) that cause incompatibility errors in construction technical document. The capability of the model to automatically propagate design changes solves the main design change problem. This capability ensures that every member of the design team is updated with the data that s/he is using as input to the design of his/her building system. The quality of the input data is absolutely essential to the quality of the design of any building system and its
compatibility with other building systems. There is no way for a good designer to provide suitable design with wrong or outdated data. However, the model does not guarantee that a design decision that is taken by a designer is the appropriate one.

Some approaches to address the design change problem help designers in recognizing these changes through providing them with the updated drawings of the other disciplines. These approaches depend on the designers to recognize the design changes through comparing the updated and old drawings. The approach taken here is to directly inform only the affected designers with textual data that clearly shows the change. As such, no room is left for failure to recognize small changes or non graphical changes.

The approaches that use knowledge-based systems to coordinate design information depends on built-in coordination knowledge that should automatically correct any incompatibility errors. It is mainly oriented to help inexperienced designers who may not be able to take correct decisions. In contrast, the approach of this model is to recognize the experiences of the involved designers. Yet, it supports them with correct and updated information that they need to make their design decisions. We see this as a more realistic approach. It uses computers in what they excel at, which is dealing with large amounts of data, and designers in what they excel at, which is creativity that is built on experience. Because of its dependence on knowledge that is provided by the developers, the knowledge-based approach will always have a limited capacity to coordinate design data (El-Bibany 1992). Such knowledge is difficult to reflect the relationships among different building components which vary from one building to the other. This model uses only
linking knowledge and does not use any induction knowledge.

The model is developed in such a fashion that provides flexibility to both the data-structure of the building components and the linking knowledge that define the relations among these components. Such dual flexibility is a great advantage to the model as it reflects the nature of building projects. However, this flexibility requires the model to be configured to suit every project. This consumes extra time from the design team. Nevertheless, this time is seen as an investment that will pay off in reducing the long time that is usually consumed in coordinating and in solving incompatibility problems.

The model capability to plan and schedule future design changes is unique and has not been addressed before to our knowledge. This capability, however, depends on collaboration among members of the design team.

The architecture of the model currently lacks a tool to accept and show graphical data. The inclusion of such a tool will greatly ease the use of the model. The problem of data security and control of access is not addressed by the model nor the problem of the legal boundaries among the various involved disciplines. The model is more oriented toward design firms that include all the needed design disciplines under one legal entity. However, with the growing use of “Partnering” in the construction industry, this limitation may be overcome. The model is developed with a vision to use client-server networks and therefore limited to be used within one building or close-by buildings. It is also oriented to sequential type of project delivery systems. It cannot function, for
example, in a fast track delivery system.

3.4 Summary

The crux of the design change problem, which is propagation of design changes, is directly tackled by the proposed information model. The model introduces the concept of active building components where each component is responsible for propagating design changes that occur to any of its attributes. The model is also developed with a unique management-database. This management-database allows the model to have flexible data-structure that is adaptable to the requirements of any building project. It also provides the model with the capability to track past design changes and plan and schedule future design changes. In this chapter, the overall architecture of the model is presented in conceptual perspective. The following two chapters detail the development of these concepts.
4.1 INTRODUCTION

The heart of the proposed information model, as shown in Figure 3-1 on page 36, is the central project-database. This project-database is composed of two parts, the management-database and the building-components-database. The management-database contains data that is required to manage the various functions of the information model. The building-components-database functions as a repository of all the design data that is necessary to describe every building component in a certain project. To enable recording this data by the design team, the building-components-database needs to have a suitable "data-structure" that echoes the nature of building components that exist in a certain project. However, the building-components-database is not a mere repository of idle data, but rather a repository of "active" building components that help the design team coordinate the building design information.

This chapter shows the used technologies and the detailed development for the project-database so as to built it as a flexible repository of active building components. The chapter also shows the process of assembling and configuring the project-database through the use of the configuration module and with the assistance of the general-database (see Figure 3-1 on page 36).
4.2 Flexible Repository of Active Building Components

The primary function of the central project-database is to be a repository of active building components. Because of this function, structuring the project-database needs to satisfy two main requirements. The first requirement is to make the database flexible enough to accept the design information of any building. The second requirement is to have the building components in this database active in managing design changes as this is the main objective of the model. This section shows how the model is developed so as to satisfy these two requirements.

4.2.1 Providing Flexibility

Buildings are unique products. A hospital contains very different components from those of a warehouse. Furthermore, a warehouse may share some components with other warehouses but not all. For example, an air-heated warehouse may contain air distribution ducts while a radiant heated warehouse will not. Therefore, the definition of a data-structure (data-model) for the building-components-database may follow either one of the following two strategies. The first strategy is to create a universal data-structure for the building-components-database that covers all possible building projects. The second strategy is to customize the data-structure of the building-components-database to suit individual building project. The first strategy should result in a reusable and therefore desirable data-structure, yet it requires the inclusion of a complete set of available and newly introduced building components. As the number of these components is virtually unlimited, this strategy rapidly becomes impractical. In this research, the second strategy
is used to define the data-structure of the building-components-database which is the flexible part of the project-database. The management-database, on the other hand, has a fixed data-structure. For reference purposes, appendix “A” displays the conceptual data-structure of the management-database. Copies of its data-tables are also used through this chapter to facilitate the explanation.

To fulfill the flexibility requirement, the Relational Database technology is utilized to structure the project-database. This technology allows dynamic and transparent manipulation of data-structure and data values through an interface program by using the Structured Query Language (SQL). With such technology, each building component (e.g. window, beam, duct) is represented as a data-table. Each data-table has a different data-structure that represents the attributes describing each component. Data-Table 4-1 shows a sample of the structure for the data-table that represents the building component “Window”. The instances of a building component (e.g. window001, window002) are stored in their own data-table. Figure 4-1 shows how these instances are stored in Data-Table 4-1.
<table>
<thead>
<tr>
<th>Window001</th>
<th>3000</th>
<th>1200</th>
<th>200</th>
<th>900</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window002</td>
<td>2500</td>
<td>1200</td>
<td>200</td>
<td>900</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4-1 Sample of data in data-table “Window” which describes instances of the building component “Window”.

The use of SQL statements enables adding, deleting and modifying data-tables that represent various building components in the building-components-database. For example, if the architect decides to use circular windows in addition to rectangular windows, s/he needs to have a data-table that contains the attribute “Radius” instead of the attributes “Width” and “Height” that are in Data-Table 4-1. Asking the design manager to create such a data-table, s/he specifies the name and the attributes of the data-table. Using SQL, the data-table “Circular_Window” is created in the building-components-database. The SQL statements for this operation include these shown in the zigzag\(^1\) frame. The first statement builds a new data-table with an initial attribute “Circular_Window_ID” that accepts alphanumeric values. The second and the third statements modifies the created data-table by adding two other attributes, “Radius” and “Frame_width” that accept numeric values.

```
CREATE TABLE [Circular_Window] [Circular_Window_ID] TEXT;
ALTER TABLE [Circular_Window] ADD [Radius] NUMERIC (5);
ALTER TABLE [Circular_Window] ADD [Frame_width] NUMERIC (5);
```

The interface program (the configuration module) creates these SQL statements

\(^1\) Text within a zigzag frame is extraction from the program code.
automatically, the design manager needs only to provide the name of the building component and its attributes (Figure 4-2). S/he does not need to know how this data is structured in the building-components-database. The created data-table (Data-Table 4-2) can now carry the data of all the circular windows in the project. The configuration module also enables the modification of the data-table at any later time to accommodate the need to describe the building component with further details.

```
| CIRCULAR_WINDOW_ID | RADIUS     | FRAME_WIDTH | SILL_HEIGHT |
```

Yet to enable the use of SQL in providing this capability, a specially developed data-table (Data-Table 4-3) in the management-database is used\(^1\). This data-table records data that describes the data-structure of the various data-tables in the building-components-database. As such, the structure of these data-tables can be tracked and manipulated. Figure 4-3 shows sample data of Data-Table 4-3. The sample shows the data that describes the data-table “Window” and the data-table “Circular_Window”.

\(^1\) Names of all data-tables that belong to management-database start with MNG.
Figure 4-2 Configuration of the building component “Circular_Window” using configuration module.
Figure 4-3 Sample of data in Data-Table 4-3 that describes data-structure of the building components “Window” and “Circular_Window”.

4.2.2 Making the Building Components Active

The second requirement for the design of the central project-database is to make the building components active in managing design changes. To fulfill this requirement, the Object-Oriented (OO) technology is adopted. In this technology, an object is a collection of related variables and procedures (Taylor 1992). Objects encapsulate methods, which are groups of procedures. When a designer modifies the design of a building component (e.g. window), the attributes of this building component (e.g. width, height) are retrieved into an OO environment (Figure 4-4). Each attribute is temporarily represented as an

1 The headings are truncated and some may have been removed in order for the figure to fit on the page.
object and consequently becomes encapsulated with a method. The method is triggered when a design change occurs to the value of the attribute. The method retrieves some rules that are related to the attribute (e.g. a 20% change in window height affects the daylight design) and checks the applicability of the rules to the design change. If any of the rules is applicable, the method automatically sends messages to the affected designers informing them about the change and asking for an appropriate response (more details in chapter five). The rules are stored in specially developed data-tables that are part of the management-database (for example, see Data-Table 5-1 on page 84 and Data-Table 5-2 on page 84).

![Diagram](image)

**Figure 4-4 Combination of relational database and object-oriented technologies.**

With such a combination of the relational database and OO technologies, only the relational database structure needs to be modified so as to accommodate the requirements of a project. Building components can be added, deleted, or modified. Their properties can be also changed. Rules can also be added or modified in the relational database.
The problem of flexibility and extendibility of data-structure may be approached with the sole use of the OO technology. This approach will ultimately face the shortcomings of this technology, mainly (Elmasri and Navathe 1994):

- No high level query language is available so as to allow the dynamic and transparent manipulation of data-structure and data values by the end user through an interface software. The end user therefore needs to know how to use the development environment (usually an OO database management system). Such a requirement will severely limit the practicality of the model.

- Behavior rigidity, where a fixed set of rules has to be pre-determined and pre-specified during the model development, makes it unfeasible to add and modify rules to building components during execution to suit specific project requirements.

The approach, which combines both the relational database and the OO technologies, provides a more practical solution.

4.3 ASSEMBLY AND CONFIGURATION

As shown in the previous section, the project-database is built to be a flexible repository of active building components that suits the requirements of individual building projects.

As such, the building-components-database part of the project-database needs to be assembled for every new project. The management-database also needs to be configured for the project at hand. While the design team uses the model, some reconfiguration may
be required as well to reflect the changing requirement of a project.

This section shows how the configuration module is developed to enable the design manager perform these tasks. Figure 4-13 on page 78 is a diagram that can be used as a reference to show the user interface through the configuration module.

4.3.1 Assembly of the Building-Components-Database of a New Project

When starting a new project, the design manager activates the configuration module. As the configuration module is instructed for a new project, it creates a new project-database file with the name of the project as provided by the design manager. The new project-database file is created as a copy of an already built standard database that only includes unpopulated data-tables of the management-database (section A.1.2). No data-table exists a priori in the building-components-database. The design manager is then required to define the data-structure of every building component that exits in the project at hand so as to enable members of the design team to record design information.

Because a building consists of a large number of components, such task is extremely difficult and time consuming. It will definitely discourage the use of the model. Therefore, a general-database is developed to provide assistance to design managers in performing this task. The general-database as shown in Figure 3-1 on page 36 is not part of the central project-database, rather a supporting resource that exists separately and can be used simultaneously by several design managers for different projects.
4.3.1.1 The General-Database

The purpose of the general-database is to provide assistance to design managers in assembling the building-components-database. It is able to do so by containing data-structure for a large pool of building components. Three main observations influence the development of the general-database:

- The first observation is that building components are repeatedly used. As most of the building components are manufactured, they have to be widely used to reduce their cost. Nevertheless, only a limited number of the available building components are used in a certain project.

- The second observation is that building components can be grouped into systems. For example, a column and beam structural system will contain components such as concrete foundation, concrete column, concrete beam, and concrete slab.

- The third observation is that some buildings resemble other buildings in most of their building components. For example, most wood construction homes will share most of their components.

The general-database development, as shown in Figure A-2 in Appendix “A”, reflects these observations through assembling building parts into three levels of increasing complexity: Components, Systems, and Configurations. Members at the Components level such as doors, beams, ducts, and luminaries are the primary ingredients in a building. Each
component is described through its attributes. The description is recorded in a special data-
table (Data-Table 4-4\(^1\)) which carries data about the data-structure of a variety of building
components. Figure 4-5 shows a sample of the data in the Data-Table 4-4. The next level is
the Systems level. Examples for members of this level are cast-in-place flat slab concrete
structural system, all-water HVAC system, and restroom system. Each system contains a
group of related components that form the system. The restroom system, for example,
includes lavatory, toilet, urinal, sink, partition, and mirror. Each of these components is
fully described in the components level (Data-Table 4-4). A special data-table in the
general-database (Data-Table 4-5) carries the data which identifies the building components
in various building systems. Figure 4-6 shows a sample data in the Data-Table 4-5. The
highest level is the Configurations level. Members of this level are complete building
configurations. Examples are a concrete structure low-rise office building with all-air
HVAC system, or a steel structure warehouse with infrared heating. Such configurations are
required when a data-structure for a similar project needs to be assembled. The general-
database carries a list of existing projects in Data-Table 4-6. Figure 4-7 shows a sample of
the data in Data-Table 4-6.

\(^1\) Names of all data-tables that belong to general-database start with GNL.
Figure 4-5 Sample of data in Data-Table 4-4 that describes data-structure of variety of building components.

Figure 4-6 Sample of data in Data-Table 4-5 that defines building components that belong to a building system.
Figure 4-7 Sample of data in Data-Table 4-6 that records current projects.

The role of the general-database is not limited to be a large pool of data-structure. As the prebuilt rules are usually valid for most buildings (for example, a change in the thermal resistance of an external wall affects the HVAC system design), they are considered reusable pieces of information and can be included, with some minor modifications, in every new project. The general-database therefore includes this type of rules. The prebuilt rules are stored in Data-Table 4-7. Figure 4-8 shows a sample of the data in Data-Table 4-7.
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Window</td>
<td>thermal_resistance</td>
<td>hvac</td>
<td>0.15</td>
<td>No</td>
</tr>
<tr>
<td>20</td>
<td>Window</td>
<td>width</td>
<td>hvac</td>
<td>0.25</td>
<td>No</td>
</tr>
<tr>
<td>21</td>
<td>Window</td>
<td>width</td>
<td>illu</td>
<td>0.25</td>
<td>No</td>
</tr>
</tbody>
</table>

Figure 4-8 Sample of data in Data-Table 4-7 that describes some prebuilt rules.

The general-database is also implemented with the use of the relational database technology. The configuration module uses the SQL capabilities to pick a component, a system, or a configuration from the general-database and reconstruct it in the building-components-database of a new building project.

4.3.1.2 The Assembly Process

As the design manager activates the configuration module to assemble the building-components-database of a new project, s/he is provided with three options (selection icon S2 in Figure 4-13). The design manager can perform the assembly task through building components, systems, or existing projects. These options follow the same rational used in developing the general-database.

In case the design manager decides to assemble the building-components-database from another project, the configuration module connects itself to the project database of that project. It reads the data in Data-Table 4-3 (page 62) in the management-database of that project and creates in the building-components-database of the new project all the data-tables that represent the building components and their attributes. The configuration module also populate the Data-Table 4-3 in the management-database of the new project with data that describe the created building components. The prebuilt rules are retrieved
as well from the existing project and recorded in the new project.

In case the design manager decides to assemble the building-components-database of the new project from the building systems (in the general-database), the configuration module recognizes, from Data-Table 4-5 (page 68) in the general-database, all the building components that belong to that system. The configuration module retrieves the data-structure of each of these building components from Data-Table 4-4 (page 68) in the general-database. It creates in the building-components-database of the new project all the data-tables that represent these components. It also populates Data-Table 4-3 (page 62) in the management-database of the new project with data that describe the created building components. The relevant prebuilt rules are also retrieved from Data-Table 4-7 (page 69) in the general-database and recorded in the management-database of the new project in Data-Table 5-1 (page 84).

In case the design manager decides to use the components level in the general-database, the configuration module retrieves the data-structure of the needed component from Data-Table 4-4 (page 68) in the general-database and creates a data-table for that component in the building-components-database. Similar to the previous cases, the configuration module populates the Data-Table 4-3 (page 62) in the management-database of the new project with data that describe the created building component. The relevant prebuilt rules are also retrieved from Data-Table 4-7 (page 69) in the general-database and recorded in the management-database of the new project in Data-Table 5-1 (page 84).
Using the general-database in assembling the building-components-database facilitates the task considerably. However, the assembled data-structure has to be fine-tuned for the requirements of a specific project. Such fine-tuning is performed using the same capabilities that exist to configure an existing project (project that already uses the model). Next section shows how the configuration module is developed for this purpose.

4.3.2 Configuring the Project-Database of an Existing Project

During the development of detailed design, the requirements of a building project slightly vary. Some new building components may need to be added, some existing building components may need to be described in further detail, or the rules that link some building component may need to be modified. Because the project-database is built to be flexible, such variations can be accommodated in the model.

The configuration module is developed to provide the design manager with the ability to configure the project-database. As Figure 4-13 shows (selection icon S3), there are three categories of configuration to an existing project. These are the configuration of building components, the configuration of prebuilt rules, and the configuration of the project menu which allows designers to access the building components in the project-database.

4.3.2.1 Configuring Building Components

The configuration module provides the design manager with several options to configure building components. As selection icon S4 in Figure 4-13 shows, the design manager can create data-structure for a new building component, modify or delete the data-structure of
an existing building component.

When creating data-structure for a new building component, the design manager is assisted with an access to the general-database. If the needed building components is available in the general-database, the design manager simply pick the component. Similar to the process in section 4.3.1.2, the configuration module uses the capability of SQL to retrieve the data-structure of the new component from the general-database and to build it in the building-components-database. Data-Table 4-3 in the management-database is also populated with data that describes the newly created building component. However, If the needed building component does not exist in the general-database, the design manager creates through the configuration module a new data-table in the building-components-database. The new data-table represents the new building components. The design manager needs to specify all the attributes that describe the component. This is similar to the creation of the Circular_Window in section 4.2.1.

In case the design manager needs to modify the data-structure of a certain building component (example in Figure 3-3 on page 40), the configuration module shows the design manager the current data-structure of the data-table that represent this building component. It does this by retrieving the relevant data from Data-Table 4-3 (page 62) in the management-database. As the design manager modifies the data-structure of the building component, the configuration module using SQL capabilities adds, deletes, and modifies attributes of the data-table of the building component in the building-components-database. The changes are also reflected in Data-Table 4-3 in the
management-database.

In case the design manager wants to delete unused building component, the configuration module also uses SQL to remove the data-table of this component from the building-components-database. It removes the corresponding data from Data-Table 4-3 as well.

4.3.2.2 Configuring Prebuilt Rules

The design manager can also manipulate the prebuilt rules (Figure 4-10 on page 76). S/he has an access to all the attributes of all the building components in the building-components-database. For each attribute, s/he is able to browse through all the rules that are related to the attribute. The design manager is able to make modifications in any rule and update the rule. S/he is also able to temporarily ignore a rule (selection icon S6 in Figure 4-13). For a certain attribute, the design manager is able to create new rule and define all the data element of this new rule. The configuration module, in each case, makes modifications in Data-Table 4-7 (page 69) in the management-database using SQL capabilities.

4.3.2.3 Configuring the Project Menu

Another necessary configuration that is needed for the model to function is the project menu. This menu enables the designers to access the building components in the building-components-database through the modules of the model as shown in Figure 4-11. The design manager classifies the building components into groups as shown in Figure 4-12. Each group has a menu heading (e.g., Architecture, Structure). Data about
the contents of a menu is stored in Data-Table 4-8 in the management-database. Figure 4-9 shows a sample of the data in Data-Table 4-8. The same menu items appear in all modules that require access to the building components in the building-components-database as in Figure 4-11.

![Diagram](image)

Figure 4-9 Sample of data in Data-Table 4-8 that describes configuration of project menu.
Figure 4-10 Configuration of a prebuilt rule using configuration module.
Figure 4-11 Project menu as it functions in data manipulation module.

Figure 4-12 Interface to configure project menu to allow access to available building components.
Figure 4-13 Diagram to illustrate user interface in configuration module\(^1\).

\(^1\) Please refer to the "Legend for User Interface Diagrams" on page XV.

Ahmed Mokhtar, Ph.D Thesis ..... 78
4.4 SUMMARY

The building-components-database that carries the design data for the various components in a building project needs to be flexible in its data-structure so as to adapt to the variety of building designs. In addition, to conform with the core concept of the information model, the components in this database need also to be active in helping the management of design changes. The use of a management-database that carries information about the data-structure of the various building components, plus the use of combined Object-Oriented and Relational Database technologies, enable the satisfaction of these requirements. The development of a general-database with three levels of building components assembly facilitates the data-structure of the building-components-database. As such, the capability of the model to manage design changes can be utilized in virtually any type of projects. This is seen to be vital for the professional utilization of the model.
5.1 INTRODUCTION

In chapter three, a conceptual perspective of the developed information model was presented. The perspective exhibited the model capabilities to manage design changes. These capabilities are not limited to propagating current design changes; rather they are extended to include tracking past design changes and planning and scheduling of future ones. This chapter exhibits the development of these capabilities in detail. For each capability, the objective is stated, the developed methodology is described, and the user interface is illustrated.

As will be shown throughout this chapter, the management-database plays a key role in the development of the model capabilities to manage design changes. For reference purposes, appendix “A” displays the conceptual data-structure of the management-database. Copies of its data-tables are also used throughout this chapter to facilitate explanations.

5.2 PROPAGATING CURRENT DESIGN CHANGES

5.2.1 OBJECTIVES

As discussed in section 1.1.3 on page 8, a major cause of failure to accommodate design
changes in a multi-disciplinary design team is the lack of an effective mechanism to propagate design changes. The objective of providing the model with this capability – which is key to the model development – is to overcome this cause of failure. The aim is to provide the design team with an effective methodology that assists in accommodating design changes.

5.2.2 Methodology

Based on our professional experience, we argue that the inability of the design team to propagate design changes is due to two difficulties. The first difficulty is the large number of building components data that exists in a project. It is almost impossible to manually follow the changes that are experienced by the individual data items in a project. The second difficulty is the lack of the linking knowledge which tells what disciplines are affected by a specific design change in a building component and how they are affected. The lack of such linking knowledge is mainly due to the different educational background of the designers who belong to the various design disciplines.

To overcome the first difficulty, the model (as illustrated in section 3.2.2 on page 39) assigns the task of propagating design changes to the building components themselves, hence reducing the need for the inefficient human monitoring of the design changes. To overcome the second difficulty, the building components are provided with the required linking knowledge.

However, the main challenge that remains is in acquiring the linking knowledge. How
can we know the design disciplines that are affected by a design change, for example, in the height of the wall? Whoever attempts to answer this question should have design experience in all the disciplines in the design team, a requirement that is almost impossible to satisfy.

5.2.2.1 Acquire the linking knowledge

The developed methodology overcomes this challenge through reversing the knowledge acquisition process. Instead of trying to define the disciplines that are affected by a change in the value of every attribute of every building component, we define the building components attributes that are needed by every discipline. Unlike the normally required process, the reversed process is easily performed by asking each designer about the input data that s/he uses during the design of a certain building system. This input data is then translated into the form of attributes of building components. For example, the structural engineer defines that one of the input that is needed to design beams is the weight of the brick walls that are carried by the beams. The input data “weight of the brick wall” is then translated to the attributes “height, width, and brick type” of the building component “brick wall”. If the value of any of these attributes is changed (by the architect or the envelope designer), then the structural engineer needs to be notified.

The input data acquired from the designers is stored in specially structured data-tables that are part of the management-database (Data-Table 5-1 on page 84 and Data-Table 5-2 on page 84). A sample of the data used in these data-tables is shown in Figure 5-1 (page 84) and Figure 5-2 (page 84). The acquired data are used in "if-then" rules that constitute
the linking knowledge. The rules are developed in such a fashion that makes them adaptable for the requirements of individual buildings. They contain variables that are later replaced by values during the execution of the model. A sample of these rules is shown in the following zigzag frame\(^1\). The variables are \(\%n\), \(\%n\text{Old}\), and \(\%n\text{New}\). The rule indicates that any change that is more than the percentage \(\%n\) should result in sending a message.

\[
\begin{align*}
\%n & := \text{PERCENT\_CHANGE\_TO\_HAVE\_AN\_EFFECT} \\
\%n\text{Old} & := \text{ATTRIBUTE\_OLD\_VALUE} \\
\%n\text{New} & := \text{ATTRIBUTE\_NEW\_VALUE} \\
\text{IF} \ (\%n = 0) \ \text{OR} \ (\%n\text{Old} \times (1 - \%n) > \%n\text{New}) \ \text{OR} \ (\%n\text{Old} \times (1 + \%n) < \%n\text{New}) \ \text{THEN} \ \text{SEND\_MESSAGE\_ELEMENTS}
\end{align*}
\]

The values that replace the rules' variables are obtained from the design data and vary from one project to the other. The rules are classified into two categories, prebuilt rules and dynamically built rules. The prebuilt rules are acquired during the configuration process of the model. These usually represent general linking knowledge that is known beforehand. For example, a prebuilt rule would reflect the fact that a change in the height of a wall affects the structural system design. The dynamically built rules are acquired during the use of the model to represent the linking knowledge that can only be defined during design. An example of dynamically built rules would be the design decision to make the height of a wall in a one-story building equal to the clear height of the floor space plus 1000mm.

\(^1\) Text within a zigzag frame is extraction from the program code.
Figure 5-1 Sample of data in Data-Table 5-1 that describes some prebuilt rules.

Figure 5-2 Sample of data in Data-Table 5-2 that describes some dynamically built rules.

1 The headings are truncated and some may have been removed in order for the figure to fit on the page.
To acquire the prebuilt rules, the design manager requests from the senior designer of each discipline to define all the input data needed to design the building system under his/her responsibility. Also defined is the percentage of change to each of these inputs that can affect the design. For example, the structural engineer defines that the "height" of "CMU_backed_brick_veneer_wall" is one of the input to design the beams that carry these walls. He also defines that a change of 10% of the "height" would impact the design of the beams. The design manager – using the configuration module (Figure 4-10 on page 76) – keys in this data. The data is stored transparently to the design manager as data elements of a prebuilt rule in Data-Table 5-1 as shown in Figure 5-1.

![Warning Message](image)

**Figure 5-3** Sample of warning message about design change that results in major design modifications.

A designer can also define if a given change in an input data can result in a major redesign in his or her system. For example, the structural engineer defines that an increase of more than 20% of the span of the open web steel joists will require a complete redesign of all the joists. If the architect attempts to increase the distance between the supporting column by more than 20%, s/he will be immediately notified by a warning message that such a change has a major impact on the structural system (Figure 5-3). S/he
then has the choice to continue with the change or consult first with the structural engineer.

To acquire the dynamically built rules, a technique has been developed. When a designer inserts or changes a value of a building component attribute, s/he is automatically asked to provide relationships with other building components (if they exist) which represent the constraints that govern the inserted or changed value. Figure 5-4 (page 87) shows a sample of a relationship that the envelope designer defines after inserting the value of the attribute “height” of the building component “CMU backed brick veneer wall”. The relationship states that the “height” of the “CMU backed brick veneer wall” should equal the “clear height” of the “floor space” + 1000mm. The data is stored transparently to the designer in Data-Table 5-2 (page 84) automatically. The component that is currently under design is stored in the data field “BUILDING_COMPONENT” along with the current instance and attribute in the fields “BUILDING_COMPONENT_ID” and “ATTRIBUTE_OF_BUILDING_COMPONENT” consecutively. The building component, which the envelope designer makes a relationship with, is stored in Data-Table 5-2 in the field “RELATED_BUILDING_COMPONENT” along with its instance and attribute in the fields “RELATED_BUILDING_COMPONENT_ID” and “RELATED ATTRIBUTE” consecutively.

The “clear height” of the “floor space” is a design decision that is taken by another discipline (the architect). The architect may not be aware that the envelope designer took a design decision that is based on his/her design decision for the clear height of the floor.
space. The architect may later change the clear height without being aware of the effect on the envelope designer. However, the linking knowledge acquired from the envelope designer makes the building component "floor space" aware of the envelope designer dependency on its clear height. The "floor space" will immediately inform the envelope designer when the architect changes its "clear height".

As the designers of various disciplines populate the building-components-database, the linking knowledge is also built up to deal with any potential design changes that may be later performed.

Figure 5-4 Acquisition of dynamically built rule using relation-acquiring submodule.
5.2.2.2 React to Design Changes

When a designer performs a design change, for example, to the attribute "Attr_X" of the instance "Inst_Y" of the building component "Comp_Z", the component becomes active and automatically reacts to the change as follows:

1. It retrieves from Data-Table 5-1 (page 84) and Data-Table 5-2 (page 84) all the data rows that are relevant to the design change. In Data-Table 5-1, the relevant rows are those with the field "BUILDING_COMPONENT" which have the value "Comp_Z" and the field "ATTRIBUTE_OF_BUILDING_COMPONENT" which have the value "Attr_X". In Data-Table 5-2, the relevant rows are those with the field "RELATED_BUILDING_COMPONENT" which have the value "Comp_Z"; the field "RELATED_BUILDING_COMPONENT_ID" which have the value "Inst_Y"; and the field "RELATED_ATTRIBUTE" which have the value "Attr_X". A SQL statement similar to the following is used:

```
SELECT * FROM [MNG-PREBUILT-RULES] WHERE
BUILDING_COMPONENT = "Comp_Z" AND
ATTRIBUTE_OF_BUILDING_COMPONENT = "Attr_X" AND
RULE_IGNORED_? = No
```

2. The data in each of the retrieved rows is used to check if some rules are violated by the design change.

a) In case of data from Data-Table 5-1 (prebuilt rule), the percent change in value of the changed attribute is evaluated. If it exceeds the value specified by the senior designer (and saved in the field "PERCENT_CHANGE_TO_HAVE_AN_EFFECT"), then the process continues as in part 3. If
not, then the data of the next row is examined. Statements similar to the following are used.

\[
\begin{align*}
\text{%n} & := \text{PERCENT\_CHANGE\_TO\_HAVE\_AN\_EFFECT} \quad (\text{from Data-Table 5-1}) \\
\text{%nOld} & := \text{OLD\_VALUE\_OF\_"ATTR\_X"} \quad (\text{retrieved from the building-components-database using SQL}) \\
\text{%nNew} & := \text{NEW\_VALUE\_OF\_"ATTR\_X"} \quad (\text{input from the designer}) \\
\text{IF} \ (\%n = 0) \ \text{OR} \ (\%nOld \times (1 - \%n) > \%nNew) \ \text{OR} \ (\%nOld \times (1 + \%n) < \%nNew) \ \text{THEN SEND\_MESSAGE\_ELEMENTS}
\end{align*}
\]

b) In case of data from Data-Table 5-2 (dynamically built rule), the relationship built by the designer is examined. If it fails then the process continues as in part 3. If the relation continues to be valid then the data of the next row is examined. Statements similar to the following are used (based on the data in the second row in Figure 5-2 on page 84 and the changed attribute is the height of the floor space FS001). The statement search for the height of the instance “EW001” of the component “CMU\_Backed\_Brick\_Veneer\_Wall” under the condition that the height is 1000mm higher than the clear_height of the instance “FS001” of the component “Floor\_Space”.

\[
\begin{align*}
\text{SELECT CMU\_Backed\_Brick\_Veneer\_Wall.height from CMU\_Backed\_Brick\_Veneer\_Wall, Floor\_Space where (CMU\_Backed\_Brick\_Veneer\_Wall.CMU\_Backed\_Brick\_Veneer\_Wall_id = 'EW001') AND (Floor\_Space.Floor\_Space_id = 'FS001') AND (CMU\_Backed\_Brick\_Veneer\_Wall.height=Floor\_Space.clear\_height+1000)} \\
\text{IF no value is retrieved THEN} \quad \text{SEND\_MESSAGE\_ELEMENTS}
\end{align*}
\]

Ahmed Mokhtar, Ph.D Thesis ....89
3. Messages are automatically sent by the building component to inform the affected designer about the change. The messages are saved in the form of data values that are recorded in a specially structured data-table (Data-Table 5-3) in the management-database. The data field that are to be filled depend on the type of rule that is used by the changed building component.

![Data-Table 5-3](image)

**Figure 5-5** Sample of data in Data-Table 5-3 that describes some message elements.
a) In case that a prebuilt rule is violated, the following data are sent to the Data-Table 5-3.

- CHANGED_ATTRIBUTE
- CHANGED_BUILDING_COMPONENT
- OLD_VALUE (of the changed attribute)
- NEW_VALUE (of the changed attribute)
- WRITING_DATE (from the computer clock)
- DISCIPLINE_TO (AFFECTED_DISCIPLINE as in Data-Table 5-1 on page 84)
- DISCIPLINE_FROM (DESIGNING_DISCIPLINE as in DATA-TABLE 4-3 on page 62)
- SENT_BECAUSE_CHANGE_ID (CHANGE_ID from DATA-TABLE 5-4 on page 99)
- PREBUILT_OR_DYNAMICALY_? (prebuilt)
- VIOLATED_RULE_ID (PREBUILT_RULE_ID from Data-Table 5-1)
- CAUSE_OF_THE_EFFECT (form Data-Table 5-1)

b) In case that a dynamically built rule is violated, the following data are sent to the Data-Table 5-3

- CHANGED_ATTRIBUTE
- CHANGED_BUILDING_COMPONENT
- CHANGED_BUILDING_COMPONENT_ID
- OLD_VALUE (of the changed attribute)
- NEW_VALUE (of the changed attribute)
- AFFECTED_ATTRIBUTE (ATTRIBUTE_OF_BUILDING_COMPONENT as in Data-Table 5-2 on page 84)
- AFFECTED_BUILDING_COMPONENT (BUILDING_COMPONENT as in Data-Table 5-2)
- AFFECTED_BUILDING_COMPONENT_ID (BUILDING_COMPONENT_ID as in Data-Table 5-2)
- RELATIONSHIP (RELATIONSHIP as in Data-Table 5-2)
- WRITING_DATE (from the computer clock)
- DISCIPLINE_TO (DESIGNING_DISCIPLINE of the affected BUILDING_COMPONENT as in DATA-TABLE 4-3 on page 62)
- DISCIPLINE_FROM (DESIGNING_DISCIPLINE of the changed BUILDING_COMPONENT as in DATA-TABLE 4-3 on page 62)
- SENT_BECAUSE_CHANGE_ID (CHANGE_ID from DATA-TABLE 5-4 on page 99)
- PREBUILT_OR_DYNAMICALY_? (dynamically built)
- RULE_WRITING_DATE (WRITING_DATE as in Data-Table 5-2)
- VIOLATED_RULE_ID (DYNAMICALLY_BUILT_RULES_ID form Data-Table 5-2)

A sample of part of these message elements is shown in Figure 5-5.
5.2.2.3 Deliver Design Change Messages

As the model is currently implemented, designers are expected to regularly check their message modules to find out about any design change messages. Sound and/or visual alert can be adopted to inform about the arrival of a new message. In any case, once a certain discipline starts the message module, the data in the Data-Table 5-3 that has the name of that discipline in the field “DISCIPLINE_TO” are automatically retrieved. From this data, messages are automatically built using a standard message structure that embeds variables. For example, the standard message that is sent as a result of a dynamically built rule is as follows:

"The attribute ", CHANGED_ATTRIBUTE, ", of the building component ", CHANGED_BUILDING_COMPONENT, " (", CHANGED_BUILDING_COMPONENT_ID, ")' has been changed from the value (", OLD_VALUE, ") to the value (", NEW_VALUE, ")."

According to a dynamically built rule, such a change may affect the design of the attribute ", AFFECTED_ATTRIBUTE, ", of the building component ", AFFECTED_BUILDING_COMPONENT, " (", AFFECTED_BUILDING_COMPONENT_ID, ")."

It is advisable to take some of the following steps:
* Redesign the affected building component.
* Negotiate the change with designing discipline.
* Modify the rule that caused this message.

If you need more information, please push the HELP button. If you want to see and edit the rule that caused this automated message to be sent, please push the RULE button"
This standard message is shown to the designer (Figure 3-6 on page 44) after replacing the variables with values from Data-Table 5-3 as follows:

The attribute 'clear_height' of the building component 'Floor_Space (FS001)' has been changed from the value (6500) to the value (7600).

According to a dynamically built rule, such a change may affect the design of the attribute 'height' of the building component 'CMU_Backed_Brick_Veneer_Wall (EW001)'.

It is advisable to take some of the following steps:
* Redesign the affected building component.
* Negotiate the change with designing discipline.
* Modify the rule that caused this message.

If you need more information, please push the HELP button.

If you want to see and edit the rule that caused this automated message to be sent, please push the RULE button.

The message as shown has the following characteristics:

1. It is sent only to the affected designers.

2. It is unambiguous, it defines exactly the attributes and the building components that have changed. It also informs about the old value of the attribute and the new value.

3. When the message is initiated through a prebuilt rule, it defines what are the possible effects of the design change on the affected building system.

4. When the message is initiated through a dynamically built rule, it defines only the attributes and the building components that are affected by the design change.

5. Finally, it provides advice on how to react to the design change.
The affected designer who receives the message is expected to take a corrective action in response to the design change. The message module allows the designer to review and modify the rules that cause messages to be sent, thus providing the model with the flexibility to adopt new relations that designers may develop among building components. Such capability is possible because of the link between Data-Table 5-3 (page 90) and both Data-Table 5-1 (page 84) and Data-Table 5-2 (page 84) (Data-Table 5-1 and Data-Table 5-2 are not linked). This link is through the field “VIOLATED_RULE_ID” in Data-Table 5-3 and each of the field “PREBUILT_RULE_ID” in Data-Table 5-1 and “DYNAMICALLY_BUILT_RULES_ID” in Data-Table 5-2.

5.2.3 USER INTERFACE

Two modules of the information model are used to realize the model capability to propagate design changes. The first module is the data manipulation module. Through this module, a designer records design decisions into the building-components-database. S/he also defines the dynamically built rules by accessing the relation-acquiring submodule. The second module is the message module where designers receive automated messages from the building components.

The interface of the data manipulation module starts by providing the designer with access to all projects that are currently under design so as to select one of them. When a designer selects a project, the data manipulation module is connected to the project-database of this project. The interface allows the designer to select a specific building
component and manipulate any instance of this component. The interface therefore provides access to all the building components that exist in the building-components-database. Because every project contains different building components, the interface has a customizable project menu that provides access to the various building components. The configuration module enables such customization in the project menu as shown in Figure 4-12 on page 77. Once a building component is selected, the designer has the option to create new instances of the component or work with existing ones (selection option [S1] in Figure 5-6). The designer is also able to delete the selected instance or modify the value of its attributes [S2]. Therefore the interface shows all the attributes of any selected building component as they are structured in the building-components-database. Figure 3-2 on page 38 shows a sample of the interface of the data manipulation module.

Once a designer changes the value of any attribute, the relation-acquiring sub-module starts. The interface for this module shows the building component, the instance, and the attribute that are currently under change. The interface provides the designer with the ability to select any attribute of any instance of any building component in the building-components-database which can have a relation with the component that is currently under design. The interface also provides the designer with the capability to build any relation between the components. It also allows multiple relations. Figure 5-4 on page 87 shows a sample of interface for the relation-acquiring sub-module.
Figure 5-6 Diagram to illustrate user interface in data manipulation module\(^1\).

\(^1\) Please refer to the "Legend for User Interface Diagrams" on page XV.
The message module also starts by allowing the designer to choose between all the projects that are currently under design. The module is then connected to the relevant project database. The interface to retrieve the required messages is able to get only the messages that are related to a specific discipline. The designer can choose among retrieving all the messages since the start of the project, only the unread messages, or the messages that are read but have not yet been reacted to (unacted upon). The designer also has access to the rule that causes any message to be automatically sent. Such access allows the designer to modify the rule so as to be updated with current relations in the
project. The designer can make the building component ignore a certain rule if it is temporary or no longer applies to the design. Figure 3-6 on page 44 shows a sample interface for the message module.

5.3 TRACKING PAST DESIGN CHANGES

5.3.1 OBJECTIVES

By tracking past design changes of a project, managers of design teams are provided with the capability to:

1. Observe and analyze the development of the detailed design.

2. Evaluate the performance of the various designers in a project. Some designers make ad hoc design decisions which need to be later modified, costing the design team time and money.

3. Determine responsibilities for design incompatibilities in case of failure.

5.3.2 METHODOLOGY

The model keeps record of every design change that is initiated by any member of the design team. When a designer performs a design change, the changed building component automatically connects to a specially structured data-table in the management-database (Data-Table 5-4 on page 99). The component sends data elements about the design change that include the changed attribute, the building component, and the instance of this component. The component also sends the value of the changed attribute
before and after the design change. In addition, the component recognizes - from the computer clock – the day and time of the design change and records them in Data-Table 5-4. It also recognizes the discipline that is responsible for the change through contacting Data-Table 4-3 on page 62 and records the result in Data-Table 5-4. A sample of the data used in Data-Table 5-4 is shown in Figure 5-8.

![Data Table Image]

**Figure 5-8 Sample of data in Data-Table 5-4 that describes some design changes and enables their tracking.**

With the data-structure of Data-Table 5-4, the data-table can be queried according to the requirements of the design manager. The following is a sample query statement using SQL to extract data. The query obtains all the design changes that occur to the height of the instance “EW001” of the component “CMU_Backed_Brick_Veneer” during a one month period starting from June 20, 1997 at noon.
SELECT * FROM [MNG-DESIGN-CHANGES] WHERE
building_component = "CMU_Back_Led_Brick_Veneer" AND
building_component_ID = "EW001" AND attribute = "height"
AND date_time >= "1997/06/20 12:00" AND date_time <=
"1997/07/20 12:00"

The design manager needs not to build such a query. It is automatically and transparently
built inside the tracking module. The design manager needs only to provide the query
data through the interface with the tracking module (see Figure 3-8 on page 47).

The management-database is developed with links between the design changes and the
related automated message that are sent by the changed building component. This link is
made through the attribute CHANGE_ID in Data-Table 5-4 which is referenced by the
attribute SENT_BECUSE_CHANGE_ID in Data-Table 5-3 on page 90. When the
building component records the design change data in Data-Table 5-4, the data-table
automatically assigns a CHANGE_ID to the design change. This CHANGE_ID is
retrieved and utilized by the building component when recording the automated messages
that result from that design change in Data-Table 5-3 (page 90). Through such a link, all
the messages that are automatically sent by the changed building component are shown to
the design manager (sample in Figure 3-10 on page 48). The messages are retrieved from
Data-Table 5-3 using SQL. A sample for a utilized SQL statement is shown in the
following zigzag frame. The statement extracts all the message elements that were sent
because of the design change number 28.

SELECT * from MNG-MESSAGE-ELEMENTS WHERE
SENT_BECUSE_CHANGE_ID = 28
The design manager is also shown whether the designer who receives any of these messages has read it or not and whether he has taken a corrective action or not. Such capability helps design managers define responsibilities in case of failure in the compatibility of design information due to design changes.

### 5.3.3 User Interface

Tracking of past design changes is performed through the tracking module. The interface of the tracking module starts with allowing the design manager to select from all the projects that are under design (Figure 5-9). The tracking module is then connected to the appropriate project-database. The design manager can then define the constraints that limit the tracking of the change data according to his/her requirements. The first constraint, which is optional, is the time period for the design change. The design manager is then able to select between tracking all the design changes that are done by a certain design discipline or tracking the design changes of a certain component (selection [S1] in Figure 5-9). In case of selecting a specific building component, the design manager has the option to further limit the component to one of its instances and/or one of its attributes. Because every project contains different building components, the interface has a customizable project menu that provides access to the various building components. The configuration module provides the capability to make such customization in the project menu as shown in Figure 4-12 on page 77. Figure 3-8 on page 47 shows a sample interface of the tracking module. Once the constraints are defined, the required design changes are shown. For each change, the interface shows the
time, the discipline, the building component, the instance, the attribute, the old value, and
the new value of the design change.

**Figure 5-9 Diagram to illustrate user interface in tracking module.**
The constraints that define the shown design changes appear to the design manager. Figure 3-9 on page 47 shows a sample of the interface for the results of a tracking query. The design manager can select one of the displayed design changes and track all the automated messages that are sent by the changed building component to the affected designers. This is achieved by double clicking on one of the shown design changes. Figure 3-10 on page 48 shows a sample of the interface that shows such messages. The interface provides the design manager with the total number of messages and enable him/her to browse through the messages. For each message, the design manager sees which discipline received the message, the text of the message, and whether the message was read and acted upon or not.

5.4 **PLANNING AND SCHEDULING FUTURE DESIGN CHANGES**

5.4.1 **OBJECTIVES**

As discussed in section 3.2.5 on page 48, some design changes have interrelated effects on several other building components in an extended fashion. Figure 5-10 on page 105 shows a sample of such changes (Figure 5-10 is a copy of Figure 3-11 on page 49 and is explained in section 3.2.5). Design managers need answers to some important questions before the design team is engaged in that type of changes. These questions are:

I. What are the interrelated design changes necessary to accommodate an initial design change?
II. Are there alternative ways to accommodate the initial change? In the example in Figure 5-10, the HVAC designer can accommodate the change in two different ways. Each alternative, composed of a sequence of changes, is termed as a change-path.

III. Are there design changes that may be repeated within a change-path and how to avoid these repetitions? In the example in Figure 5-10, the type of luminaires (L1) is redesigned several times.

IV. Ultimately, how long will it take to fully accommodate the initial design change through a sequence of interrelated design changes or change-path? Are some of these changes critical to determine the total duration along a given change-path? These are important questions especially when the initial design change is requested by the owner. The design manager can then ask for a penalty-free extension for submitting the working drawings late.

V. How many design man-hours are required to accommodate the initial change? The answer to this question can help determine additional design fees.

VI. How much is the change expected to impact the construction costs of the building? Some interrelated design changes in the change-path may incur so much expenditure as to have the initial change reconsidered.

VII. What level of recommendation is assigned to every change in a change-path by the
designer? A level of recommendation indicates whether or not a designer sees the change improving the building system that s/he is responsible for. It can also indicate that the change is impossible.

![Diagram showing interrelated design changes]

**Legend**

- **(OR) Relationship**
  - H1
  - H2
  - S1
  - S3

- **(AND) Relationship**
  - S2

<table>
<thead>
<tr>
<th>Change in:</th>
<th>F5  Location of sprinkler outlets (Fire Safety)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1. Space function <em>(Architecture)</em></td>
<td>H1. HVAC duct width <em>(HVAC)</em></td>
</tr>
<tr>
<td>(from executive office to computer lab)</td>
<td>H2. HVAC duct depth <em>(HVAC)</em></td>
</tr>
<tr>
<td>A2. Reduce clear height <em>(Architecture)</em></td>
<td>L1. Luminaire type <em>(Lighting)</em></td>
</tr>
<tr>
<td>I1. Wall finishing <em>(Interior Design)</em></td>
<td>S1. Width of hole in structural beam <em>(Structure)</em></td>
</tr>
<tr>
<td>(from dark mahogany wood to white paint)</td>
<td>S2. Height of hole in structural beam <em>(Structure)</em></td>
</tr>
<tr>
<td></td>
<td>S3. Depth of beam <em>(Structure)</em></td>
</tr>
</tbody>
</table>

**Figure 5-10 Scenario for interrelated design changes.**

The objective of the planning and scheduling capability of the model is to provide answers to these questions. Consequently, an informed decision can be taken about whether or not the change is to be done and the best way to do it.

In traditional design environments, answers to these questions are not easy to obtain. Two
types of difficulty are identified. The first type is due to the lack of information. This is apparent when trying to answer question #1 on page 103. It is hard for the design manager and for individual team members to identify all the disciplines that are affected by a specific change in a given building component. Change-paths as shown in Figure 5-10 are difficult to define because there is a scarcity of knowledge regarding how design changes are linked to one another. The second type of difficulty stems from the fact that some information exists but is scattered among various disciplines that work independently and are separated geographically, and there is no structured methodology to collect and organize this data.

5.4.2 Methodology

The methodology proposed uses the active components of the project-database in order to overcome these two types of difficulties. The methodology consists of four stages which are described below:

a) Collect data.
b) Organize changes.
c) Schedule changes.
d) Calculate costs and durations.

5.4.2.1 Collect Data

Collecting the data is accomplished through the Virtual Change Module of the information model (Figure 3-1 page 36). When interrelated design changes need to be studied, the designer who is responsible for the initial change starts a ‘New Case’ in the virtual change module. S/he defines what needs to be changed as shown in Figure 5-11
where the architect selects the attribute “used_for_activity” of the instance “space009”. The architect changes the function of that space from an executive office to a computer lab as shown in Figure 5-12. Such a change is not actually done in the building-components-database; nevertheless, the linking knowledge embedded in the management-database (Data-Table 5-1 on page 84 and Data-Table 5-2 on page 84) is used to recognize the disciplines that are affected by the change. Automated messages are sent to these disciplines by populating a specially structured data-table (Data-Table 5-5 on page 109) with data about the virtual change. The elements of this message (called virtual message) are similar to those of the regular automated messages that are used in propagating design changes (Data-Table 5-3 on page 90). The only difference is in including a field for CASE_ID. The CASE_ID is automatically generated when a designer selects a new case (Figure 5-11). Its function is to link all the data and messages that belong to a group of interrelated virtual design changes in the various data-tables in the management-database.

![Image](image_url)  
*Figure 5-11 Starting a new case in virtual change module.*
The affected designers receive the messages also in the virtual change module (Figure 5-13 on page 110). The messages notify the designers about the initial change and ask for a response. In Figure 5-13, a sample message to the HVAC designer would read as follows:

“In VIRTUAL MODE, The attribute 'used_for_activity' of the building component 'space (space009)' has been changed from the value (Executive Room) to the value (Computer Lab). Such a change may affect the design of the 'HVAC' system. Please indicate the expected modifications in the building components.”

![Data for A Design Change](image)

**Figure 5-12 Data provided by designer for proposed design change.**

Each affected discipline defines what components in its system need to be changed and also how they are to be changed. For example, the HVAC engineer, using his/her design experience, defines that the change in the space activity requires a minimum 30% increase in the supply duct size. S/he can do that in two ways: either by increasing the
width (dimension_x) of the duct from 600 mm to 800 mm (Figure 5-13 and Figure 5-14) or by changing the duct depth from 400 mm to 550 mm. Once the HVAC designer specifies the two possibilities, automated messages are sent to the disciplines that are affected by each of these changes. The affected disciplines read the messages and react to them in similar fashion.

<table>
<thead>
<tr>
<th>ID</th>
<th>CHANGED_ATTRIBUTE</th>
<th>CHANGED_BUILDING_COMPONENT</th>
<th>CHANGED_BUILDING_COMPONENT_ID</th>
<th>OLD_VALUE</th>
<th>NEW_VALUE</th>
<th>AFFECTED_ATTRIBUTE</th>
<th>AFFECTED_BUILDING_COMPONENT</th>
<th>AFFECTED_BUILDING_COMPONENT_ID</th>
<th>RELATIONSHIP</th>
<th>WRITING_DATE</th>
<th>BEEN_READ_?</th>
<th>BEEN_ACTED_UPON_?</th>
<th>DISCIPLINE_TO</th>
<th>DISCIPLINE_FROM</th>
<th>SENT_BECauses CHANGE_ID</th>
<th>PREBUILT_OR_DYNAMICALLY_?</th>
<th>RULE_WRITING_DATE</th>
<th>VIOLATED_RULE_ID</th>
<th>CASE_ID</th>
<th>CAUSE_OF_THE_EFFECT</th>
</tr>
</thead>
</table>

The data specifying the sequence of changes and the alternatives are collected in a specially structured data-table (Data-Table 5-6 on page 111). Each design change is automatically assigned a code (e.g. A1, H1) that is used in the data-table. The details of
the design change are stored in Data-Table 5-7 on page 112. Both Data-Table 5-6 and Data-Table 5-7 are linked through the design change code. A sample of the collected data is illustrated in Figure 5-15 on page 111 which represents the sequence of design changes that appear in Figure 5-10 on page 105.

Figure 5-13 Response to virtual change message in virtual change module.
Figure 5-14 Data provided by designer for proposed design change.

Figure 5-15 Sample of data in Data-Table 5-6 that defines sequence of changes.
Additional data is also collected from designers during the process. As Figure 5-14 shows, when the HVAC designer specifies a change in the duct width, the system requires him/her to answer questions about the actual design time required to make the changes; the additional man-hours needed; the expected difference on building cost when the change is done; and the level of recommendation for that change. This data is saved in Data-Table 5-7 and a sample of the data is shown in Figure 5-16. The process continues till all related changes are addressed and their data are collected.

<table>
<thead>
<tr>
<th>ID</th>
<th>CASE_ID</th>
<th>CHANGE</th>
<th>BUILDING_COMPONENT</th>
<th>BUILDING_COMPONENT_ID</th>
<th>ATTRIBUTE</th>
<th>REQUIRED_MAN_HOURS</th>
<th>DIFFERENCE_IN_BUILD_COST</th>
<th>RECOMMENDATION_LEVEL</th>
<th>REASON_FOR_RECOMMENDATION</th>
<th>DURATION</th>
<th>OLD_VALUE</th>
<th>NEW_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A1</td>
<td>space</td>
<td>space09</td>
<td>used_for_activity</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>As required</td>
<td>executive office</td>
<td>computer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>H1</td>
<td>Hvac_duct_c</td>
<td>duct09</td>
<td>Dimension_x</td>
<td>3</td>
<td>100</td>
<td>3</td>
<td>Better propor</td>
<td>2</td>
<td>600</td>
<td>800</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-16 Sample of data in Data-Table 5-7 which is collected from designers through virtual change module.

5.4.2.2 Organize Changes

After data for the interrelated design changes is collected, two problems need to be
addressed. The first problem is to define the number of possible change-paths as well as the actual design changes that compose each of them. A sub-routine is developed to provide the computer with the logic that can address this problem. The sub-routine, which is described in details in section B.1 (page 179), functions on the data stored in Data-Table 5-6 (page 111). It stores the initial design change as an item in the first change-path along with all its successor changes. When there are alternate successors ("OR" arrows in Figure 5-10), it considers first only one of them. For the first alternative, all interrelated design changes are defined and grouped together to constitute the first change-path.

<table>
<thead>
<tr>
<th>ID</th>
<th>DESIGN_CHANGE</th>
<th>SUCCESSOR</th>
<th>ALTERNATIVE#</th>
<th>CASE_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4734</td>
<td>NULL</td>
<td>A1</td>
<td>ALTERNATIVE#1</td>
<td>2</td>
</tr>
<tr>
<td>4735</td>
<td>A1</td>
<td>H1</td>
<td>ALTERNATIVE#1</td>
<td>2</td>
</tr>
<tr>
<td>4736</td>
<td>A1</td>
<td>L1</td>
<td>ALTERNATIVE#1</td>
<td>2</td>
</tr>
<tr>
<td>4737</td>
<td>A1</td>
<td>I1</td>
<td>ALTERNATIVE#1</td>
<td>2</td>
</tr>
<tr>
<td>4738</td>
<td>H1</td>
<td>F5</td>
<td>ALTERNATIVE#1</td>
<td>2</td>
</tr>
<tr>
<td>4739</td>
<td>H1</td>
<td>S1</td>
<td>ALTERNATIVE#1</td>
<td>2</td>
</tr>
<tr>
<td>4740</td>
<td>I1</td>
<td>L1</td>
<td>ALTERNATIVE#1</td>
<td>2</td>
</tr>
<tr>
<td>4741</td>
<td>NULL</td>
<td>A1</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
<tr>
<td>4742</td>
<td>A1</td>
<td>H2</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
<tr>
<td>4743</td>
<td>A1</td>
<td>L1</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
<tr>
<td>4744</td>
<td>A1</td>
<td>I1</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
<tr>
<td>4745</td>
<td>H2</td>
<td>S3</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
<tr>
<td>4746</td>
<td>H2</td>
<td>S2</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
<tr>
<td>4747</td>
<td>I1</td>
<td>L1</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
<tr>
<td>4748</td>
<td>S3</td>
<td>A2</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
<tr>
<td>4749</td>
<td>A2</td>
<td>L1</td>
<td>ALTERNATIVE#2</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 5-17 Sample of data in Data-Table 5-8 that defines alternative change-paths.
The sub-routine then backtracks along the first change-path till an alternative change is encountered (sample is highlighted in Figure 5-15). A new change-path is then generated using the second design alternative (alternate_successor in Figure 5-15 on page 111) along with its successors. The process continues until all change-paths are generated. The result is stored in a specially structured data-table in the management-database (Data-Table 5-8). A sample of the result is illustrated in Figure 5-17 where two change paths (Alternative #1 and Alternative #2) are identified for the sample data in Figure 5-15.

![Diagram a)](image)

![Diagram b)](image)

**Figure 5-18 Alternate change-paths for scenario in Figure 5-10**

The second problem is to define a logical sequence of the design changes in every change-path. As Figure 5-10 shows, some building components may need to be repeatedly redesigned because they are affected by changes in several other building components (for example, the type of luminaire). It is clearly inefficient to have the lighting designer change the type of luminaire each time a change occurs in a building component. It is therefore important to involve the lighting designer only after all these changes are done. Another sub-routine is therefore developed to determine the proper
“location” of such changes so as to minimize redesigning. The sub-routine, which is described in details in section B.2 on page-182, functions on the data stored in Data-Table 5-8. It delays the positioning of the repeated design changes in a change-path till all preceding design changes have taken their own positions. The subroutine also prepares the design change data to be scheduled using a standard critical path algorithm.

Figure 5-18 shows (in conceptual form that is not actually used in the computer) the result of applying the two developed sub-routine to the scenario in Figure 5-10 where two alternate change-paths are generated. The lower portion of Figure 5-18 shows that the change “L1” is located in the flow diagram only in one position that comes after changes A1, I1, and H2-S3-A2.

5.4.2.3 Schedule Changes

Once the change-paths are identified and the sequence of changes is defined for each of them, a bar chart is generated for each path as in Figure 5-19 using the expected duration of individual design changes (collected in the first stage in section 5.4.2.1 on page 106 and stored in Data-Table 5-7 on page 112). Critical changes are also identified using a standard critical path method (CPM) algorithm (Paulson 1995). Figure 5-19 shows the critical design change activities in darker tone.
Figure 5-19 Bar chart for alternate change-path (b) in Figure 5-18

5.4.2.4 Calculate Costs and Durations

Using data collected in Data-Table 5-7 (page 112) and shown in Figure 5-20 (page 118), the expected impact on the building construction costs ($C_T$) is calculated for each change-path. It is the sum of the positive and negative effects of all design changes on the building cost. The following equation is used:

$$C_T = \sum_{x=1}^{n} C_x$$

Where

- $n$ is the total number of design changes in a change path.
- $x$ is a design change.
- $C_x$ is the value of the attribute "DIFFERENCE_IN_BUILD_COST" in Data-Table 5-7.
For example, the change-path shown in Figure 5-18b on page 114 would result in building construction cost savings of $2900 as indicated in Figure 5-19. Also calculated is the level of recommendation for each change-path \((Rr)\). It is the average of the recommendation levels assigned to all the changes in that path. If one of the recommendation level values of a design change path equals zero, it means that the designer of that change indicates that such a change is not possible to be done. Reasons can include violation of code. In such case, the whole change path is given the recommendation level of zero which tells the design manager that such an alternative cannot be used. The recommendation level has a value of 2.1 in Figure 5-19. This information provides the design manager with the ability to choose the best change-path from the design point of view. The following equation is used:

\[
R_T = \left( \sum_{x=1}^{n} R_x \right)/n \quad \text{(if none of } R_x \text{ equals zero)} \quad \text{or} \quad R_T = 0 \quad \text{(if any of } R_x \text{ equals zero)}
\]

where \( n \) is the total number of design changes in a change path.

\( x \) is a design change.

\( R_x \) is the value of the attribute “RECOMMENDATION_LEVEL” in Data-Table 5-7.

Other calculations are also performed for each change-path, such as the additional design time required to make the changes \((T_T)\) (9 hours in Figure 5-19) using the equation:

\[
T_T = \sum_{x=1}^{n} T_x \quad \text{where}
\]

\( n \) is the total number of critical design changes in a change path.

\( x \) is a design change.
$T_x$ is the value of the attribute “DURATION” in Data-Table 5-7. and the corresponding design cost ($Dr$) ($\$210$ in Figure 5-19) using the equation:

$$Dr = \left( \sum_{x=1}^{n} M_x \right) \cdot h$$

where

- $n$ is the total number of design changes in a change path.
- $h$ is the cost of man hour in the design firm.
- $x$ is a design change.
- $M_x$ is the value of the attribute “REQUIRED_MAN_HOURS” in Data-Table 5-7.

If the initial design change is requested by the owner, this information would substantiate the design manager request for extra design fees and time so as to accommodate the initial change. If the owner agrees, the initial design change can be accomplished through the best “design path” by switching from virtual to actual design mode. If the calculated impact of the initial design change is minimal, the design manager can proceed to actual change mode without delay.

<table>
<thead>
<tr>
<th>Job</th>
<th>Required-Man-Hours</th>
<th>Difference in Added Cost</th>
<th>Required in Main Design</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>H1</td>
<td>1</td>
<td>-3000</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>H2</td>
<td>3</td>
<td>100</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>L1</td>
<td>0.5</td>
<td>200</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>S2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>2</td>
<td>200</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

*Figure 5-20 Partial data from Data-Table 5-7 that is used for calculating costs and durations.*
5.4.3 User Interface

Two modules are used for the planning and scheduling of design changes. First is the virtual change module which interfaces with designers to collect the data. Second is the planning and scheduling module which shows the results to the design manager. The interface for the virtual change module (Figure 5-21 on page 121) starts by allowing the designer to choose among the projects that are currently under design. The designer has the option to start a new change case or to respond to a virtual change message (selection [S1] in Figure 5-21). In the event of starting a new change case, the designer selects the building component, the instance, and the attribute that s/he intends to change. A sample interface design is shown in Figure 5-11 on page 107. Because every project contains different building components, the interface has a customizable project menu that provides access to the various building components. The configuration module has the capability to make such customization in the project menu as shown in Figure 4-12 on page 77. Once the designer defines the changed building component, s/he is asked to provide the data about the design change. A sample interface design for collecting the data is shown in Figure 5-14 on page 111. To reduce the time needed to fill this data, the interface provides the designer with any historic data about the same design change. Figure 5-22 on page 122 shows a sample of the interface for such historical data. The designer can directly use these data or modify some of them.

In the event of responding to a virtual change message, the interface shows the total number of virtual messages that are sent to a specific discipline. The designer is able to
browse through these messages, read each one and respond to it separately. To enable the designer to respond, s/he has access to any building component, any instance, and any attribute of the component. Figure 5-13 on page 110 shows a sample of that interface. Similar to the event of starting a new case, the designer is asked to provide data about the design change. The historical data is also shown to help the designer. The interface also allows the designer to provide a second design change alternative for the same message.

The interface for the planning and scheduling module starts by allowing the design manager to select one of the projects that are currently under design (Figure 5-23 on page 122). S/he selects among the available change cases in the specified project. The interface then shows the results of the change case. The total number of the alternate change paths is shown. The design manager can browse among the various change paths. The results of each change path is shown separately. Figure 5-19 on page 116 shows a sample interface of the results. The bar chart shows every design change in a coded format. The design manager is able to know the details of every design change. This is achieved by clicking on the relevant bar in the bar chart. The interface then shows all the data about the selected design change. Figure 5-24 on page 123 shows a sample of the interface to present the data about a design change. The interface allows the design manager to compare among the various alternatives by showing the results of all the alternatives in a table format (Figure 5-25 on page 123).
Figure 5-21 Diagram to illustrate user interface in virtual change module.
Figure 5-22 Interface to show historical data about certain design change.

Figure 5-23 Diagram to illustrate user interface in planning and scheduling module.
Figure 5-24 Interface to show detailed data of a design change.

Figure 5-25 Interface to show results of all alternative change-paths so as to enable comparing them.
5.5 Summary

The concept of active building components enables the model to manage design changes in terms of propagating current design changes, tracking past design changes, and planning and scheduling future ones. The building components are made active by providing them with the necessary linking knowledge. The acquisition of this knowledge becomes possible after reversing the knowledge acquisition process. The management-database plays a vital role in supporting the model ability to manage design changes. It sustains the linking knowledge in term of rules elements, records the data regarding every design change, carries the message sent by the building components, and maintains the data needed to plan and schedule future design changes. A methodology that included two specially developed subroutines is introduced to encounter the challenges of interrelated design changes. It makes use of the linking knowledge and the active building components to collect needed data from the designers. It then provides the design manager with clear idea on how such a design change can be done.
CHAPTER SIX
VALIDATION OF THE MODEL

6.1 INTRODUCTION

This chapter aims at demonstrating that the developed model, as described in previous chapters and implemented, can achieve its stated objectives. The first part of this chapter describes the environment in which the model is implemented in terms of software and hardware. The second part focuses on validating the model through three specific levels of validation.

6.2 COMPUTING ENVIRONMENT

The information model is implemented on a client/server network. NOVEL-NETWARE (1993) is used as the network management software while MS-WINDOWS-95 (1995) is used as the local operating system. The central project-database is implemented in MS-ACCESS (1994). It is installed on the server, hence accessible by all users. The modules in the designer system and the design manager system (see Figure 3-1 on page 36) are implemented in the software environment LEVEL5 OBJECT (1994). Depending on whether the user is a designer or a design manager, his/her own (client) computer contains the relevant modules. The modules - on the client computers - are connected with the central project-database on the server through the Object Database Connectivity (ODBC) component that is part of the MS-WINDOWS-95 system. This component enables the modules to use SQL in order to manipulate the project-database.
Hardware for the client computers consists of four INTEL™ based personal computers with 166 MHz Pentium processors and 16 MB of RAM. The server is also an INTEL™ based personal computer but with 200 MHz Pentium processors and 64 MB of RAM. These are common type of computers that are used by building design professionals. The computers are located in the same computer lab at the School for Building, Concordia University. Yet, their positions are selected so as none of the users can see the others. This is to emulate the actual design environment where designers of various disciplines are separated spatially.

6.3 VALIDATION

6.3.1 APPROACH

The nature of the developed model makes it unfeasible to compare its outputs with experimental data or data from literature. However, it is important to make sure that the model, as developed, functions properly and achieves its objectives. Therefore, the model is subjected to three levels of validation. The first level aims to show that the model is properly coded on the implementation software and in accordance with the model development. The second level demonstrates that the model provides the capabilities that are expected from its development. The third level establishes that the model improves the current practices of dealing with design changes and can solve their related problems.
6.3.2 First Level of Validation

This level of validation aims to ensure that the model components are coded according to the described development in chapters four and five. Among numerous tasks in this level, the validation includes making sure that the input data are used correctly, the program routines function accurately, the modules are properly connected to the database, and that the expected output data is presented. Several tools that exist within the development software (LEVEL5 OBJECT) are used in the process. The automated syntax validation ensures that any saved program code in syntactically correct even before compiling the code. The “session monitor” tool allows the developer to dynamically follow the program flow step by step and discover any logical error. The “history” tool is also used to print out all the program flow in a textual format (Figure 6-1), hence it can be reviewed for detecting any source of logical error. Several samples of data are used to validate each subroutine in order to make sure that the code is trouble-free.

![Image](image.png)

Figure 6-1 Sample of statements recorded by “history” tool in LEVEL5 OBJECT.

Ahmed Mokhtar, Ph.D Thesis ....127
6.3.3 Second Level of Validation

This level of validation aims to ensure that the components successfully function together and provide the user with the capabilities that are expected from the model. A setup that simulates a multi-disciplinary design environment is formed for this purpose. In this setup, graduate students who have professional experience represent four design disciplines. These disciplines are architecture, structure, HVAC, and envelope. A graduate student is assigned the role of the design manager. Data is extracted to populate the various parts of the model. The first part which needs to be populated with data is the general-database (see Figure 3-1 on page 36). Two references are used for this purpose "Architectural Graphic Standards" (Ramsey and Sleeper 1994) and "Building Construction Illustrated" (Ching 1991). From these references, a number of building components are extracted along with the attributes that describe each of them. Most of the extracted building components are grouped together to form building systems. Prebuilt rules are also added to many of the building components. With such data, the general-database is ready to be used in assembling the data-structure of the building-components-database for a variety of building types. The second part that needs to be populated with data is the central project-database. A warehouse project¹ (DeChiara 1980) is selected for this purpose. The warehouse is 50,292mm (165') by 36,678mm (120' 4''). It consists of two main zones, a storage zone that is 6,198mm (20' 4'') high and an administration zone that is 3,327mm (10' 11'') high.

The difference in height is used as room for mechanical equipment. The administrative zone consists of a general office area, entrance and a sales room, a lunchroom, and two rest rooms.

The structure of the warehouse consists of steel columns that support open web steel joists. The columns are located on non-equally spaced grid. The depth of the open web steel joists that support the building main roof is 457mm (18”) while those supporting the floor above the administration zone – where the mechanical equipment are installed - is 356mm (14”) in depth. The roof is 38mm (1.5”) corrugated steel deck covered with 51mm (2”) rigid insulation while the floor above the administrative zone is open web steel joists that carry 51mm (2”) concrete on 38mm (1.5”) corrugated steel deck. Acoustical ceiling is used in the administrative zone and is suspended from the joists. The external walls are mainly made of CMU_Back_Brick_Veneer with 38mm (1.5”) of rigid insulation. They are supported by reinforced concrete girders. The floor is 127mm (5”) concrete slab over sand and gravel fill.

With the use of the data from the warehouse project, the following sections illustrate the validation of the various capabilities of the model. These capabilities are categorized in similar fashion as the model development which is presented in chapters four and five.

6.3.3.1 Assembly and Population of the Project-Database

The first step in using the model is to assemble the data-structure of the building-components-database so that it can accept the design data of the warehouse. The design
manager performs this task using the configuration module and with the help of the general-database. He starts the process by creating a new project and uses the building systems option to assemble the data-structure (sample is in Figure 6-2). The majority of the building components are assembled through this option. The assembly process automatically adds the relevant prebuilt rules as stored in the general-database. The remaining building components are individually obtained from the general-database (sample is in Figure 6-3) or created from scratch. Each designer reviews the data-structure of the building components that are under his responsibility and may ask the design manager to modify some of these components by adding or editing attributes. For example, the concrete column component that is part of the structural system in the general-database (Figure 6-2) is removed and a steel column component is added (Figure 6-4). The design manager asks each designer about the elements of data which are used as input to the design of his system. Accordingly, the design manager adjusts the automatically added prebuilt rules and creates new ones (samples are in Figure 6-5 to Figure 6-8). After completing the model configuration, each designer uses his own data manipulation module to populate the building-components-database by creating instances of the building components under his responsibility. Occasionally, the designers through the use of the relation-acquiring sub-module create dynamically built rules (sample is in Figure 6-9).

The process proves that the model is capable of flexibly adapting to the data requirements of the project. The initial assembly process of the data-structure takes about one hour for
the warehouse project while adjusting the prebuilt rules take about half an hour for each discipline. This is seen as a reasonable overhead time for using the model. The input of graphical data in form of textual data is difficult. The use of a CADD tool would be useful for this purpose but remains beyond the scope of this project, given time constraints.

Figure 6-2 Assembly of building components in building-components-database from building systems available in general-database.
Figure 6-3 Adding building component from general-database to building-component-database.

Figure 6-4 Configuration of a building component using configuration module.
Figure 6-5 Input data to structural design translated to prebuilt rule.
(If any change occurs to clear height of floor space then the structural engineer needs to be notified because it can affect the height of columns).

Figure 6-6 Input data to structural design translated to prebuilt rule.
(If a change more than or equal 15% occurs to height of CMU backed brick veneer wall then the structural engineer needs to be notified because it can affect the design of beam that carries the wall)
Figure 6-7 Input data to HVAC design translated to prebuilt rule.
(If a change more than or equal 15% occurs to clear height of the floor space then the HVAC engineer needs to be notified because it affect the volume of the conditioned space)

Figure 6-8 Input data to envelope design translated to prebuilt rule.
(If any change occurs to clear height of floor space then the envelope engineer needs to be notified because it affects the height of the wall)
Figure 6-9 Dynamically built rule acquired from structural designer defining one of the conditions used to design depth of Open_Web_Steel_Joist. (The depth of the open web steel joist Ow001 is determined based on the information that the weight of the boiler above it is less than or equal two tons)

6.3.3.2 Propagating Current Design Changes

To validate the model capability to propagate current design changes, one designer (the architect) introduces a random design change. The design change is to increase the height of the storage zone by 1.2m (4’) to satisfy a modification in the owner requirements. Consequently, the changed component automatically sends messages to the affected designers. The structural engineer receives a message informing about the change in height and suggesting that columns design may be affected (Figure 6-10). Similarly, a message is sent to the HVAC designer stating that the HVAC system might be affected.

Ahmed Mokhtar, Ph.D Thesis .... 135
due to the change in the space volume (Figure 6-11). Another message is sent to the envelope designer indicating that the height of the walls is now incompatible with current height of the storage space (Figure 6-12). The structural engineer reads the message, evaluates the design change and only modifies the length of the columns. The HVAC designer reads the message, evaluates the capacity of the Boiler and specifies a larger type. He also resizes the HVAC ducts that supply the storage zone. The envelope designer reads the message and increases the height of the walls. Other automated messages are sent following these design changes. The structural engineer receives two messages. The first informs that the height of the walls has increased, which can affect the design of the beams that carry these walls (Figure 6-13). The second message informs that a heavier Boiler is used, which may affect the design of the floor that carries this equipment (Figure 6-14). The structural engineer re-evaluates the design of the beams that carry the walls and increases their dimensions. He also re-evaluates the design of the columns and the open web steel joists that carry the Boiler and finds these to be safe.

As this scenario illustrates, building components using the captured linking knowledge proves to be capable of adequately propagating design changes and only to the interested designers. The designers are able to perform necessary design modifications in the design information so as to ensure compatibility. The likelihood of errors is greatly reduced.
Figure 6-10 Message received by structural designer informing of design change made by architect.

Figure 6-11 Message received by HVAC designer informing of design change made by architect.
Figure 6-12 Message received by envelop designer informing of design change made by architect.

Figure 6-13 Message received by structural designer informing of design change made by envelope designer.

Ahmed Mokhtar, Ph.D Thesis .... 138
6.3.3.3 Tracking Past Design Changes

To validate the model capability to track past design changes, the design manager inquires about some design changes which occurred during the scenario in section 6.3.3.2. A sample of these inquiries is about the design changes that occur to the weight of the building component “Boiler”. As a result of this inquiry, the change in weight of the boiler appears to the design manager (Figure 6-15). As the design manager double clicks on the design change, the message that is automatically sent to the structural engineer due to this design change appears (Figure 6-16). From the performed inquiries, the model always performed correct tracking of past design changes and their related messages. Hence, its development provides the capability of tracking past design changes.
Figure 6-15 Results of request made by design manager to track design changes made to a building component.

Figure 6-16 Results of request made by design manager to track automated messages that are sent because of certain design change.

Ahmed Mokhtar, Ph.D Thesis .... 140
6.3.3.4 Planning and Scheduling of Future Changes

The aim of this section is to validate the methodology which provides the model with its capability to plan and schedule future design changes. A hypothetical scenario for extended design changes is developed for this purpose (Figure 6-17). The scenario intends to include several design change alternatives that relate to each other differently. Some of these design change alternatives are dependent on the choice in a previous alternative. For example, design change alternative [(A5) or (A6)] is feasible only if the design change (H2) is chosen out of design change alternative [(H1) or (H2)]. On the other hand, design change alternative [(A5) or (A6)] is independent of design change alternative [(H3) or (H5)], yet they coexist in parallel. The scenario also includes repeated design change alternatives like [(H3) or (H5)]. The purpose for developing the scenario with these features is to make sure that the specially developed subroutines (section 5.4.2.2) function correctly in all possible scenarios.

The virtual change module is used to initiate a new case (similar to Figure 5-11 on page 107) and to collect the data about all the design changes in the developed scenario. The planning and scheduling module is then run to generate the various possible change-paths for the scenario. Figure 6-18 to Figure 6-23 present the result as produced by the planning and scheduling module. The model is able to recognize all the possible change-paths and to calculate the needed design time, design cost, impact on building cost and the level of recommendation for every change-path. In change-paths number three (Figure 6-20) and number five (Figure 6-22), the levels of recommendation are calculated with the result of
the value zero. This indicates to the design manager that both of these change-paths cannot be used. It is also shown that the reason is design change (A5), that is change in the slab to slab height. The architect assigns a level of recommendation of zero to this design change because, if it is pursued, will result in exceeding the maximum permissible building height according to the code.

The module also generates a comparison table among all the alternate change-paths as shown in Figure 6-24 to help the design manager in selecting the best change-path. All the generated bar charts and the calculations are identical to those performed manually for the scenario, only much faster. As such, the model proves to be capable of providing the expected planning and scheduling capability.
Change in:
A1. Space function
(from executive office to
computer lab).
A3. Height of façade panel.
A5. Slab to slab height.
A6. Reduce clear height.
L1. Luminaire type.
L2. Luminaire x dimension.
L3. Location of luminaires.

Legend
Design Change
Alternate Design Change

H1. HVAC duct width.
H2. HVAC duct depth.
H3. Add HVAC duct.
H5. Size of HVAC outlet.
H6. Location of HVAC outlets.
I1. Wall finishing.
(from dark mahogany
wood to white paint)
I2. Ceiling pattern x size.

S1. Width of hole in
structural beam.
S2. Height of hole in
structural beam.
S3. Depth of beam.
S5. Beam section type.

F5. Location of sprinkler
outlets.

Figure 6-17 Scenario for extended design changes with several alternatives.
Figure 6-18 Bar chart and data for alternate change-path (1) for the scenario in Figure 6-17

Figure 6-19 Bar chart and data for alternate change-path (2) for the scenario in Figure 6-17
Figure 6-20 Bar chart and data for alternate change-path (3) for the scenario in Figure 6-17

Figure 6-21 Bar chart and data for alternate change-path (4) for the scenario in Figure 6-17
Figure 6-22 Bar chart and data for alternate change-path (5) for the scenario in Figure 6-17

Figure 6-23 Bar chart and data for alternate change-path (6) for the scenario in Figure 6-17
Figure 6-24 Comparison among various alternate change-paths for the scenario in Figure 6-17
6.3.4 Third Level of Validation

The third level is to validate that the model provides a better tool over current practice and help in solving the problem of design changes in a multi-disciplinary team. The validation is performed through using a real case study that is selected from literature (Fazio 1990). The case study exhibits a real situation where incompatibility in design information led to unexpected cost overruns of 2 million dollars in a 122m (400’) tall office building in a Canadian city. The building is designed using a column-and-beam steel structure system with reinforced concrete core and slabs. Figure 6-25 shows a plan view of a steel column with surrounding building components. The building envelope consists of metal air pans (component F in Figure 6-25) that are connected to the steel structure by metal clip angles (H). The function of the metal pans is to prevent airflow through the envelope. The external layer of the envelope consists of precast concrete panels (A) that are one story in height, spanning between the steel columns (B). The panels are supported on brackets that are connected to the steel beams. The windows (D) are supported by aluminum angles that are bolted to the concrete panels. Tracks that guide the window washing equipment are embedded on the exterior face of the precast panels and require strict vertical alignment of the precast panels. The columns are covered with sprayed-on fireproofing (C) and are encased by 13mm (1/2”) gypsum board on metal studs (E). The induction units (G) of the HVAC system span between the gypsum board encasements of adjacent exterior columns.

Three independent design disciplines, architecture, structural, and mechanical engineering
are involved in the design of the building components that appear in Figure 6-25. However, the design of these components is inter-dependent. The architect designs the precast concrete panels, the gypsum board encasement, the metal pan and clip angles. The design objective of the architect is to get maximum utilization of internal space and maximum window area. To do so, he runs the window up to the encasement and keeps the width of the encasement to a minimum. The vertical component of the precast concrete panel (A) is designed to have the same width as the gypsum board encasement (E) so that encasement does not overlap with the windows. Both of these widths, however, should cover the width of the column flange (B), the thickness of the fire proofing (C) and take into consideration the construction tolerance of the column. The spacing between the clip angles (H) needs to be as wide as possible to reduce the surface area of the metal pan (F) and consequently its cost. Yet, the design of this spacing should take into consideration the width of the column flange and the construction tolerance of the column. The width of the metal pan is defined so that the gypsum board encasement can be fixed to it. The following data is collected and used by the architect to design the building components under his responsibility:

- The width of the steel column flange is 356mm (14”).
- The maximum construction tolerance for the steel column is 13mm (0.5”).
- The thickness of the sprayed fireproofing is 13mm (0.5”).

The structural engineer designs the steel column (B) and later specifies its construction tolerance according to the code. The local code allows a maximum of 127mm (5”) for
construction tolerance at the top of the building. This indicates a clear conflict with the same data element as used by the architect and may push the column outside the encasement. The error can be attributed to either oversight by the architect or poor communication. The structural engineer also specifies the type and the thickness of the fire proofing (C) for the column. A thickness of 13mm (0.5") is found to be inadequate and is later modified to 36mm (1 7/16"), consequently invalidating the value used by the architect for the same data element. The HVAC engineer designs the induction unit (G) to fit between two adjacent column encasements in accordance with the width of the encasement as given by the architect.

During construction, problems start to appear. The first problem is the connection between the welded clip angles (H) and the column flange. The strict vertical alignment of the angles conflicts with the construction tolerance of the columns. At high stories, one side of the angles would fall off the flange. The connection is redesigned and a change order is issued to use C-shape (component J in Figure 6-26) instead of the clip angles. The gypsum board encasement as designed does not cover the column and the sprayed fireproofing. A change order is issued to make the encasement wider. However, the metal pan is not wide enough for fixing the encasement. The issued change order requires the use of Z-clips (M) to support gypsum boards. The extra width of the encasement makes it impossible to fit the induction units in their place. Because these units are already delivered to site, a change order is issued to trim their length. These series of change orders also result in several construction delays. The contractors make a claim of $6
millions for cost overruns due to conflicts in design. The case is eventually settled out of
court for $2 millions.

The proposed information model is used for this design case to examine the model ability
to overcome such design incompatibility errors as described. The first step in using the
model is to assemble the data-structure of the building components using the
configuration module. The ability of the model to flexibly adapt to any design
configuration makes it possible for the design manager to develop such an assembly.
Prebuilt and dynamically built rules are then used to capture the linking knowledge
among the involved disciplines. The following knowledge is captured using prebuilt
rules:

- The architect uses construction tolerance of the column as an input data to design
  the connection of the envelope to the structure and to size the encasement of the
column (Figure 6-27).

The following knowledge is captured using dynamically built rules:

- The architect sizes the width of the gypsum board column encasement (E) to be
  the same as the width of the precast concrete panel (A) and based on the following
data:
  - A width of 356mm (14") for the flange of the steel column.
  - A thickness of 13mm (0.5") for the sprayed fireproof (Figure 6-28).
The HVAC engineer sizes the length of the induction unit (G) based on the width of the gypsum board encasement and the column spacing.

The design changes that occurred in the case study are then performed in the model. The structural engineer increases the thickness of the fire proofing from a value of 13mm (0.5") to an adequate value of 36mm (1 7/16"). Consequently, two messages are automatically sent to the architect informing him about the change. The first message (Figure 6-29) shows that the change affects the width of the gypsum board encasement and the second message shows that the change affects the width of the precast concrete panel (A). Accordingly, the architect increases the width of these two components along with the width of the metal pan (F) to enable fixing of the gypsum board encasement. However, as soon as the architect changes the width of the gypsum board encasement, a message is sent automatically to the HVAC engineer informing about the change and indicating that the length of the induction unit (G) is affected. Similarly, when the structural engineer specifies the value of the construction tolerance for the steel columns, a message is automatically sent to the architect. The message shows the correct value of the tolerance, informing that this value affects the design of the connection between the envelope and the structure and also the width of the encasement. The architect then uses the correct tolerance value and redesigns the connection using C-shape instead of the clip angles. He also resizes the width of both the encasement and the metal pan to accommodate the expected construction tolerance. Another message is then sent to the HVAC engineer automatically informing him about the new width of the encasement and
indicating that the length of the induction unit (G) is affected. The HVAC engineer then corrects the length of this unit.

This case study demonstrates how small elements of data can easily be missed in current design practice. These result in incompatibilities in design information and lead to change orders and unnecessary increase in the building cost and project duration. Using the model, design changes are accommodated before starting building construction. The success of the model largely depends on the linking knowledge that is captured from designers. However, the model provides the design team with the ability to record and make use of this knowledge. In current practice, such knowledge is scattered around, unused and/or completely wasted throughout the design process. The case study proves that the use of the model provides a better tool for designers in multi-disciplinary design environment to accommodate design changes. Consequently, the related problems that appear during construction are avoided.

6.4 SUMMARY AND CONCLUSIONS

The model is validated through three levels of validation. Hypothetical design scenarios ensured that the model as it is developed and implemented can provide the expected capabilities in managing design changes. A real case study proved that the model can improve the current design practices and help building design professionals eliminate problems related to incompatibility in design information.
Figure 6-25 Plan view of an exterior column and the surrounding components as designed (Fazio 1990 reproduced)
Figure 6-26 Plan view of the exterior column and the surrounding components as constructed (Fazio 1990 reproduced)
Figure 6-27 Prebuilt rule to capture link between construction tolerance and design of envelope and encasement.
Figure 6-28 Dynamically built rule to capture link between width of the gypsum board encasement and fireproofing thickness\(^1\).

\(^1\) Units in Figure 6-28 and Figure 6-29 are in Inches.
Figure 6-29 Automated message to inform architect about change in thickness of fireproofing.
CHAPTER SEVEN
SUMMARY, CONTRIBUTIONS, AND RECOMMENDATIONS

7.1 SUMMARY

Incompatibilities in construction technical documents represent a major problem for the construction industry. An analysis of the production process of these documents during the detailed design stage reveals that managing design changes is a main cause for incompatibilities. More specifically, propagation of design changes is a principal source of problems. The large amount of data that is generated within a multi-disciplinary design environment makes this task very complex, especially when considering the fact that the involved design disciplines are, in most cases, separated both spatially and educationally.

This thesis presents the development of a model that uses information technology to address the problem. The core concept of the model is a central database that functions as a repository of active building components. Each building component in this database not only carries its design data, but is also capable of recognizing the design disciplines that are affected by any change in these data and automatically send them messages. As such, the complex task of propagating design changes is assigned to the building components themselves. The components efficiently perform the task regardless of the amount of data or the number of disciplines involved in a project. Notification messages are automatically sent by any building component to only affected design disciplines as soon as a design change is made to any attribute of the building components. The messages

Ahmed Mokhtar, Ph.D Thesis ..... 159
clearly describe the design changes and advise each design discipline on how the change may influence its building system. An appropriate reaction is left to the designer who is now well informed. The active building components are able to perform this task because they are equipped with the necessary linking knowledge. The linking knowledge is acquired from the designers and is implemented in the form of rules.

The central database is composed of two parts. The first is the building-components-database which carries the active building components data. The second part is the management-database which carries data that is necessary for the various functions of the model. The concept of management-database allows the building-components-database to adapt easily to any building configuration, an essential requirement because buildings are unique products. The utilization of combined Object-Oriented and Relational Database technologies enables the model to achieve this flexibility.

The developed information model is also equipped with the capability to track past design changes. It can answer the requests of design managers to track past design changes in a variety of ways. It can also show the automated messages that are sent by a building component to each affected discipline as a reaction to a design change. The response of the informed discipline to each of these messages can also be tracked.

The model is also developed to facilitate the planning and scheduling of future design changes. A methodology is incorporated within the model and uses its active components to collect future design change data. This data is organized using especially developed
sub-routines, then scheduled and manipulated with some calculations that provide the design manager with a clear view on how to conduct a future design change.

The information model is successfully implemented in a client/server setup. It is subjected to three levels of validation. The validation process ensured not only that the model provides the capabilities that are expected from its development but also that the model is a better tool over current practices for managing design changes in a collaborative multi-disciplinary design environment.

7.2 CONTRIBUTIONS

Aiming to improve the quality of buildings, the contributions of this research towards eliminating the serious problems associated with incompatibility errors in construction technical documents can be summarized as follows:

- A thorough analysis of the production process of construction technical documents with an emphasis on the techniques for coordinating design information so as to accommodate design changes. The analysis diagnoses the main problems that cause incompatibilities in design information. These problems are the management of design changes, the media used to communicate design information, and the source of final documents.

- Review of research work in the area of using computer technology in collaborative design. The review includes an analysis of approaches used in these researches so as
to evaluate their capabilities and limitations to eliminate incompatibilities in design information.

- Development of an information model that establishes the theoretical basis for a computer-aided collaborative multi-disciplinary design environment that facilitates the management of design changes. The model provides remedy for problems with current production processes of construction technical documents and helps ensure the compatibility of design information scattered among these documents. The model, being the first of its kind, not only helps accommodating design changes but also tracks past design changes and helps planning and scheduling of future ones as well. In addition, the model has the flexibility to include any configuration of different building components along with the knowledge that connect them. Hence, it is able to function with any type of building. The model, however, does not cover all the information needed for the design of a building such as codes and design constraints. The model is also intended for sequential type of project delivery systems. It is not intended for other types of delivery systems such as fast track system. Designers who will use the model are expected to collaborate using client-server networks and therefore they should be located in the same building or nearby buildings. Other types of networks such as wide area networks, intranets, or internet are beyond the scope of this model.

- Development and implementation of the concept of active building components. Such a concept enables the management of design changes by assigning the task of propagating
design changes to the building components that are subject to change rather than the designers who perform these changes. The building components are made active by capturing the necessary linking knowledge from the designers. A process is developed and implemented to capture this knowledge in terms of prebuilt and dynamically built rules. The active components, however, do not use a knowledge-based system and therefore no induction is performed by the components.

- Development and implementation of the concept of management database. Such database manages the data-structure of the building-components-database, which carries the design data, and allows it to be customizable to fit any type of building. The management database also keeps track of all design changes as well as the messages that are automatically sent to the designers due to these changes. In addition, it carries the data necessary to plan and schedule future design changes.

- Development and implementation of the concept of planning and scheduling future design changes. To realize this concept, a methodology has been outlined to utilize the capabilities of active building components in collecting design change data from designers so as to enable the planning of extended design changes. Subroutines have been created to translate the collected data into organized design change paths that show the design manager all the possible paths to perform a design change. It also shows the time needed, the design cost, the impact on building cost, and the recommendation level for each of these change paths.
• Validation of the developed model not only through hypothetical scenario but also with a real design case. The hypothetical scenario proves that the model provides the capabilities that are expected from its development while the real design case shows that the model improves the current design practices, which is the ultimate objective of developing the model.

7.3 **RECOMMENDATIONS FOR FUTURE WORK**

Due to the fast technological development in this field of research, many new avenues can be explored to apply and evolve the concepts that are presented in this thesis. At the time of writing these recommendations, we can see potential developments as follows:

• Explore the use of the internet as the carrier of the central project-database and as the medium for automated messages propagation.

• Develop multimedia messages that provide clearer contents with more information.

• Apply the standards of IFCs (IAI·1996) and STEP (Froese 1996) for the structure of the building-components-database.

Some complementary components of the information model have not been developed in this research due to resource limitations. Other researchers can work on these developments as:

• The construction technical documents generation module. This module is responsible for generating, from the project-database, the documents that are used by the
contractors to construct the building. The module needs to have the following features:

1. Document formats and classifications need not to be based on disciplines any longer e.g., architectural drawings and structural drawings. Some documents could be prepared on the basis of the requirements of the contractors and the subcontractors so as to make their job easier and with fewer errors (see Mokhtar and Bédard 1994).

2. The output medium for construction technical documents needs not to be paper anymore; it can be a laptop connected through a wireless modem to the central database that lies on the internet.

- A drafting interface that facilitates the manipulation of graphical data.
- A mechanism to control the access to the project-database by various users.

More development for some of the model main components is also recommended. The following is advocated:

- More sophisticated relation-acquisition capability that can acquire and incorporate design rationale.
- Automated control of the design changes after an alternative is chosen through the planning and scheduling capability of the model.
- Connect the results of the planning and scheduling to a project management software such as Primavera (1996).


A.1 MANAGEMENT-DATABASE

A.1.1 CONCEPTUAL DATA-STRUCTURE

Figure A-1 Conceptual data-structure for management-database presented in EXPRESS-G.
### A.1.2 Structure of Data-Tables

**Data-Table 1:** MNG-BUILDING-COMPONENTS

- ID
- BUILDING_COMPONENT
- ATTRIBUTE_OF_BUILDING_COMPONENT
- ATTRIBUTE_TYPE
- DESIGNING_DISCIPLINE
- LINKED_BUILDING_COMPONENT
- LINKED_ATTRIBUTE_OF_BUILDING_COMPONENT
- GRAPH_NAME

**Data-Table 2:** MNG-MENU-CONFIGURATION

- ID
- LABEL
- MAIN_MENU_LABEL

**Data-Table 3:** MNG-PREBUILT-RULES

- PREBUILT_RULE_ID
- BUILDING_COMPONENT
- ATTRIBUTE_OF_BUILDING_COMPONENT
- AFFECTED_DISCIPLINE
- PERCENT_CHANGE_TO_HAVE_AN_EFFECT
- RESULT_IN_MAJOR_MODIFICATION?
- RULE_IGNORED?
- CAUSE_OF_THE_EFFECT

**Data-Table 4:** MNG-DYNAMICALLY-BUILT-RULES

- DYNAMICALLY_BUILT_RULES_ID
- BUILDING_COMPONENT
- BUILDING_COMPONENT_ID
- ATTRIBUTE_OF_BUILDING_COMPONENT
- RELATED_BUILDING_COMPONENT
- RELATED_BUILDING_COMPONENT_ID
- RELATED_ATTRIBUTE
- RELATIONSHIP
- WRITING_DATE
- RULE_IGNORED?
<table>
<thead>
<tr>
<th>DATATABLES</th>
<th>MNG-DESIGN-CHANGES</th>
<th>DATATABLES</th>
<th>MNG-VIRTUAL-CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CHANGE_ID</td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td>DATE_TIME</td>
<td></td>
<td>CASE_ID</td>
</tr>
<tr>
<td></td>
<td>DISCIPLINE</td>
<td></td>
<td>CHANGE</td>
</tr>
<tr>
<td></td>
<td>BUILDING_COMPONENT</td>
<td></td>
<td>BUILDING_COMPONENT</td>
</tr>
<tr>
<td></td>
<td>BUILDING_COMPONENT_ID</td>
<td></td>
<td>BUILDING_COMPONENT_ID</td>
</tr>
<tr>
<td></td>
<td>ATTRIBUTE</td>
<td></td>
<td>ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>OLD_VALUE</td>
<td></td>
<td>REQUIRED_MAN_HOURS</td>
</tr>
<tr>
<td></td>
<td>NEW_VALUE</td>
<td></td>
<td>DIFFERENCE_IN_BUILD_COST</td>
</tr>
<tr>
<td></td>
<td>PRECEDEING_CHANGE_ID</td>
<td></td>
<td>RECOMMENDATION_LEVEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>REASON_FOR_RECOMMENDATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DURATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OLD_VALUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NEW_VALUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATATABLES</td>
<td>MNG-MESSAGE-ELEMENTS</td>
<td>DATATABLES</td>
<td>MNG-VIRTUAL-MESSAGE-ELEMENTS</td>
</tr>
<tr>
<td></td>
<td>ID</td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td>CHANGED_ATTRIBUTE</td>
<td></td>
<td>CHANGED_ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>CHANGED_BUILDING_COMPONENT</td>
<td></td>
<td>CHANGED_BUILDING_COMPONENT</td>
</tr>
<tr>
<td></td>
<td>CHANGED_BUILDING_COMPONENT_ID</td>
<td></td>
<td>CHANGED_BUILDING_COMPONENT_ID</td>
</tr>
<tr>
<td></td>
<td>OLD_VALUE</td>
<td></td>
<td>OLD_VALUE</td>
</tr>
<tr>
<td></td>
<td>NEW_VALUE</td>
<td></td>
<td>NEW_VALUE</td>
</tr>
<tr>
<td></td>
<td>AFFECTED_ATTRIBUTE</td>
<td></td>
<td>AFFECTED_ATTRIBUTE</td>
</tr>
<tr>
<td></td>
<td>AFFECTED_BUILDING_COMPONENT</td>
<td></td>
<td>AFFECTED_BUILDING_COMPONENT</td>
</tr>
<tr>
<td></td>
<td>AFFECTED_BUILDING_COMPONENT_ID</td>
<td></td>
<td>AFFECTED_BUILDING_COMPONENT_ID</td>
</tr>
<tr>
<td></td>
<td>RELATIONSHIP</td>
<td></td>
<td>RELATIONSHIP</td>
</tr>
<tr>
<td></td>
<td>WRITING_DATE</td>
<td></td>
<td>WRITING_DATE</td>
</tr>
<tr>
<td></td>
<td>BEEN_READ_</td>
<td></td>
<td>BEEN_READ_</td>
</tr>
<tr>
<td></td>
<td>BEEN_ACTED_UPON_</td>
<td></td>
<td>BEEN_ACTED_UPON_</td>
</tr>
<tr>
<td></td>
<td>DISCIPLINE_TO</td>
<td></td>
<td>DISCIPLINE_TO</td>
</tr>
<tr>
<td></td>
<td>DISCIPLINE_FROM</td>
<td></td>
<td>DISCIPLINE_FROM</td>
</tr>
<tr>
<td></td>
<td>SENT_BECUSE_CHANGE_ID</td>
<td></td>
<td>SENT_BECUSE_CHANGE_ID</td>
</tr>
<tr>
<td></td>
<td>PREBUILT_OR_Dynamically_?</td>
<td></td>
<td>PREBUILT_OR_Dynamically_?</td>
</tr>
<tr>
<td></td>
<td>RULE_WRITING_DATE</td>
<td></td>
<td>RULE_WRITING_DATE</td>
</tr>
<tr>
<td></td>
<td>VIOLATED_RULE_ID</td>
<td></td>
<td>VIOLATED_RULE_ID</td>
</tr>
<tr>
<td></td>
<td>RULE_IGNORED_?</td>
<td></td>
<td>RULE_IGNORED_?</td>
</tr>
<tr>
<td></td>
<td>CAUSE_OF_THE_EFFECT</td>
<td></td>
<td>CAUSE_OF_THE_EFFECT</td>
</tr>
</tbody>
</table>

*Ahmed Mokhtar, Ph.D Thesis......175*
<table>
<thead>
<tr>
<th>ID_NUMBER</th>
<th>CASE_ID</th>
<th>DESIGN_CHANGE</th>
<th>SUCCESSOR</th>
<th>ALTERNATE_SUCCESSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>DESIGN_CHANGE</th>
<th>SUCCESSOR</th>
<th>ALTERNATIVE#</th>
<th>CASE_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.2 GENERAL-DATABASE

A.2.1 CONCEPTUAL DATA-STRUCTURE

Figure A-2 Conceptual data-structure for general-database presented in EXPRESS-G.
A.2.2 Structure of Data-Tables

<table>
<thead>
<tr>
<th>Dat-Table A.11</th>
<th>GN-L-BUILDING-COMPONENTS ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>BUILDING_COMPONENT</td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE_OF_BUILDING_COMPONENT</td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE_TYPE</td>
<td></td>
</tr>
<tr>
<td>DESIGNING_DISCIPLINE</td>
<td></td>
</tr>
<tr>
<td>LINKED_BUILDING_COMPONENT</td>
<td></td>
</tr>
<tr>
<td>LINKED_ATTRIBUTE_OF_BUILDING_COMPONENT</td>
<td></td>
</tr>
<tr>
<td>GRAPH_NAME</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dat-Table A.12</th>
<th>GN-L-PREBUILT-RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREBUILT_RULE_ID</td>
<td></td>
</tr>
<tr>
<td>BUILDING_COMPONENT</td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE_OF_BUILDING_COMPONENT</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>AFFECTED_DISCIPLINE</td>
<td></td>
</tr>
<tr>
<td>PERCENTCHANGE_TO_HAVE_AN_EFFECT</td>
<td></td>
</tr>
<tr>
<td>RESULT_IN_MAJOR_MODIFICATION</td>
<td></td>
</tr>
<tr>
<td>RULE_IGNORED</td>
<td></td>
</tr>
<tr>
<td>CAUSE_OF_THE_EFFECT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dat-Table A.13</th>
<th>GN-L-BUILDING-SYSTEMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>BUILDING_SYSTEM</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>DISCIPLINE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dat-Table A.14</th>
<th>GN-L-COMPONENTS OF BUILDING-SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>BUILDING_SYSTEM</td>
<td></td>
</tr>
<tr>
<td>BUILDING_COMPONENT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dat-Table A.15</th>
<th>GN-L-CURRENT-PROJECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>PROJECT_NAME</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX B

SUBROUTINES FOR PLANNING AND SCHEDULING DESIGN CHANGES

B.1 SUBROUTINE TO GENERATE THE VARIOUS CHANGE-PATHS.

B.1.1 INITIALIZATIONS

Sub1-A. Connect to Data-Table 5-6 (page 111) and Data-Table 5-8 (page 113) of the related project-database.

Sub1-B. Limit the reading of the data from Data-Table 5-6 to the rows that have the specified value for the attribute “CASE_ID” (i.e. CASE_ID = 2).

Sub1-C. Create two lists for “Processed design changes” and “Under processing design changes”.

B.1.2 GENERATE THE FIRST CHANGE-PATH

Sub1-D. Query Data-Table 5-6 (page 111) for the data row that have the value “NULL” in the attribute “DESIGN_CHANGE” and read the value of the attribute “SUCCESSOR” in the found row. This value is the initial design change that has been defined when a designer starts a new case (i.e. A1 in Figure 5-15 on page 111). Add this design change (i.e. A1) to both the list of “Processed design changes” and the list of “Under processing design changes”.

Sub1-E. Save this data in Data-Table 5-8 (page 113) with the attribute “ALTERNATIVE#” given the value “ALTERNATIVE#1”. The value for
the attribute "ID" is automatically generated in the database. Sample for
the saved data is in the row with ID = 4734 in Figure 5-17 (page 113).

For each design change in the list of "Under processing design changes", perform
steps Sub1-F to Sub1-H:

Sub1-F. Query from Data-Table 5-6 (page 111) all the data rows which have the
value of the attribute "DESIGN_CHANGE" equals the under processing
design change (i.e. A1). Read the corresponding values in the attribute
"SUCCESSOR" (i.e. H1, L1, I1 in Figure 5-15 on page 111).

Sub1-G. Save the data in Data-Table 5-8 (page 113) for each read successor similar
to the saving process in step Sub1-E. Samples are rows with ID = 4735,
4736, 4737 in Figure 5-17 on page 113.

Sub1-H. For each read successor in Sub1-F, check if the successor exists in the list
of "Processed design changes".
- If it exists then ignore it.
- If it does not exist then add it to the list of "Processed design changes"
and the list of "Under processing design changes". Figure B-1 shows
the result of applying steps Sub1-D till Sub1-H on the example given
in Figure 5-10 on page 105.
Figure B-1 Result of applying steps Sub1-D till Sub1-H on example given in Figure 5-10.

B.1.3 GENERATE THE OTHER CHANGE-PATHS

Sub1-I. Read sequentially the design changes in the last generated change-path starting from the end design changes (i.e. F5, S1, or L1 in Figure B-1) to the first design change.

Sub1-J. For each design change that is read, check using Data-Table 5-6 (page 111) if the design change has an alternate successor:

- If it does not have an alternate successor (i.e. S1) then ignore this design change.
- If it has one but the last generated change-path uses the value of the alternate successor then ignore this design change.
- If it has an alternate successor (i.e. A1) and the last generated change-path uses the value of the successor, then repeat steps Sub1-D to Sub1-H to generate the next change-path with the use of the "ALTERNATE_SUCCESSOR" for this change (i.e. H2 in Figure 5-15 on page 111) instead of the "SUCCESSOR".

In case that the second option is the valid one, the value of the
"SUCCESSOR" - not the "ALTERNATE_SUCCESSOR" - is used the next time this design change passes through steps Sub1-D to Sub1-H. Figure B-2 shows the result of applying steps Sub1-I and Sub1-J on the example given in Figure 5-10 on page 105.

Repeat step Sub1-I and Sub1-J till all the possible change-paths are generated as shown in Figure 5-17 (page 113). Figure B-4 shows a flowchart for the subroutine.

![Diagram of change-paths](image)

Figure B-2 Result of applying steps Sub1-I and Sub1-J on example given in Figure 5-10.

### B.2 SUBROUTINE TO GENERATE THE LOGICAL SEQUENCE OF DESIGN CHANGES.

The logic for developing this subroutine is based on providing a start node value and an end node value for each of the design changes in a certain change-path (Figure 5-18 on page 114). This is to satisfy the requirement for using a CPM algorithm that can schedule the design changes. The following rules apply when defining the values for the start node and the end node:

- The value of the start node of a design change is always less than the value of its end node.

- The value of the start node of a successor to a design change is the same value of the
end node of the design change.

- More than one design change can share the same value for the start node (i.e. H2 and I1 in Figure 5-18b on page 114).

- The value of the end node is always unique. This rule is not required for using the CPM algorithm, rather to support the logic of the subroutine.

- Dummy design changes are used (shown as dotted lines with arrow heads in Figure 5-18) to reflect the case where two design changes share the same successor (i.e. I1 and A2 in Figure 5-18b).

Figure B-3 represents a matrix that is used to represent the logic for the developed subroutine and reflects the previous rules. The vertical values of the matrix indicate the start nodes of design changes while the horizontal values indicate the end nodes. Figure B-3 reflects the change-path in Figure 5-18b.

```
<table>
<thead>
<tr>
<th>Values of start nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
</tr>
<tr>
<td>H2</td>
</tr>
<tr>
<td>I1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Values of end nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>A2</td>
</tr>
<tr>
<td>L1</td>
</tr>
</tbody>
</table>
```

Figure B-3 Matrix to represent start nodes and end nodes for change-path in Figure 5-18b.
B.2.1 Initialization

Sub2-A Connect to Data-Table 5-8 (page 113) of the related project-database.

Sub2-B Limit the reading of the data from Data-Table 5-8 to the rows that have the specified value for the attribute “CASE_ID” (i.e. CASE_ID = 2).

Sub2-C Create two lists for “Processed design changes” and “Under processing design changes”. Each element in these lists has attributes for the value of the start node and the value of the end node.

Repeat the following steps for each group of design changes in Data-Table 5-8 that has the same “ALTERNATIVE#” (i.e. Alternative#1).

B.2.2 Process the Initial Design Change

Sub2-D Query Data-Table 5-8 (page 113) for the data row that have the value “NULL” in the attribute “DESIGN_CHANGE” and read the value of the attribute “SUCCESSOR” in the found row. This value is the initial design change that has been defined when a designer starts a new case (i.e. A1 in Figure 5-17 on page 113). Add this design change to the list of “Processed design changes” and the list of “Under processing design changes”.

Sub2-E Make the value of the start node = 1 for the initial design change and make a unique consecutive value (i.e. 2) for the end node.

B.2.3 Process the Rest of the Design Changes in the Change-Path.

For each design change in the list of “Under processing design changes” perform the steps Sub2-F to Sub2-I:
Sub2-F Read the value of the end node for this design change (i.e. 2 for the design change A1). Assign this value to the variable (e).

Sub2-G Get from Data-Table 5-8 on page 113 all the successors of the design change (i.e. H1, L1, I1 in Figure 5-17 on page 113).

For each successor, perform steps Sub1-H to Sub1-I:

Sub2-H Read from Data-Table 5-8 (page 113) all the design changes that have this successor. (i.e. design changes A1 and I1 for the successor L1).

Sub2-I Check if all the found design changes in Sub2-H (i.e. A1 and I1) exit in the list of “Processed design changes”.

• If any of the found design changes does not exist then ignore this successor. (i.e. L1)

• If all of the found design changes exit then add this successor to the list of “Processed design changes” and the list of “Under processing design changes”. Make the value of the start node for this successor equals the value of the end node of the under processing design change. (i.e. the value of the variable (e) which is defined in step Sub2-F) and use a unique consecutive value for the end node of this successor. Using unique consecutive value for the end nodes makes use of all the rows in the matrix of Figure B-3 and ensures that only one design change exists in every row hence reflecting the stated rules at the beginning of section B.2 on page 182.
B.2.4 GENERATE THE DUMMY DESIGN CHANGES

Sub2-J For the value of a variable (i) that starts from the value (1) to the highest value used as a start node in the processed design change (i.e. 1 to 7 in Figure B-3 on page 183), check if the value is used as a start node in any of the processed design change. This is similar to searching for the empty columns in the matrix in Figure B-3.

- If the value of (i) is used (e.g. 1), then check the next value.
- If the value of (i) is not used (e.g. 4), then continue the following steps.

Sub2-K Find the design change in the list of processed design changes that have the same value of (i) for its end node (e.g. I1). This should be a single design change as the end node is a unique value for every design change. This is demonstrated in the matrix in Figure B-3 by having only one design change in any row. The matrix also shows that there are no empty rows.

Sub2-L Query Data-Table 5-8 (page 113) for all the successors of the found design (e.g. I1) change and determine - from the list of processed design changes - the successor that has the lowest start node value (i.e. L1). Assign this value (i.e. 7) to the variable (ii).

Sub2-M Create a dummy design change that has a start node value equals the value of (i) (i.e. 4) and an end node value equals the value of (ii) (i.e. 7). Add the dummy design change to the list of processed design changes.
After generating all the dummy design changes Figure 5-18 on page 114, arrange all the design changes in the processed design change list according to their start node value. The design change of this change-path is ready to be scheduled using a standard CPM algorithm. Figure B-5 shows a flowchart for the described subroutine.
Figure B-4 Flowchart for sub-routine developed to generate various change-paths.
Figure B-5 Flow chart for sub-routine developed to define sequence of performing design changes so as to eliminate redundancy.
C.1. Configuration Module

EXEC SQL:
SELECT * 
FROM [AB-attribute-designer] where building_element = 'current_rule_data.building_element'
END SQL INTO
building_element_attribute
END
WITH attribute_of_building_element STRING
WHEN CHANGED
BEGIN
EXEC SQL:
SELECT * 
FROM [AB-general-rules] where building_element = 'current_rule_data.building_element' AND
attribute_of_building_element = 'current_rule_data.attribute_of_building_element'
END SQL INTO AB_general_rules
END
WITH affect_a_discipline STRING
WHEN result_in_major_modification STRING
WHEN cause_of_the_effect STRING
WITH percent_change NUMERIC
WITH rule_ignored STRING

CLASS Building_element
WITH graph_name STRING
WHEN CHANGED
BEGIN
EXEC SQL:
SELECT * 
FROM [AB-attribute-designer] where building_element = 'current_data' AND
building_element_attribute = 'linked_building_element'
END SQL INTO
linked_building_element_attribute
BEGIN
EXEC SQL:
SELECT * 
FROM [AB-attribute-designer] where building_element = 'current_data' AND
building_element_attribute = 'linked_building_element'
END SQL INTO
linked_building_element_attribute
END
WITH affect_a_discipline STRING
WHEN result_in_major_modification STRING
WHEN cause_of_the_effect STRING
WITH percent_change NUMERIC
WITH rule_ignored STRING

CLASS edge_table
WHITH attachment child_abstract
WHITH column INSTANCE REFERENCE
WHITH column INSTANCE REFERENCE
WHITH heading SIMPLE
WHITH true SIMPLE
WHITH link cell NUMERIC
WHITH row_count NUMERIC
WHITH array size 2
INT [1] [1]
INT [2] [1]
WHITH true SQL INTO
WHITH highlight test color COLOR
WHITH highlight fill color COLOR
WHITH display only SIMPLE
WHITH selected SIMPLE
WHITH double clicked SIMPLE

CLASS edge_table_column
WHITH attachment ATTRIBUTE REFERENCE
WHITH heading label STRING
WHITH column style INSTANCE REFERENCE
WHITH column width NUMERIC
WHITH max characters NUMERIC
WITH format STRING
WHITH justify COMPOUND
WHITH left, center, right
WHITH pen color COLOR
COLLECTION
WHITH fill color COLOR
COLLECTION

CLASS GENERAL_Buiding_element
WHITH building_element STRING
WHITH graph_name STRING

CLASS GENERAL_Buiding_element_attribute
WHITH attribute_of_building_element STRING
WHITH attribute_type STRING
WHITH result_in_major_modification STRING
WHITH cause_of_the_effect STRING
WHITH percent_change NUMERIC
WHITH rule_ignored STRING

Ahmed Makhtar, Ph.D Thesis... 190
BEGIN
  attribute_type OF current_data := "v"
  attribute_type OF building_element_attribute := "v"
END
  IF attribute_type OF input_data IS Numeric data THEN
    BEGIN
      attribute_type OF current_data := "v"
      attribute_type OF building_element_attribute := "v"
    END
  END
END

EXEC 150e8c 1 SQL INSERT into [AB-menu_item] (label, main_menu_label) values
  (Menu manipulation data.current_menu_B_comp, Menu manipulation data.current_menu_group) END SQL
  EXEC 150e8c 1 SQL SELECT label from [AB-menu_item] WHERE main_menu_label = Menu manipulation data.current_menu_group END SQL INTO Menu_group_items
  FIND Menu_unclassified_B_comp WHERE label OF Menu_unclassified_B_comp = current_menu_B_comp OF Menu manipulation data WHEN FOUND FORGET current_menu_B_comp OF Menu manipulation data
  *** end of IF END ELSE BEGIN test OF message box 2 := "A selected Menu Group or Building Component is missing" put up OF message box 2 := TRUE END
  *** end of method END

EXEC 150e8c 1 SQL DELETE from [AB-menu_item] WHERE label = Menu manipulation data.current_menu_B_comp END SQL
  EXEC 150e8c 1 SQL SELECT label from [AB-menu_item] WHERE main_menu_label = Menu manipulation data.current_menu_group END SQL INTO Menu_group_items
  MAKE Menu_unclassified_B_comp label := current_menu_B_comp OF Menu manipulation data
  FORGET current_menu_B_comp OF Menu manipulation data
  *** end of IF END ELSE BEGIN test OF message box 2 := "A selected Menu Group or Building Component is missing" put up OF message box 2 := TRUE END
  *** end of method END

BEGIN
  WITH attribute_type OF input_data := 0 THEN
    BEGIN
      IF attribute_type OF input_data IS Text data THEN
        BEGIN
          attribute_type OF current_data := "v"
          attribute_type OF building_element_attribute := "v"
        END
      END
END

CLASS input data
  WITH discipline COMPOUND
    Architect, Structure, HVAC, Illuminat, Interior, Envelope
WHEN CHANGED BEGIN
  IF (CONF(discipline OF input_data)) > 0 THEN BEGIN
    IF discipline OF input_data IS Architect THEN BEGIN...
      building_element_attribute := "arch"
      designing_discipline := current_data := "arch"
    END
    IF discipline OF input_data IS Structure THEN BEGIN...
      building_element_attribute := "stru"
      designing_discipline := current_data := "stru"
    END
    IF discipline OF input_data IS HVAC THEN BEGIN...
      building_element_attribute := "hvac"
      designing_discipline := current_data := "hvac"
    END
    IF discipline OF input_data IS Illuminat THEN BEGIN...
      building_element_attribute := "illu"
      designing_discipline := current_data := "illu"
    END
    IF discipline OF input_data IS Interior THEN BEGIN...
      building_element_attribute := "inter"
      designing_discipline := current_data := "inter"
    END
    IF discipline OF input_data IS Envelope THEN BEGIN...
      building_element_attribute := "enve"
      designing_discipline := current_data := "enve"
    END
  END
END

WITH attribute type COMPOUND
  Text data, Numeric data
WHEN CHANGED BEGIN
  IF CONF(attribute_type OF input_data) > 0 THEN BEGIN
    IF attribute_type OF input_data IS Text data THEN...
Building element) 
BEGIN 

** i 1; create the database table with its id attribute 
** general statement of sql = CONCAT("CREATE TABLE ", building_element_name OF Building element, ", [", "attribute", "OF Building element, ", "", "text", "50", ");"

EXEC code 1 SQL WITH statement of sql END SQL END 

! i 1; read the attributes of the components from the other project FOR (i = 1 TO number instances OF navigate 
Building element) 
BEGIN 
instance number OF navigate Building element 

** i 1; the next step is to overcome the same 
"Building element" which can not be linked in Access 
= building_element OF Building element 
EXEC code 1 SQL SELECT * FROM [AB- 
attribute-designer] WHERE building_element = 

END SQL INTO building_element_attribute 

FOR (i = 1 TO number instances OF navigate building 
Building element attributes) 
BEGIN 
instance number OF navigate building 
Building element attributes 

** i 1; ignore the attribute that has the _id_ of 
attribute_of_building_element OF 
building_element_attribute = 0 THEN 
BEGIN 

** i 1; put the attributes in the database 
general statement of sql = CONCAT(" 
alter table ", building_element_name OF Building element, ", [", "attribute", "OF Building element, ", "", "text", "50", ");"

WHERE building_element_attribute = _id_ 
EXEC code 1 SQL SELECT * FROM [AB- 
attribute-designer] WHERE building_element = 

END SQL INTO building_element_attribute 

END SQL INTO AB_general_rules 

BEGIN 
instance number OF rules navigate = 1 
EXEC code 1 SQL INSERT INTO [AB- 
general-rules] (building_element_attribute, 
attribute_of_building_element, affected_discipline, 
percentage_change, result_in_major_modification, 
cause_of_the_effect) VALUES 

END SQL INTO AB_general_rules 

END SQL INTO building_element_attribute 

FOR (i = 1 TO number instances OF navigate 
Building element attributes) 
BEGIN 
instance number OF navigate building 
Building element attributes = ii 
IF CONFI(linked_building_element OF 
building_element_attribute) = 100 THEN 
BEGIN 

** i 1; the following step is necessary as data 
can not be got directly from access to this attribute 
EXEC code 1 SQL SELECT 
(linked_attribute) FROM [AB- 
attribute-designer] WHERE building_element = 

END SQL INTO building_element_attribute 

END 

WITH add component to the new project SIMPLE
****** the following step is necessary as data can be lost if not directly accessed to this attribute
EXEC Inaquery 1 SQL SELECT (linked attributeazon, building element) from [GENERAL-AB-attribute-designer] where building_element = \\
:NEW_to_make_a_new_project.building_element end and attribute_of_building_element = \\
:building_element.attribute.attribute_of_building_element end
END SQL INTO domain (int)
linked_attributeazon, building element OF building_element.attribute.attribute_of_building_element end

EXEC Inaquery 1 SQL INSERT INTO [AB-attribute-designer] (building_element, attribute_of_building_element, attribute_type, \\
designing_discipline, linked_building_element, linked_attributeazon, building_element.attribute.attribute_of_building_element values \\
:NEW_to_make_a_new_project.building_element, \\
:building_element.attribute.attribute_of_building_element, NULL, NULL, NULL, NULL, NULL
)

SQL
END ELSE
BEGIN
EXEC Inaquery 1 SQL INSERT INTO [AB-attribute-designer] (building_element, attribute_of_building_element, attribute_type, \\
designing_discipline, linked_building_element, linked_attributeazon, building_element.attribute.attribute_of_building_element values \\
:NEW_to_make_a_new_project.building_element, \\
:building_element.attribute.attribute_of_building_element, NULL, NULL, NULL, NULL, NULL
)

SQL
END END

****** end of method
END

WITH finish components of the new project SIMPLE WHEN CHANGED
BEGIN

****** this method aids to add the general rules of the chosen components to the new project database
EXEC Inaquery 2 SQL SELECT distinct (building_element) from [AB-attribute-designer] end INTO building_element

FOR (i -> 1 TO number instances OF navigate_building_element) BEGIN

instance number OF navigate_building_element := i

a := building_element.attribute.attribute_of_building_element
EXEC Inaquery 2 SQL select * from [GENERAL-AB-general-rules] where building_element = domain.t
END SQL INTO AB_general_rules

****** add the rules to the new project database

FOR (g -> 1 TO number instances OF rules navigate 2) BEGIN

instance number OF rules navigate 2 := g
EXEC Inaquery 1 SQL INSERT INTO [AB-attribute-designer] (building_element, attribute_of_building_element, affected_discipline, percent_change, result_in_major_modification, cause_of_the_effect) values \\
(AB_general_rules.building_element, \\
AB_general_rules.attribute.attribute_of_building_element, \\
null, null, null, null)

SQL
END ELSE
BEGIN

IF attribute_type = "text" THEN
begin
begin
begin
begin
begin
BEGIN

 **** put the attributes in the database

 FOR project IN (SELECT id, attribute FROM [AB-attribute-designer]) BEGIN

instance number OF navigate_building_element := i

FOR (g -> 1 TO number instances OF navigate_building_element) BEGIN

instance number OF navigate_building_element := g

EXEC Inaquery 1 SQL WITH general statement of sql END SQL

****** add the rest of attributes to the created table

FOR (i -> 1 TO number instances OF navigate_building_element.attribute.attribute_of_building_element) BEGIN

instance number OF navigate_building_element.attribute.attribute_of_building_element := i

EXEC Inaquery 1 SQL WITH general statement of sql END SQL

****** this method aids to add the general rules of the components of the chosen system to the new project

Ahmed Mokhtar, Ph.D Thesis ..... 194
WHEN NONE FOUND
  i := 2
END

IF i = 1 THEN
  BEGIN
  text OF message box 2 := "There is already a building component with that name"
  put up OF message box 2 := TRUE
  FORGET new_component_name
  END

IF the name was not used before
  IF i = 2 THEN
    BEGIN
    i := building_element_OF_building_element
    EXEC SQL 2 SQL SELECT distinct (building_element) FROM (building_element_AREA) WHERE building_element = new_component_name
    END
  END SQL
END SQL
END
END

**** end of method

END

ATTRIBUTES Building Components SIMPLE
WHEN CHANGED
BEGIN
  action OF ISode 1 IS connect := TRUE
  EXEC ISode 1 SQL SELECT distinct (building_element, graph_name) FROM (building_element_AREA) WHERE building_element = new_component_name
END SQL INTO Building_element

EXEC ISode 1 SQL SELECT distinct (building_element) FROM (building_element_AREA) WHERE building_element = new_component_name
END SQL INTO Linked_building_element

visible OF main window := FALSE
output OF configuration window := config visible OF configuration window := TRUE

END

ATTRIBUTE new_component_name STRING
ATTRIBUTE new_component_name NUMERIC
ATTRIBUTE new_component_name STRING
BEGIN
  i := 0
  IF CONF(new_component_name) >= 0 THEN
    BEGIN
    FIND Building_element
    WHERE building_element OF Building_element = new_component_name
    WHEN FOUND
    i := 1
  END

BEGIN
  linked_building_element OF building_element ATTRIBUTE = linked_building_element OF current_data
  linked_attribute_type OF building_element ATTRIBUTE = current_data
  linked_attribute_type OF current_data ATTRIBUTE = current_data
  END

**** put in the database
  general statement of sql := CONCAT("ALTER table ", building_element_AREA.current_data, ")
  IF attribute_type OF current_data = "t" THEN
  general statement of sql := CONCAT("CREATE TABLE " , new_component_name, " (" , new_component_name, ")
  EXEC ISode 1 SQL WITH general statement of sql END SQL
END SQL
END
END

BEGIN
  EXEC ISode 1 SQL UPDATE [building_element_AREA] WHERE building_element = current_data linked_building_element
  linked_attribute_type OF building_element ATTRIBUTE = current_data
  WHERE building_element = current_data
  AND attribute_type = current_data
  END SQL
END SQL
END

ELSE
BEGIN
  text OF message box 2 := "Some essential data are missing. Check attribute name, type, and designing discipline."
  put up OF message box 2 := TRUE
  END

BEGIN
  attribute_type OF building_element ATTRIBUTE = current_data
  END

Ahmed Mohkhar, Ph.D Thesis .... 195
designing discipline OF building_element Attribute := designing discipline OF current_data
IF (CONF(linked_building_element OF building_element) = current_data) THEN
BEGIN
linked_building_element Attribute := linked_building_element
OF current_data
linked_attribute OF building_element Attribute := linked_attribute
OF current_data END
!
FORGET new attribute name
FORGET attribute_type OF building_element Attribute
OF current_data
FORGET designing discipline OF current_data
FORGET linked_building_element Attribute
OF current_data
!
FORGET linked_attribute OF building_element Attribute
OF current_data
FORGET attribute_type OF input data
FORGET discipline OF input data
location OF pushbutton 16 invisible2 := 180,195,220,220
location OF prompt 3 invisible2 := 25,195,220,220
location OF message box 18 invisible2 := 25,155,220,190
!
ELSE
BEGIN
text OF message box 2 := "Some material data are missing. Check attribute name, type, and designing discipline.
You may need to push New Rule button."
push up OF message box 2 := TRUE
END

attribute of attribute name SIMPLE WHEN CHANGED BEGIN
IF (CONF(new_attribute_name) = 0) THEN
BEGIN
**** check if there is an attribute with the same name
WHERE attribute_name OF building_element Attribute := new_attribute_name
WHEN FOUND
i := 1
WHEN NONE FOUND
i := 2
END
IF i = 2 THEN
BEGIN
MAKE building_element Attribute
WITH attribute_name OF building_element Attribute := new_attribute_name
location OF pushbutton 12 invisible2 := 25,265,225,285
location OF pushbutton 16 invisible2 := 0,0,0,0
END
!
END
!
**** end of method
END

ATTRIBUTE general rule SIMPLE WHEN CHANGED BEGIN
action Of IFSoc 1 IS connect := TRUE
EXEC IFSoc 1 SQL SELECT distinct (building_element) FROM [attribute_designer]
END SQL INTO building_element
!
visible OF main window := FALSE
output OF configuration window := General rules dialog
visible OF configuration window := TRUE
!
**** end of method
END

ATTRIBUTE apply rule SIMPLE WHEN CHANGED BEGIN
IF (CONF(building_element OF current_data) = 0 AND CONF(attribute_of_building_element OF building_element OF current_data) = 0 AND CONF(attribute_name OF building_element OF current_data)) THEN
BEGIN
EXEC IFSoc 1 SQL UPDATE [attribute_designer]
WHERE building_element = current_data
AND attribute_name = [attribute_designer]
END SQL INTO building_element
EXEC IFSoc 1 SQL SELECT distinct (building_element) FROM [attribute_designer]
WHERE building_element = current_data
AND attribute_name = [attribute_designer]
END SQL INTO linked_building_element
!
FORGET new_attribute_name
FORGET attribute_type OF input data

Ahmed Mokhtar, Ph.D Thesis .... 196
BEGIN
read line of file 1 := TRUE
IF SUBSTR (current line of file 1, 1, 16) = "path of ODBC.ini" THEN
    path_of_odbc := SUBSTR (current line of file 1, 16)
ELSE
    path_of_odbc := "SYSTEM\ODBC\16.DLL";
ENDIF
BEGIN
    read line of file 1 := TRUE
    write line of file 3 := CONCAT ("Driven", 
        path_of_odbc, "\ODBC.INI")
    write line of file 3 := CONCAT ("DBQ:", 
        project_name, "\MB")
    write line of file 3 := CONCAT ("DefaultINI", 
        current_path)
    write line of file 3 := "FILE:Microsoft Access"
    write line of file 3 := "IISPath\InAccess20.ini"
    write line of file 3 := "UID\Admin"
    write line of file 3 := TRUE
    action of file 1 is close := TRUE
    add the new file to the general database
    EXEC Eobdc 2 SQL insert into Current_projects
    (Project_name) values ("domain\project_name")
END SQL
BEGIN
    make a batch file to have a new database
    action of file 2 is close := TRUE
    action of file 2 is open new := TRUE
    write line of file 2 := CONCAT ("copy.", 
        current_path, 
        "empty.bat", 
        current_path, 
        "empty.bak", 
        current_path, 
        "empty.bak")
    write line of file 2 := CONCAT ("copy.", 
        current_path, 
        "empty.bat", 
        current_path, 
        "empty.bat")
    action of file 2 is close := TRUE
    make a copy of the empty database
    ACTIVATE "IPU, EXTERN, project.bat"
    connect the display to the window
    output of main window := 
    NEW_project_selection_display
BEGIN
end of method
END
ATTRIBUTE path_of_odbc STRING
ATTRIBUTE current_path STRING
ATTRIBUTE is_numeric NUMERIC
ATTRIBUTE in STRING
ATTRIBUTE get from general database SIMPLE
WHEN CHANGED
BEGIN
EXEC Eobdc 2 SQL select distinct 
(blding_element) from [General-AB-attribute-designer]
END SQL INTO
GENERAL_Blding_element
BEGIN
end of method
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
WHERE label of Menus_classified_B_comp := 
building_element OF Building_element
WHEN NONE FOUND
MAKE Menus_unclassified_B_comp
WITH label := building_element OF
Building_element
BEGIN
END
GENERAL_building_element_attribute() = 100 THEN BEGIN
  
  **** the following step is necessary as data can not be got directly from access to this attribute
  EXEC pbof SQL SELECT (linked_attribute_of_building_element() from [GENERAL-AB-attribut-designer])
    where building_element = :domain.GENERAL_building_component and 
    END SQL INTO domain (nt)
  END
  
  EXEC pbof 1 SQL INSERT INTO ([AB-attribute-designer])
    (building_element, attribute_of_building_element, attribute_type, designing_discipline, linked_building_element, 
    linked_attribute_of_building_element) values
    (:domain.GENERAL_building_component, 
    :GENERAL_building_element_attribute.attribute_of_building_element, 
    :GENERAL_building_element_attribute.attribute_type, 
    :GENERAL_building_element_attribute.designing_discipline, 
    :GENERAL_building_element_attribute.linked_building_element, 
    :GENERAL_building_element_attribute.linked_attribute_of_building_element)

END

SQL END ELSE BEGIN
  EXEC pbof 1 SQL INSERT INTO ([AB-attribute-designer])
    (building_element, attribute_of_building_element, attribute_type, 
    designing_discipline) values
    (:domain.GENERAL_building_component, 
    :GENERAL_building_element_attribute.attribute_of_building_element, 
    :GENERAL_building_element_attribute.attribute_type, 
    :GENERAL_building_element_attribute.designing_discipline)
END SQL END

**** add the rest of attributes to the created table
FOR (i = 1 TO number attributes OF navigate GENERAL_building_element_attributes)
BEGIN
instance number OF navigate GENERAL_building_element_attributes = ii
    **** ignore the attribute that has the id of the table
    IF SEARCHSTR iy, attribute_of_building_element, attribute_of_building_element() = 0 THEN BEGIN
        **** put the attribute in the database
        general statement of sql = CONCAT("alter table ", building_element, ": ADD [", attribute_of_building_element, " ]")
        IF attribute_type OF GENERAL_building_element_attribute = "x" THEN 
        general statement of sql = CONCAT( 
            "ALTER table ", building_element, ": ADD [", attribute_of_building_element, " ]")
        IF attribute_type OF GENERAL_building_element_attribute = "y" THEN 
        general statement of sql = CONCAT(
            "ALTER table ", building_element, ": ADD [", attribute_of_building_element, " ]")
        END SQL INTO END SQL END

**** end of method
END

**** add the attributes to the table [AB-attribute-designer] in the new project
FOR (i = 1 TO number attributes OF navigate GENERAL_building_element_attributes)
BEGIN
instance number OF navigate GENERAL_building_element_attributes = ii
    IF CONF(linked_building_element, attribute_of_building_element, attribute_of_building_element() = 0 THEN BEGIN
        **** the following step is necessary as data can not be got directly from access to this attribute
        EXEC pbof 2 SQL SELECT (linked_attribute_of_building_element() from [GENERAL-AB-attribut-designer])
          where building_element = :domain.GENERAL_building_component and attribute_of_building_element = 
          :GENERAL_building_element_attribute.attribute_of_building_element
          END SQL INTO domain (nt)
        END
        
        EXEC pbof 1 SQL INSERT INTO ([AB-attribute-designer])
          (building_element, attribute_of_building_element, attribute_type, 
          designing_discipline, linked_building_element, 
          linked_attribute_of_building_element) values
          (:domain.GENERAL_building_component, 
          :GENERAL_building_element_attribute.attribute_of_building_element, 
          :GENERAL_building_element_attribute.attribute_type, 
          :GENERAL_building_element_attribute.designing_discipline, 
          :GENERAL_building_element_attribute.linked_building_element, 
          :GENERAL_building_element_attribute.linked_attribute_of_building_element)

WITH border width = 3
WITH background color = 192,192,192
WITH fill color = 192,192,192
WITH highlight color = 255,255,255
WITH shadow color = 128,128,128
WITH location = 255,110,473,520

INSTANCE border 21 ISA border
WITH style 3D picture frame
WITH perspective 15 in
WITH border width = 3
WITH background color = 192,192,192
WITH fill color = 0,128,128
WITH highlight color = 255,255,255
WITH shadow color = 128,128,128
WITH location = 20,53,415,330

INSTANCE border 24 ISA border
WITH style ISA group
WITH perspective ISA in
WITH border width = 1
WITH background color = 192,192,192
WITH fill color = 192,192,192
WITH highlight color = 255,255,255
WITH shadow color = 128,128,128
WITH location = 5,40,413,340

INSTANCE border 25 ISA border
WITH style ISA group
WITH perspective ISA in
WITH border width = 1
WITH background color = 192,192,192
WITH fill color = 192,192,192
WITH highlight color = 255,255,255
WITH shadow color = 128,128,128
WITH location = 10,43,410,335

INSTANCE edit table column 1 ISA edit table column
WITH attachment := attribute_of_building_element
OF building_element_attribute
WITH header label := "attribute_of_building_element"
WITH width := 70
WITH justify left
WITH pen color [1] := UNDETERMINED
WITH fill color [1] := UNDETERMINED

INSTANCE edit table column 2 ISA edit table column
WITH attachment := attribute_type
OF building_element_attribute
WITH header label := "attribute_type"
WITH width := 70
WITH justify left
WITH pen color [1] := UNDETERMINED
WITH fill color [1] := UNDETERMINED

INSTANCE edit table column 3 ISA edit table column
WITH attachment := designing_discipline
OF building_element_attribute
WITH header label := "designing_discipline"
WITH width := 70
WITH justify left
WITH pen color [1] := UNDETERMINED
WITH fill color [1] := UNDETERMINED

INSTANCE edit table column 4 ISA edit table column
WITH attachment := linked_building_element
OF building_element_attribute
WITH header label := "linked_building_element"
WITH width := 70
WITH justify left
WITH pen color [1] := UNDETERMINED
WITH fill color [1] := UNDETERMINED

INSTANCE edit table column 5 ISA edit table column
WITH attachment := attribute_of_building_element
OF building_element_attribute
WITH header label := "attribute_of_building_element"
WITH width := 70
WITH justify left
WITH pen color [1] := UNDETERMINED
WITH fill color [1] := UNDETERMINED

INSTANCE 150txt 1 ISA 150txt:
WITH data source := "general"
WITH connection prompting := FALSE
WITH auto commit := TRUE
WITH append := FALSE
WITH default error handling := TRUE

INSTANCE 150txt 3 ISA 150txt:
WITH data source := "general"
WITH connection prompting := FALSE
WITH auto commit := TRUE
WITH append := FALSE
WITH default error handling := TRUE

INSTANCE message box 1 ISA message box
WITH title := "Warning Messages"
WITH text := "Are you sure that you want to DELETE the current component attribute?"
WITH modal style ISA sys modal
WITH icon ISA stop sign
WITH button ISA ok
WITH default button ISA none

INSTANCE message box 2 ISA message box
WITH title := "Input Error"
WITH modal style ISA app modal
WITH icon ISA stop sign
WITH button ISA ok
WITH default button ISA none

INSTANCE navigate 1 ISA navigate
WITH class attachment := building_element_attribute

INSTANCE rules navigate 2 ISA navigate
WITH class attachment := AB_general_rules

INSTANCE navigate Building element ISA navigate
WITH class attachment := Building element

INSTANCE navigate menu_group_items ISA navigate
WITH class attachment := Menu_group_items

INSTANCE navigate building_element_attribute ISA navigate
WITH class attachment := building_element_attribute

INSTANCE navigate GENERAL building_element ISA navigate
WITH class attachment := GENERAL_building_element

INSTANCE navigate building_element_attribute ISA navigate
WITH class attachment := building_element_attribute

INSTANCE the application ISA application
WITH unknown fail := TRUE
WITH threshold := 50
WITH title display := Starting display
WITH ignore breakpoints := FALSE
WITH reasoning on := FALSE
WITH numeric precision := 8
WITH demo strategy ISA fire first

INSTANCE column 1 ISA column
WITH attachment := attribute_of_building_element
OF building_element_attribute
WITH width := 100
WITH justify left
WITH header label := "Attribute"
WITH wordwrap := TRUE
WITH heading label := "Name"

INSTANCE column 2 ISA column
WITH attachment := attribute_type
OF building_element_attribute
WITH width := 60
WITH justify left
WITH wordwrap := TRUE
WITH heading label := "Type"

WITH default error handling => TRUE

INSTANCE file 3 ISA file
WITH shared => FALSE
WITH default error handling => TRUE

INSTANCE listbox 1 ISA listbox
WITH show current instance => TRUE
WITH source => building_element OF Building_element
WITH destination => building_element OF current_data
WITH location => 16,71,210,95

INSTANCE listbox 2 ISA listbox
WITH show current instance => TRUE
WITH source => attribute_of_building_element OF building_element_attribute
WITH destination => attribute_of_building_element OF current_data
WITH location => 16,171,235,240

INSTANCE listbox 3 ISA listbox
WITH show current instance => TRUE
WITH source => building_element OF Linked building_element
WITH destination => linked_building_element OF current_data
WITH location => 25,320,235,345

INSTANCE listbox 4 ISA listbox
WITH show current instance => TRUE
WITH source => attribute_of_building_element OF Linked building_element
WITH destination => linked_attribute OF building_element OF current_data
WITH location => 25,265,223,390

INSTANCE listbox 5 ISA listbox
WITH show current instance => TRUE
WITH source => building_element OF Building_element
WITH destination => building_element OF Current_rule_data
WITH location => 265,75,489,99

INSTANCE listbox 6 ISA listbox
WITH show current instance => TRUE
WITH source => attribute_of_building_element OF building_element_attribute
WITH destination => attribute_of_building_element OF Current_rule_data
WITH location => 265,120,489,144

INSTANCE listbox 8 ISA listbox
WITH show current instance => TRUE
WITH source => label OF Menus_group
WITH destination => current_menu B_comp OF Menus_manipulation_data
WITH location => 16,91,165,315

INSTANCE listbox 9 ISA listbox
WITH show current instance => TRUE
WITH source => current_menu_group OF Menus_manipulation_data
WITH location => 185,90,335,205

INSTANCE listbox 10 ISA listbox
WITH show current instance => TRUE
WITH source => label OF Menus_group_items
WITH destination => current_menu B_comp OF Menus_manipulation_data
WITH location => 350,90,495,225

INSTANCE listbox 11 ISA listbox
WITH show current instance => TRUE
WITH source => project_name OF Current_projects
WITH destination => project_name
WITH location => 11,140,160,165

INSTANCE listbox 12 ISA listbox
WITH show current instance => TRUE
WITH source => project_name OF Current_projects
WITH destination => another_project_name OF Project projects
WITH location => 10,95,160,115

INSTANCE listbox 13 ISA listbox
WITH show current instance => TRUE
WITH source => building_element OF Building_element
WITH destination => building_element OF NEW_to_make_a_new_project
WITH location => 35,95,460,180

INSTANCE listbox 14 ISA listbox
WITH show current instance => TRUE
WITH source => attribute_of_building_element OF building_element_attribute
WITH location => 115,235,460,330

INSTANCE listbox 15 ISA listbox
WITH show current instance => TRUE
WITH source => building_element OF Building_element
WITH location => 265,200,465,285

INSTANCE listbox 16 ISA listbox
WITH show current instance => TRUE
WITH source => building_system OF NEW_building_systems
WITH destination => building_system OF Building_element
WITH location => 30,150,220,270

INSTANCE listbox 17 ISA listbox
WITH show current instance => TRUE
WITH source => discipline OF NEW_design_disciplines
WITH destination => design_disciplines OF NEW_to_make_a_new_project
WITH location => 265,54,460,87

INSTANCE listbox 18 ISA listbox
WITH show current instance => TRUE
WITH source => building_element OF GENERAL_Buidling_element
WITH destination => GENERAL_Building_component
WITH location => 21,76,400,160

INSTANCE listbox 19 ISA listbox
WITH show current instance => TRUE
WITH source => attribute_of_building_element OF GENERAL_building_element_attribute
WITH destination => 110,305,400,230

INSTANCE menu c FOUNDATION ISA menu
WITH label => "Foundation"
WITH items [1] => UNDETERMINED
WITH menu c FLOOR ISA menu
WITH label => "Floor"
WITH items [1] => UNDETERMINED
WITH menu c WALL ISA menu
WITH label => "Wall"
WITH items [1] => UNDETERMINED
WITH menu c ROOF ISA menu
WITH label => "Roof"
WITH items [1] => UNDETERMINED
WITH menu c OPENING ISA menu
WITH label => "Opening"
WITH items [1] => UNDETERMINED
WITH menu c PROTECTION ISA menu
WITH label => "Protection"
WITH items [1] => UNDETERMINED
WITH menu c SPECIAL ISA menu
WITH label => "Special"
WITH items [1] => UNDETERMINED
WITH menu c MECHANICAL ISA menu
WITH label => "Mechanical"
WITH items [1] => UNDETERMINED
WITH menu c ELECTRICAL ISA menu
WITH label => "Electrical"
WITH items [1] => UNDETERMINED
WITH menu c PIER ISA menu
WITH label => "Pier"
WITH items [1] => UNDETERMINED

Ahmed Mohr, Ph.D Thesis ... 202
WITH attribute attachment := finish system of the new project
OF NEW_make_a_new_project
WITH location := 253,560,473,385

INSTANCE pushbutton 51 ISA pushbutton
WITH label := "Ignore highlighted component."
WITH attribute attachment := ignore highlighted
component OF NEW_make_a_new_project
WITH location := 265,285,485,110

INSTANCE pushbutton 52 ISA pushbutton
WITH label := ""
WITH attribute attachment := get from general
database
WITH location := 425,40,500,85

INSTANCE pushbutton 53 ISA pushbutton
WITH label := "Add the Component"
WITH attribute attachment := add component from
get from general database
WITH location := 20,285,215,305

INSTANCE pushbutton 54 ISA pushbutton
WITH label := "Return"
WITH attribute attachment := return from get from
general database
WITH location := 215,285,400,305

INSTANCE pushbutton 55 ISA pushbutton
WITH label := "Add the Component and its related
General Rules"
WITH attribute attachment := add component and
general rules
WITH location := 20,285,400,325

INSTANCE radio button group 1 ISA radio button group
WITH pen color := 0,0,0
WITH fill color := 192,192,192
WITH frame := TRUE
WITH group label := ""
WITH show current := TRUE
WITH attachment := discipline OF input data
WITH location := 235,145,355,295

INSTANCE radio button group 2 ISA radio button group
WITH pen color := 0,0,0
WITH fill color := 192,192,192
WITH frame := TRUE
WITH group label := "attribute type"
WITH show current := TRUE
WITH attachment := discipline OF input data
WITH location := 235,145,450,225

INSTANCE radio button group 3 ISA radio button group
WITH pen color := 0,0,0
WITH fill color := 192,192,192
WITH frame := TRUE
WITH group label := ""
WITH show current := TRUE
WITH attachment := discipline OF input data
WITH location := 260,220,395,385

INSTANCE table 1 ISA table
WITH attachment := building_element_attribute
WITH columns [ 1 ] := column 1
WITH columns [ 2 ] := column 2
WITH columns [ 3 ] := column 3
WITH columns [ 4 ] := column 4
WITH columns [ 5 ] := column 5
WITH heading := TRUE
WITH heading height := 50
WITH fill color := 0,255,255
WITH column lines := TRUE
WITH raw lines := TRUE
WITH frame := TRUE
WITH show current instance := TRUE
WITH location := 10,45,590,240

INSTANCE textbox 1 ISA textbox
WITH pen color := 255,255,255
WITH fill color := 0,128
WITH justify IS center
WITH font := "Times New Roman"
WITH font style IS bold, italic, underline CF FALSE, stroke CF FALSE
WITH font size := 12
WITH text := "CONFIGURATION MODULE"
WITH location := 10,30,155,80

INSTANCE textbox 2 ISA textbox
WITH pen color := 255,255,255
WITH fill color := 128,0,0
WITH justify IS center
WITH font := "Times New Roman"
WITH font style IS bold, italic CF FALSE, underline CF FALSE, strikeout CF FALSE
WITH font size := 12
WITH frame := TRUE
WITH text := "Configuration of Building Components
and Attributes"
WITH location := 5,5,420,35

INSTANCE textbox 3 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "List of Building Components"
WITH location := 15,50,205,70

INSTANCE textbox 4 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Current Building Components"
WITH location := 215,50,410,70

INSTANCE textbox 5 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Building Component Attributes"
WITH location := 15,150,220,170

INSTANCE textbox 6 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Current Attributes"
WITH location := 15,240,130,260

INSTANCE textbox 7 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Select Linked Component"
WITH location := 213,390,223,230

INSTANCE textbox 8 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Graph name"
WITH location := 15,105,100,125

INSTANCE textbox 9 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Select Linked Attributes"
WITH location := 25,345,185,365

INSTANCE textbox 10 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Current Linked Component"
WITH location := 260,300,460,320

INSTANCE textbox 11 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Current Linked Attributes"
WITH location := 260,345,420,365

INSTANCE textbox 12 ISA textbox
WITH pen color := 0,255
WITH fill color := 192,192,192
WITH justify IS left
WITH font := "System"
WITH text := "Then the discipline"
WITH location := 355,225,445,250

Ahmed Mokhtar, Ph.D Thesis ....204
WITH clipped := TRUE
WITH attachment := current_menu_B_comp OF Menu_manipulation_data
WITH location := 150,295,495,320

INSTANCE valuebox 15 ISA valuebox
WITH pen color := 0,0,255
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH frame := TRUE
WITH clipped := TRUE
WITH attachment := current_menu_group OF Menu_manipulation_data
WITH location := 183,230,335,225

INSTANCE valuebox 16 ISA valuebox
WITH pen color := 0,0,255
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH frame := TRUE
WITH clipped := TRUE
WITH attachment := building_element OF NEW_to_make_a_new_project
WITH location := 115,215,660,245

INSTANCE valuebox 17 ISA valuebox
WITH pen color := 0,0,255
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH frame := TRUE
WITH clipped := TRUE
WITH attachment := building_system OF NEW_to_make_a_new_project
WITH location := 285,135,665,160

INSTANCE valuebox 18 ISA valuebox
WITH pen color := 0,0,255
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH frame := TRUE
WITH clipped := TRUE
WITH attachment := description OF NEW_building_systems
WITH location := 15,115,230,370

INSTANCE valuebox 19 ISA valuebox
WITH pen color := 0,0,255
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH frame := TRUE
WITH clipped := TRUE
WITH attachment := GENERAL_Building_component
WITH location := 110,170,400,200

INSTANCE main window ISA window
WITH location := 207,160,181,413
WITH menus[1] := UNDETERMINED
WITH style IS moveable, sizeable, closable
WITH title := ""
WITH visible := TRUE
INSTANCE configuration window ISA window
WITH location := 61,11,378,646
WITH menus[1] := UNDETERMINED
WITH style IS moveable, sizeable, closable
FALSE
WITH title := ""
WITH visible := FALSE
INSTANCE attribute window ISA window
WITH location := 22,62,627,371
WITH menus[1] := UNDETERMINED
WITH style IS moveable, sizeable, closable
WITH title := ""
WITH visible := FALSE
INSTANCE get from general database window ISA window
WITH location := 192,11,2,617,490
WITH menus[1] := UNDETERMINED
WITH style IS moveable, sizeable, closable
WITH title := ""
WITH visible := FALSE

! DEMON GROUP: affected_discipline OF AB_general_rules

DEMON 11
IF Conf(affected_discipline OF AB_general_rules) >= 0 THEN FORGET discipline OF input data

! DEMON GROUP: button selected OF message box

DEMON 5
IF button selected OF message box 1 IS YES THEN delete a component := TRUE

DEMON 10
IF button selected OF message box 3 IS YES THEN delete an attribute := TRUE

! DEMON GROUP: selected OF pushbutton

DEMON 1
IF selected OF pushbutton 6 THEN text OF textbox 16 := building_element OF current_data
AND output OF attribute window := attributes table display
AND visible OF attribute window := TRUE

DEMON 2
IF selected OF pushbutton 8 THEN visible OF attribute window := FALSE

DEMON 4
IF selected OF pushbutton 3 THEN put up OF message box 1 := TRUE

DEMON 6
IF selected OF pushbutton 4 THEN location OF border 6 := invisible := 15,50,410,95
AND location OF textbox 17 := invisible := 25,55,160,90
AND location OF promptbox 2 := invisible := 170,62,340,85
AND location OF pushbutton 10 := invisible := 355,50,410,70
AND location Of pushbutton 11 := invisible := 355,75,410,95
AND location Of promptbox 12 := invisible := 0,0,0,0
AND location Of promptbox 3 := invisible := 0,0,0,0
AND location Of border 7 := invisible := 0,0,0,0
AND location Of textbox 18 := invisible := 0,0,0,0
AND location Of textbox 19 := invisible := 0,0,0,0
AND location Of pushbutton 13 := invisible := 0,0,0,0
AND FORGET current_data
AND FORGET building_element_attribute

DEMON 7
IF selected OF pushbutton 11 := invisible THEN location Of border 6 := invisible := 0,0,0,0
AND location Of textbox 17 := invisible := 0,0,0,0
AND location Of promptbox 2 := invisible := 0,0,0,0
AND location Of pushbutton 10 := invisible := 0,0,0,0
AND location Of pushbutton 11 := invisible := 0,0,0,0

DEMON 8
IF selected OF pushbutton 15 THEN location Of pushbutton 16 := invisible := 180,195,220,220
AND location Of promptbox 3 := invisible := 25,195,220,220
AND location Of border 7 := invisible := 15,150,235,250
AND location Of textbox 18 := invisible := 25,155,220,190
AND location Of textbox 19 := invisible := 25,235,255,260
AND FORGET attribute_of_building_element OF current_data
AND FORGET attribute_type OF current_data
AND FORGET designing_discipline OF current_data
AND FORGET linked_building_element OF current_data
AND FORGET linked_attribute_of_building_element OF current_data

DEMON 9
IF selected OF pushbutton 14 THEN put up OF message box 3 := TRUE

! DEMON GROUP: double clicked OF table

DEMON 3
IF double clicked OF table THEN visible OF attribute window := FALSE
AND attribute_of_building_element OF current_data := attribute_of_building_element OF building_element_attribute
END
C.2. DATA MANIPULATION MODULE

$\text{WITH attribute of building element STRING}$
$\text{WHEN CHANGED BEGIN}$
$\text{IF ( CONF(attribute of building element OF current_data) ?? 0 ) THEN}$
$\text{BEGIN}$
$\text{attribute_type OF current_data := attribute_type OF building element attribute}$
$\text{designing Discipline OF current_data := designing Discipline OF building element attribute}$
$\text{linked_building_element OF current_data := linked_building_element OF building element attribute}$
$\text{linked_attribute_of_building_element attribute := linked_attribute_of_building_element attribute}$
$\text{FORGET attribute_type OF input data}$
$\text{FORGET designing Discipline OF input data}$
$\text{END}$
$\text{END}$

$\text{WITH cause of the effect STRING}$
$\text{WITH percent_change NUMERIC}$
$\text{WITH rule_ignored STRING}$

CLASS edit table INHERITS add_on, display item
$\text{WITH attribute of building element STRING}$
$\text{WHEN CHANGED BEGIN}$
$\text{EXEC [code] 1 SQL SELECT * from [attribute designer] WHERE building_element = current_data.building_element}$
$\text{FORGET attribute_type OF input data}$
$\text{FORGET designing Discipline OF input data}$
$\text{END}$

Ahmed Mokhtar, Ph.D Thesis ....208
Menus_manipulation_data
|
| *** end of IF END IF ELSE BEGIN
| BEGIN text_of message box 2 := "A selected Menu Group is missing" put_up_of message box 2 := TRUE END
|
| *** end of method END
| WITH delete menu group SIMPLE WHEN_CHANGED BEGIN
| IF CON(NEW_name_of_menu_group OF Menus_manipulation_data) > 0 THEN BEGIN
| EXEC Iso/dbc 1 SQL SELECT distant (NEW_name_of_menu_group) FROM [AB-menu_item]
| WHERE main_menu_label = Menu_manipulation_data.current_menu_group END SQL INTO Menu_group
|
| EXEC Iso/dbc 1 SQL SELECT distant (NEW_name_of_menu_group) FROM [AB-menu_item]
| WHERE main_menu_label = Menu_manipulation_data.current_menu_group END SQL INTO Menu_group
|
| IF true THEN BEGIN
| EXEC Iso/dbc 1 SQL DELETE from [AB-menu_item] WHERE main_menu_label = Menu_manipulation_data.current_menu_group END SQL
|
| FOR i := 1 TO number instances OF navigate Building_element BEGIN instance number OF navigate Building_element := i
| FIND Menu_unclassified B_comp WHERE label OF Menu_unclassified B_comp = building_element OF Building_element WHEN NONE FOUND MAKE Menu_unclassified B_comp = building_element OF Building_element
| FIND END END
|
| *** end of IF END ELSE BEGIN
| text_of message box 2 := "A selected Menu Group is missing" put_up_of message box 2 := TRUE END
|
| *** end of method END
| WITH make menu group SIMPLE WHEN_CHANGED BEGIN
| IF CON(NEW_name_of_menu_group OF Menus_manipulation_data) > 0 AND CON(
| current_menu_group OF Menus_manipulation_data) > 0 THEN BEGIN
| FOR i := 1 TO number instances OF navigate menu_group_item BEGIN
| instance number OF navigate menu_group_item := i
| EXEC Iso/dbc 1 SQL update [AB-menu_item] set main_menu_label = Menu_manipulation_data.new_name_of_menu_group
| WHERE label = Menu_group_item.label END
| WITH building_element STRING
|
| CLASS NEW_to_make a new project WITH from another project SIMPLE WITH from systems SIMPLE WHEN_CHANGED BEGIN EXEC Iso/dbc 2 SQL SELECT distant (discipline) from [building_systems] END SQL INTO New_design_disciplines
|
| EXEC Iso/dbc 1 SQL SELECT distant (building_element) FROM [general-AB-attribute-designer]
| END SQL INTO Building_element
|
| EXEC Iso/dbc 1 SQL SELECT distant (building_element) FROM [general-AB-attribute-designer] END SQL INTO Building_element
|
| CLASS New to make a new project WITH from another project SIMPLE WITH from systems SIMPLE WHEN_CHANGED BEGIN EXEC Iso/dbc 1 SQL SELECT distant (building_element) FROM [general-AB-attribute-designer] END SQL INTO Building_element
output OF main window := 
CURRENT_project_selection_display
END
ATTRIBUTE:
check_the_existence_of_new_project_name SIMPLE
WHEN CHANGED
BEGIN
FIND Current_projects
WHERE project_name OF Current_projects = project_name
WHEN FOUND
text OF message box 2 := "This name already exists for a current project.
Type another name please.
put up OFF message box 2 := TRUE
setup a new project := TRUE
FIND END
END
**** end of method
END
ATTRIBUTE setup a new project SIMPLE
WHEN CHANGED
BEGIN
***** this method does the following:
1) Read the paths of the ODBC.ini file and the path of the directory that is used for this program and contains an
empty database
2) Modify the ODBC.ini file so it links to the new database
3) Add the file name to the General database so as to be considered a new project
4) Prepare a batch file to execute the next step
5) Make a new database by copy the empty database to a file with the same name
BEGIN
--- read the path of the files from the file project.ini
action OF file 1 IS open old := TRUE
WHERE (NOT EOF OF file 1)
BEGIN
read line OF file 1 := TRUE
IF SUBSTR(line, 1, 1) = "path of ODBC.ini" THEN
path_of_odbc := SUBSTR(line, 1, 1)
ELSE IF SUBSTR(line, 1, 29) = "path of the developed program" THEN
path_of_current := SUBSTR(line, 1, 1)
END
action OF file 1 IS close := TRUE
--- modify the ODBC.ini
filename OF file 3 := CONCAT(path_of_odbc,
"ODBC.INI")
action OF file 3 IS open old := TRUE
WHERE (NOT EOF OF file 3)
BEGIN
write line OF file 3 := CONCAT("[", project_name,"]")
write line OF file 3 := CONCAT("Driver=",
path_of_odbc,"/SYSTEM/ODBC/16.DLL")
write line OF file 3 := CONCAT("DBQ=",
project_name,".MDB")
action OF file 3 IS close := TRUE
END
--- add the new file to the general database
EXEC SQL SELECT * FROM [General-db
attribute = domain.project_name]
WHERE file_name = file_1
END
--- make a batch file to have a new database
action OF file 2 IS delete := TRUE
action OF file 2 IS open new := TRUE
write line OF file 2 := CONCAT("copy",
current_path,"empty.odb", current_path,
project_name,".odb")
write line OF file 2 := CONCAT("copy",
current_path,"empty.odb", current_path,
project_name,".odb")
action OF file 2 IS close := TRUE
EXEC SQL INTO General_Building_element WHERE (NOT EOF OF file 2)
BEGIN
instance number OF navigate Building element :=
FIND GENERAL_Building_element
WHERE building_element OF
General_Building_element = building_element OF
General_Building_element
WHEN FOUND
FORGET CURRENT
END GENERAL_Building_element
END
--- output OF get from general database window := TRUE
output OF get from general database window := TRUE
--- add from general database display
END
**** end of method
END
ATTRIBUTE GENERAL_Building_element STRING
WHEN CHANGED
BEGIN
EXEC SQL SELECT * FROM [General-
attribute = domain.project_name]
WHERE building_element =
General.Building_element
END GENERAL_building_element
ATTRIBUTE add component from _get from general
database SIMPLE
WHEN CHANGED
BEGIN
i := 0
FIND Building_element
WHERE LOWCASE(building_element OF
Building_element) = LOWCASE(GENERAL_Building_element)
END
WHEN FOUND
i := 1
WHEN NONE FOUND
i := 2
FIND END
--- add the new message box := "This Component
Already exists"
put up OFF message box 2 := TRUE
END
--- create the database table with its id attribute
--- general statement of sql := CONCAT("CREATE
TABLE [", GENERAL_Building_element, ","]
--- ("..."
Ahmed Mokhtar, Ph.D Thesis ....220
INSTANCE pushbutton 8 ISA pushbutton
WITH label := "Return"
WITH location := 175,250,395,275

INSTANCE pushbutton 10 invisible ISA pushbutton
WITH label := "Done"
WITH attribute attachment := done of new component
WITH location := 0,0,0

INSTANCE pushbutton 11 invisible ISA pushbutton
WITH label := "Cancel"
WITH location := 0,0,0

INSTANCE pushbutton 12 invisible2 ISA pushbutton
WITH label := "Done"
WITH attribute attachment := done of new attribute
WITH location := 0,0,0

INSTANCE pushbutton 13 invisible2 ISA pushbutton
WITH label := "Next Attribute"
WITH attribute attachment := next new attribute
WITH location := 0,0,0

INSTANCE pushbutton 14 ISA pushbutton
WITH label := "Delete"
WITH location := 180,250,235,270

INSTANCE pushbutton 15 ISA pushbutton
WITH label := "New"
WITH location := 180,370,235,290

INSTANCE pushbutton 16 invisible2 ISA pushbutton
WITH label := "OK"
WITH attribute attachment := ok of attribute name
WITH location := 0,0,0

INSTANCE pushbutton 17 ISA pushbutton
WITH label := "General Rules"
WITH location := 10,165,160,190

INSTANCE pushbutton 18 ISA pushbutton
WITH label := "First"
WITH attribute attachment := action OF rules new IS first
WITH location := 400,240,490,265

INSTANCE pushbutton 19 ISA pushbutton
WITH label := "Next"
WITH attribute attachment := action OF navigate IS next
WITH location := 400,265,490,290

INSTANCE pushbutton 20 ISA pushbutton
WITH label := "Previous"
WITH attribute attachment := action OF navigate IS previous
WITH location := 400,290,490,315

INSTANCE pushbutton 21 ISA pushbutton
WITH label := "Last"
WITH attribute attachment := action OF navigate IS last
WITH location := 400,315,490,340

INSTANCE pushbutton 22 ISA pushbutton
WITH label := "END"
WITH attribute attachment := back_to_current_project_display
WITH location := 280,400,490,425

INSTANCE pushbutton 23 ISA pushbutton
WITH label := "Update Rule"
WITH attribute attachment := update rule
WITH location := 30,400,230,425

INSTANCE pushbutton 24 ISA pushbutton
WITH label := "New Rule"
WITH attribute attachment := new rule
WITH location := 400,355,490,380

INSTANCE pushbutton 25 ISA pushbutton
WITH location := 10,102,110,35

INSTANCE pushbutton 26 invisible2 ISA pushbutton
WITH label := "Done"
WITH attribute attachment := done of ID attribute
WITH location := 0,0,0

INSTANCE pushbutton 26 ISA pushbutton
WITH label := "Menus"
WITH attribute attachment := Menus
WITH location := 10,135,160,160

INSTANCE pushbutton 27 ISA pushbutton
WITH label := "END"
WITH attribute attachment := back_to_current_project_display
WITH location := 245,300,500,415

INSTANCE pushbutton 28 ISA pushbutton
WITH label := "Add to the Menu Group"
WITH attribute attachment := add build comp OF Menus_manipulation_data
WITH location := 10,385,170,410

INSTANCE pushbutton 29 ISA pushbutton
WITH label := "Del. Component"
WITH attribute attachment := delete build comp OF Menus_manipulation_data
WITH location := 350,325,495,250

INSTANCE pushbutton 30 ISA pushbutton
WITH label := "Add A Menu Group"
WITH attribute attachment := add menu group OF Menus_manipulation_data
WITH location := 185,360,335,385

INSTANCE pushbutton 31 ISA pushbutton
WITH label := "Delete Menu Group"
WITH attribute attachment := delete menu group OF Menus_manipulation_data
WITH location := 185,260,335,285

INSTANCE pushbutton 32 ISA pushbutton
WITH label := "Rename Menu Group"
WITH attribute attachment := rename menu group OF Menus_manipulation_data
WITH location := 185,385,335,410

INSTANCE pushbutton 33 ISA pushbutton
WITH label := "EXIT"
WITH attribute attachment := exit OF application
WITH location := 10,195,160,220

INSTANCE pushbutton 34 ISA pushbutton
WITH label := "New Project"
WITH attribute attachment := New_project
WITH display := NEW_project_start_display
WITH location := 10,105,160,135

INSTANCE pushbutton 35 ISA pushbutton
WITH label := "Existing Project"
WITH attribute attachment := Current_project
WITH display := CURRENT_project_start_display
WITH location := 10,145,160,175

INSTANCE pushbutton 36 ISA pushbutton
WITH label := "EDIT"
WITH attribute attachment := exit OF application
WITH location := 10,195,160,220

INSTANCE pushbutton 37 ISA pushbutton
WITH label := "OK"
WITH location := 10,170,160,195

INSTANCE pushbutton 38 ISA pushbutton
WITH label := "Return"
WITH location := 10,170,160,195

INSTANCE pushbutton 39 ISA pushbutton
WITH label := "from Another Project"
WITH display := NEW from another project display
WITH location := 10,105,160,140

INSTANCE pushbutton 40 ISA pushbutton
WITH label := "from Systems"
WITH attribute attachment := from systems OF NEW_to_make_a_new_project
WITH location := 10,145,160,180

INSTANCE pushbutton 41 ISA pushbutton
WITH label := "from Components"
WITH attribute attachment := from components OF NEW_to_make_a_new_project
WITH location := 10,185,160,220

INSTANCE pushbutton 42 ISA pushbutton
WITH label := "Modify components"
WITH location := 10,120,160,145

INSTANCE pushbutton 43 ISA pushbutton
WITH label := "Modify Gen. Rules"
WITH location := 10,145,160,170

INSTANCE pushbutton 44 ISA pushbutton
WITH label := "Conf. Memas"
WITH location := 10,170,160,195

INSTANCE pushbutton 45 ISA pushbutton
WITH label := "Back"
WITH display => NEW_project_selection_display
WITH location := 10,195,160,220

INSTANCE pushbutton 46 ISA pushbutton
WITH label := "OK"
WITH attribute attachment := from systems OF NEW_to_make_a_new_project
WITH location := 20,195,160,220

INSTANCE pushbutton 47 ISA pushbutton
WITH label := "Add selected component to the new project"
WITH attribute attachment := add component to the new project OF NEW_to_make_a_new_project
WITH location := 20,355,315,380

INSTANCE pushbutton 48 ISA pushbutton
WITH label := "Finish"
WITH attribute attachment := finish components of the new project OF NEW_to_make_a_new_project
WITH location := 345,355,410

INSTANCE pushbutton 49 ISA pushbutton
WITH label := "Add to the new project"
WITH attribute attachment := add to the new project OF NEW_to_make_a_new_project
WITH location := 253,330,475,325

INSTANCE pushbutton 50 ISA pushbutton
WITH label := "Finish"
WITH attribute attachment := finish systems of the new project OF NEW_to_make_a_new_project
WITH location := 253,360,475,285

INSTANCE pushbutton 51 ISA pushbutton
WITH label := "Ignore highlighted compo.
WITH attribute attachment := ignore highlighted component OF NEW_to_make_a_new_project
WITH location := 265,285,465,310

INSTANCE pushbutton 52 ISA pushbutton
WITH label := """"WITH attribute attachment := get from general database
WITH location := 425,400,500,85

INSTANCE pushbutton 53 ISA pushbutton
WITH label := "Add the Component"
WITH attribute attachment := add component from get from general database
WITH location := 20,285,215,305

INSTANCE pushbutton 54 ISA pushbutton
WITH label := "Return"
WITH attribute attachment := return from get from general database
WITH location := 215,285,490,305

INSTANCE pushbutton 55 ISA pushbutton
WITH label := "Add the Component and its related General Rules"
WITH attribute attachment := add component and general rules
WITH location := 20,305,400,325

INSTANCE radiobutton group 1 ISA radiobutton group
WITH pen color := 0,0,0
WITH fill color := 192,192,192
WITH frame := TRUE
WITH group label := """"WITH show current := TRUE
WITH attribute attachment := show current OF input data
WITH location := 253,145,355,295

INSTANCE radiobutton group 2 ISA radiobutton group
WITH pen color := 0,0,0
C.3. MESSAGE MODULE

VERSIONS

LOCATION ARE PIXELS

CLASS AB_general_rules

WITH AB_general_rules_ID NUMERIC
WITH building_element STRING
WITH attribute_of_building_element STRING
WITH affected_discipline STRING
WITH percent_change NUMERIC
WITH result_in_major_modification STRING
WITH cause_of_the_effect STRING

CLASS AB_specific_rules

WITH AB_specific_rules_ID NUMERIC
WITH building_element STRING
WITH attribute_of_building_element STRING
WITH related_instance STRING
WITH related_attribute STRING
WITH relationship STRING
WITH writing_date STRING
WITH relation COMPOUND

CLASS message_box

WITH id STRING
WITH title STRING
WITH text STRING
WITH variable_text STRING
WITH model_style COMPOUND
app modal, 
exit modal, 
not modal
WITH default_button COMPOUND
one, 
two, 
three
WITH button selected COMPOUND
short, 
cancelled, 
ignore, 
ok, 
retry
WITH put up SIMPLE

CLASS navigate_inherits

WITH class attachment CLASS REFERENCE
WITH action COMPOUND
first, 
previous, 
next, 
last

WHEN CHANGED
BEGIN
IF instance number OF navigate 1 <> 0 THEN
BEGIN

***** define the message color
IF been_read OF discipline_message = "Yes" AND been_actor_saved OF discipline_message = "Yes" THEN
pen color OF valuebox 1 := 0,0,255
END
END
END
END
END
END
END
END
END

Ahmed Mokhtar, Ph.D Thesis
AND old_value <-
  domain includes initial design value
END SQL INTO discipline_message
AND FORGET attachment OF valuebox 1

DEMON 6
IF message status OF required_message IS Unread Yet
AND chosen discipline OF required_message <>
"UNDETERMINED"
THEN EXEC 150db: SQL select * from
message_elements
where discipline_to =
required_message.chosen discipline
AND been_read = 'No$
AND rule ignore = domain.No$
END SQL INTO discipline_message
AND FORGET attachment OF valuebox 1

! DEMON GROUP: selected OF pushbutton

DEMON 17
IF selected OF message has been read button
THEN EXEC 150db: SQL update message_elements
set been_read = 'Yes$
where id = :discipline_message.id
END SQL

DEMON 18
IF selected OF message has been acted upon button
THEN EXEC 150db: SQL update message_elements
set been_act upon = 'Yes$
been_read = 'Yes$

where id = :discipline_message.id
END SQL

DEMON 8
IF selected OF pushbutton 9
THEN action OF 150db: IS disconnect

DEMON 9
IF selected OF pushbutton 11
THEN visible OF helping window = FALSE
AND FORGET output OF helping window

DEMON 11
IF selected OF pushbutton 14
THEN visible OF helping window = FALSE
AND FORGET output OF helping window

DEMON 12
IF selected OF pushbutton 17
THEN visible OF specific rule window = FALSE
AND FORGET output OF specific rule window

END

Ahmed Mokhtar, Ph.D Thesis ......231
C.4. TRACKING MODULE

class A

attribute_of_monitored_object
WITH name STRING

class A

change_monitoring_results
WITH change_id STRING
WITH date_time STRING
WITH discipline STRING
WITH object STRING
WITH object_id STRING
WITH attribute STRING
WITH old_value STRING
WITH new_value STRING
WITH proceeeding_change_id STRING

class A

monitored_object_input
WITH object STRING
WITH object_id STRING
WITH attribute STRING
WITH start_time STRING
WITH end_time STRING
WITH select_discipline COMPOUND

architecture
structure
envelope
HVAC
illumination
interior
when changed BEGIN

attribute_of_monitored_object_input := ""!

forge source of listbox 1
forge source of listbox 2
change_id_of_monitored_object_input := ""

if select discipline of A_monitored_object_input IS illumination THEN
BEGIN

discipline_of_monitored_object_input := ""

forge source of listbox 1
forge source of listbox 2
change_id_of_monitored_object_input := ""

end

if select discipline of A_monitored_object_input IS interior THEN
BEGIN

discipline_of_monitored_object_input := ""

forge source of listbox 1
forge source of listbox 2
change_id_of_monitored_object_input := ""

end

if select discipline of A_monitored_object_input IS architecture THEN
BEGIN

discipline_of_monitored_object_input := "arch"

forge source of listbox 1
forge source of listbox 2
change_id_of_monitored_object_input := ""

end

if select discipline of A_monitored_object_input IS structure THEN
BEGIN

discipline_of_monitored_object_input := "str"

forge source of listbox 1
forge source of listbox 2
change_id_of_monitored_object_input := ""

end

if select discipline of A_monitored_object_input IS envelope THEN
BEGIN

discipline_of_monitored_object_input := "env"

forge source of listbox 1
forge source of listbox 2
change_id_of_monitored_object_input := ""

end

if select discipline of A_monitored_object_input IS HVAC THEN
BEGIN

discipline_of_monitored_object_input := "hvac"

forge source of listbox 1
forge source of listbox 2
change_id_of_monitored_object_input := ""

end

for each discipline of A_monitored_object_input := ""

begin

forge source of listbox 1
forge source of listbox 2
change_id_of_monitored_object_input := ""

end

end class A

INT 255,255,255
WITH dropdown color COLOR
INT 128,128,128

class classbox

inherits on, display item
WITH attachment ATTRIBUTE REFERENCE
WITH selected SIMPLE
WITH enabled SIMPLE
WITH true FALSE
WITH update with current instance SIMPLE
WITH true TRUE

class current_project

WITH project_name STRING

class discipline

inherits on, string
WITH changed_attribute STRING
WITH changed_class STRING
WITH new_value STRING
WITH old_value STRING
WITH effect_attribute STRING
WITH effect_class STRING
WITH effect_instance_id STRING
WITH relationship STRING
WITH writing_date TIME
WITH reading_date DATE
WITH time_read STRING
WITH time_acquired STRING
WITH id NERIC
WITH discipline_to STRING
WITH discipline_from STRING
WITH set because change_id NERIC
WITH general_specific STRING
WITH role_writing_data STRING
WITH violated_role_id NERIC
WITH cause_of_the_effect STRING

class Isocode

inherits on, add
WITH data_source STRING
WITH world STRING
WITH password STRING
WITH connection STRING
WITH connection prompting SIMPLE
WITH false FALSE
WITH auto commit SIMPLE
WITH action COMPOUND
WITH connect disconnect
WITH records NERIC
WITH transaction COMPOUND
WITH rollback
WITH append SIMPLE
WITH true TRUE
WITH status STRING
WITH show error SIMPLE
WITH default error handling SIMPLE
WITH true TRUE
WITH error message STRING
WITH native error NERIC
WITH trace file STRING

class message_box

inherits on, add
WITH title STRING
WITH text STRING
WITH variable text STRING
WITH model style COMPOUND
WITH rys modal
WITH rys modal
WITH not modal
WITH rys modal
WITH icon COMPOUND
WITH image STRING
WITH information string
WITH question
WITH stop sign
WITH icon icon
WITH icon icon
WITH button COMPOUND
WITH retry ignore ok

Ahmed Mokhtar, Ph.D Thesis ....232
WITH designers_of_related_class attributes STRING
ARRAY SIZE 1
CLASS required_messages
WITH discipline_name COMPUND
Architectural,
HVAC,
Illumination,
Envelope,
Structure,
Interior
WITH message_status COMPUND
All Messages,
Unfound Yet,
Unknown Upon
WITH chosen discipline STRING
WITH message_builder STRING
CLASS temp_current_menu_items
WITH label STRING
ATTRIBUTE as NUMERIC
ATTRIBUTE number of table columns NUMERIC
ATTRIBUTE show results SIMPLE
WHEN CHANGED
BEGIN
! ***** to test that there are input data
IF (object OF A_monitored_object_input <> "")
OR (discipline OF A_monitored_object_input <> "")
OR (start_time OF A_monitored_object_input <> "")
OR (end_time OF A_monitored_object_input <> ")"
OR (change_ID OF A_monitored_object_input <> "")
 THEN
BEGIN
 put up message box 2 := TRUE
 ELSE
 BEGIN
 END
 ELSE
 END
! ***** Start of SELECT statement
 selection_statement OF
 A_monitored_object_input := SELECT * FROM (A-
 change_monitoring-data) WHERE * 
! ***** To select based on the discipline
 IF discipline OF A_monitored_object_input <> ""
 THEN
 BEGIN
 selection_statement OF
 A_monitored_object_input := CONCAT(
 selection_statement OF A_monitored_object_input,
 discipline OF A_monitored_object_input)
 THEN
 END
 END
! ***** To test based on the ID
 IF Change_ID OF A_monitored_object_input <> ""
 THEN
 BEGIN
 numeric Change_ID := TO NUMERIC(
 Change_ID OF A_monitored_object_input)
 selection_statement OF
 A_monitored_object_input := CONCAT(
 selection_statement OF A_monitored_object_input,
 "A_monitored_object_input纪律")
 THEN
 END
 END
! ***** To test based on object data
 IF object OF A_monitored_object_input <> ""
 THEN
 BEGIN
 selection_statement OF
 A_monitored_object_input := CONCAT(
 selection_statement OF A_monitored_object_input,
 object OF A_monitored_object_input")
 THEN
 IF object ID OF A_monitored_object_input <> ""
 THEN
 END
 ELSE
 END
! ***** reset all the other query data
 Change_ID OF A_monitored_object_input := ""
 discipline OF A_monitored_object_input := ""

IF (number instances OF navigate menu > 0) THEN BEGIN
RESIZE menu of responding window := number instances OF navigate menu
BEGIN instance number OF navigate menu := n
****** load the menu items in every menu EXEC Sync 1 SQL SELECT label from [AB-menu_item] WHERE i (main_menu label) = menu label
END SQL INTO temp_current_menu
RESIZE items OF menu := number instances OF navigate menu temp_current_menu
BEGIN instance number OF navigate menu := n
****** load the menu items WHERE label OF menu item = label OF temp_current_menu
END temp_current_menu
begin FIND menu item WHERE label OF menu item = label OF temp_current_menu
END
begin menu [n] OF responding window := menu item
END
begin visible OF Select a project window := FALSE visible OF responding window := TRUE
new case or responding := "responding"

### end of method
begin ATTRIBUTE another change in new case SIMPLE ATTRIBUTES close change data display SIMPLE WHEN CHANGED
begin IF Conf(Required max hours OF A_change_data) > 1 AND Conf(Difficultром) = Build_Cost_OF_A_change_data) > 1 AND Conf(Recommendation OF A_change_data) > 1 AND Conf(Duration OF A_change_data) > 1 AND Conf(value OF A_change_data) > 1 THEN BEGIN
IF 0 = Recommendation OF A_change_data AND Recommendation OF A_change_data = 0 THEN BEGIN
IF NOT (Recommendation OF A_change_data) THEN BEGIN
****** define the CHANGE OF A_change_data (combine the "discipline letter", "*", with "ID OF A_change_data") EXEC Sync 1 SQL SELECT MAX(id) FROM [A_change_data] END SQL INTO domain (sn) := 1
EXEC Sync 1 SQL SELECT discipline from [AIR] WHERE discipline OF A_change_data = "A_change_data_object AND A_change_data_attribute = "A_change_data_object"
END SQL INTO domain (tn)
CHANGE OF A_change_data := CONCAT(5, TO STRING(10))
****** put the time in a format that can be stored in the database (NAT number format)
current_time := NOW
year := YEAR(current_time)
month := MONTH(current_time)
day := DAY(current_time)
hour := HOUR(current_time)
minute := MINUTE(current_time)
date_time OF A_change_data := CONCAT(TO STRING(10), ".") ! to add zero before the number for months less than 10
IF month < 10 THEN
date_time OF A_change_data := CONCAT(date_time OF A_change_data, "0", TO STRING(10), "", TO STRING(10), "", TO STRING(10), "", TO STRING(10), "", TO STRING(10))
ELSE
date_time OF A_change_data := CONCAT(date_time OF A_change_data, "", TO STRING(10), "", TO STRING(10), "", TO STRING(10), "", TO STRING(10))
END IF
BEGIN Case_ID OF A_change_data := CASE_ID OF A_Responseing
data END IF
begin [NEW] begin reason for recom OF A_change_data > 0) THEN
NEW Reason for recom OF A_change_data := EXEC Sync 1 SQL insert into [A-change-data] (CASE_ID, CHANGE, object, object_ID, attributes, Required_max_hours, Difficultrom, Build_Cost, Recommendation, Reason_for_recom, Duration, date_time, old, new, old_value, new_value) VALUES (A_change_data.CASE_ID, A_change_data.CHANGE, A_change_data.object, A_change_data.object_ID, A_change_data.attributes, Required_max_hours, A_change_data.Duration, A_change_data.reason_for_recom, A_change_data.Duration, A_change_data.date_time, old, new, old, new) END SQL
****** input the data in [A-CM-Alternatives] table IF (Conf(Required_max_hours OF A_change_data) > 1) THEN
begin PREDECESSOR OF A_change_data := "NULL" EXEC Sync 1 SQL insert into [A-CM-Alternatives] (CASE_ID, PREDECESSOR, CHANGE, PREDECESSOR_OF_A change_data, object, object_ID) VALUES (A_change_data.CASE_ID, PREDECESSOR, CHANGE, PREDECESSOR_OF_A_change_data, A_change_data.object, A_change_data.object_ID) END SQL
******** make sure that the change has not been made before
visible OF design data window = FALSE

**Forget output OF design data window**

**** end of method

**END**

ATTRIBUTE Num_num NUMERIC
** Attribute NumNew NUMERIC
** Attribute Idspline FROM STRING
** Attribute Effective attribute value STRING
** Attribute Effective attribute value NUMERIC
** Attribute No77 NUMERIC
** Attribute Does existing data display SIMPLE

**WHEN CHANGED**

Required max_hours OF A_change_data =

Required max_hours OF A_change_data data_retrieved

**DURATION OF A_change_data = DURATION**

**OF A_change_data = retrieved**

**Difference in Build Cost OF A_change_data = difference in build cost**

**OF A_change_data = retrieved**

**Recommendation OF A_change_data = retrieved**

Reason for recom of A_change_data =

Reason for recom of A_change_data =

Reason for recom of A_change_data =

Visible OF design data window = TRUE

**output OF design data window = data** for a design change display

Visible OF existing design window = FALSE

**FORGET output OF existing design window**

**END**

ATTRIBUTE second alternative SIMPLE

**WHEN CHANGED**

**IF**( CONF(first_alternative_value) = 0) THEN

BEGIN

**put up OF message box 1 = TRUE**

**IF** button selected OF message box 3 IS yes

THEN

**put up OF message box 1 = FALSE**

**yes for second alternative = "YES"**

**END**

**IF** button selected OF message box 3 IS no

THEN

**put up OF message box 1 = FALSE**

**ELSE**

BEGIN

**test OF message box 1 = "No first alternative was given"**

**put up OF message box 1 = TRUE**

**END**

**** end of method

**END**

ATTRIBUTE yes for second alternative STRING

**ATTRIBUTE fine alternative value STRING**

**ATTRIBUTE fine alternative STRING**

**ATTRIBUTE general message SIMPLE**

**WHEN CHANGED**

**BEGIN**

**IF**( CONF(cause OF the effect OF discipline message) < 0) THEN

**cause OF the effect OF discipline message = "Undefined"**

**message builder OF required message =**

**CONCAT( "is VIRTUAL MODE. Ti

be attribute ", Effective attribute value OF discipline message, ", of the build

ding component ", Effective class OF discipline message, ", (", Changed insta

stance ID OF discipline message, ")

has been changed from the value (", old value OF discipline

message, ")

to the value (", new value OF discipline message, ").

Such a change may affect the design of the ",

discipline TO OF discipline message, ")

system.

The possible cause of the effect is ",

cause OF the effect OF discipline message TO message.

Please indicate the expected modifications in the
building components."

**END**

ATTRIBUTE specific message SIMPLE

**WHEN CHANGED**

**BEGIN**

message builder OF required message =

**CONCAT("is VIRTUAL MODE. Ti

be attribute ",, Effective attribute value OF discipline message, ", of the build

ing component ",, Effective class OF discipline message, ", (", Changed insta

stance ID OF discipline message, ")

has been changed from the value (", old value OF discipline

message, ")

to the value (", new value OF discipline message, ").

According to a previously defined specific rule, such a
change may effect the design of the sub

ute attribute OF discipline message, ", of the build

ing component ",, Effective attribute value OF discipline message, ", (", Changed insta

stance ID OF discipline message, ")

Please indicate the expected modifications in the
affected building comp

units.

**END**

**attachment OF valuebox 15 = message builder OF required message**

**END**

ATTRIBUTE effect of menu item SIMPLE

**WHEN CHANGED**

**BEGIN**

**object OF A_change_data = label OF menu item

column as monitored object = TRUE**

**END**

**** end of method

**END**

ATTRIBUTE project_name STRING

**WHEN CHANGED**

**BEGIN**

**data source OF 1506a 1 = project_name**

title OF new case window = CONCAT( "New

Case for Project ":, project

name)

output OF Select a project window = select_new_or_responding_display

**END**

**ATTRIBUTE go ahead SIMPLE

WHEN CHANGED**

**BEGIN**

**action OF 1506a 2 GENERAL IS connect = TRUE**

**EXEC 1506a 2 GENERAL SQL SELECT**

Project_name FROM Current_project

**END SQL INTO Current projects**

(Projec

name)

**visible OF main window = FALSE**

**visible OF Select a project window = TRUE**

**output OF Select a project window = select_project_display**

**END**

**END**

**INSTANCE border 1 ISA border**

**WITH style IS group**

**WITH perspective IS in**

**WITH border width = 1**

**WITH background color = 192,192,192**

**WITH fill color = 192,192,192**

**WITH highlight color = 235,235,235**

Ahmed Mohktar, Ph.D Thesis .....243
WITH pen color := 0,0,128
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH clipped := TRUE
WITH attachment := old_value OF A_change_data_retrieved
WITH location := 120,145,195,170

INSTANCE valuebox 25 ISA valuebox
WITH pen color := 0,0,128
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH frame := TRUE
WITH clipped := TRUE
WITH attachment := new_value OF A_change_data_retrieved
WITH location := 305,145,380,170

INSTANCE valuebox 26 ISA valuebox
WITH pen color := 233,0,0
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH frame := TRUE
WITH clipped := TRUE
WITH attachment := Case_ID
WITH location := 315,90,400,115

INSTANCE valuebox 27 ISA valuebox
WITH pen color := 255,0,0
WITH fill color := 192,192,192
WITH justify IS center
WITH font := "System"
WITH frame := TRUE
WITH clipped := TRUE
WITH attachment := Case_ID
WITH location := 505,215,565,345

INSTANCE main window ISA window
WITH location := 312,134,432,384
WITH menus [1] := UNDETERMINED
WITH style IS movable, sizeable, closable
WITH title := "Change-Monitoring"
WITH visible := TRUE

INSTANCE expand window ISA window
WITH location := 39,72,793,584
WITH menus [1] := UNDETERMINED
WITH style IS movable, sizeable, closable
WITH title := "Change-Monitoring Module"
WITH visible := FALSE

INSTANCE design data window ISA window
WITH location := 122,80,530,442
WITH menus [1] := UNDETERMINED
WITH style IS movable, sizeable, closable CF
FALSE
WITH title := "Data for A Design Change"
WITH visible := FALSE

INSTANCE existing design data window ISA window
WITH location := 114,44,522,450
WITH menus [1] := UNDETERMINED
WITH style IS movable, sizeable, closable
WITH title := "Similar Change Data"
WITH visible := FALSE

INSTANCE new case window ISA window
WITH location := 30,124,615,384
WITH menus [1] := UNDETERMINED
WITH style IS movable, sizeable, closable
WITH title := "New Case"
WITH visible := FALSE
WITH output := New Case display

INSTANCE responding window ISA window
WITH location := 20,5,615,455
WITH menus [1] := UNDETERMINED
WITH style IS movable, sizeable, closable CF
FALSE
WITH title := "Responding to Design Change"
WITH visible := FALSE
WITH output := responding display

INSTANCE Select a project window ISA window
WITH location := 151,149,407,383
WITH menus [1] := UNDETERMINED
WITH style IS movable, sizeable, closable
WITH title := ""
WITH visible := FALSE

END

Ahmed Mokhtar, Ph.D Thesis ....249
C.6. PLANNING AND SCHEDULING MODULE

Part A

WITH text STRING
WITH variable text STRING
WITH model style COMPOUND
app modal,
sys modal,
not modal
INIT app modal
WITH icon COMPOUND
cancellation,
information,
question,
stop sign
INIT escalation
WITH button COMPOUND
short retry ignore,
o,
oc cancel,
retry cancel,
yes no,
yes no cancel
INIT ok
WITH default button COMPOUND
two,
three
INIT one
WITH button selected COMPOUND
short,
oc cancel,
o,
retry,
yes
WITH put up SIMPLE
CLASS navigate INHERITS add on
WITH data attachment CLASS REFERENCE
WITH action COMPOUND
first,
previous,
next,
last
WITH instance number NUMERIC
WITH number instances NUMERIC
CLASS alternative
WITH predecessor STRING
ARRAY SIZE 1
WITH alternative change STRING
ARRAY SIZE 1
CLASS temporary current
WITH name STRING
SHARED ATTRIBUTE CASE_ID NUMERIC
WHEN CHANGED
BEGIN
DELETE A_CPMData := TRUE
END
SHARED ATTRIBUTE project_name STRING
WHEN CHANGED
BEGIN
data source of IsDocument := project_name
action of IsDocument := TRUE
EXEC IsDocument SQL select DISTINCT CASE_ID
from [A_CPM] where CASE_ID = CASE_ID
END SQL INTO available change case id
title of main window := CONCAT("Project ":
project name)
output of main window := select_case display
END
ATTRIBUTE i NUMERIC
ATTRIBUTE f NUMERIC
ATTRIBUTE s NUMERIC
ATTRIBUTE delete A_CPMData SIMPLE
WHEN CHANGED
BEGIN
EXEC IsDocument SQL delete FROM [A-CPMData]
WHERE CASE_ID = domain.CASE_ID
END SQL
WHERE data elements := TRUE
END
ATTRIBUTE fill data elements SIMPLE
WHEN CHANGED
BEGIN
**** this is to create data elements that have each predecessor and all
**** it possible alternative (each predecessor and one of its alternatives)
**** is considered as a separate element (instance)
EXEC IsDocument SQL select PREDECESSOR,
CHANGE, alternative CHANGE FROM [A-CPM-Alternatives]
WHERE CASE_ID := CASE_ID
END SQL INTO data elements
(predecessor, CHANGE, alternative CHANGE)
**** make current data alternative in use :=
alternative(1) CHANGE
FOR (i := 1 TO number instances OF navigate
begin
instance number OF navigate := i
current data alternative in use OF
data elements := CHANGE OF data elements
END
**** make first alternative := TRUE
END
ATTRIBUTE make first alternative SIMPLE
WHEN CHANGED
BEGIN
***** Define the field ALTERNATIVE in A-
CPMData table in the database
alternative := 1
***** select the starting data elements
***** also put alternative #1 as the
current data alternative in use OF data elements
FIND data elements
WHERE predecessor OF data elements =
"NULL"
WHERE FOUND
current data alternative in use OF
data elements := CHANGE OF data elements
current_cpm_alternative OF data elements :=
CONCAT(\"ALTERNATIVE\", TO STRING(
alternative))
MAKE current
WITH name :=
current data alternative in use OF data elements
EXEC IsDocument SQL insert into [A-CPMData]
(PREDECESSOR, CHANGE, ALTERNATIVE,
CASE_ID) values
(data elements, predecessor,
data elements, current data alternative in use,
data elements, current_cpm_alternative,
domain.CASE_ID)
END SQL
END
**** Repeat to fill all the A-CPMData table in the
data base
**** select all the elements that come after the
previously used ones (called current)
**** the assumption here is that any particular
change will always result in the same changes
Part B

SVersions5
Slocations are pixels

CLASS available_change_case id
WITH CASE ID NUMERIC

CLASS border Inherits add on, display item
WITH style COMPUND
picture frame, edit control, group, shadow
INIT picture frame
WITH perspective COMPUND in, on
INIT in
WITH border width NUMERIC
INIT 1
WITH background color COLOR
INIT 255,255,255
WITH fill color COLOR
INIT 255,255,255
WITH highlight color COLOR
INIT 255,255,255
WITH shadow color COLOR
INIT 255,255,255

CLASS current
WITH same STRING

CLASS Current_projects
WITH project name STRING

CLASS data elements
WITH professor STRING
WITH CHANGE STRING
WITH alternative CHANGE STRING
WITH current_open Alternative STRING
WITH current_data_alternative_in_use STRING

CLASS finished
WITH name STRING

CLASS Isocode Inherits add on
WITH data source STRING
WITH width STRING
WITH password STRING
WITH connection string STRING
WITH connection prompting SIMPLE
INIT FALSE
WITH auto commit SIMPLE
INIT TRUE
WITH actions COMPUND
connect, disconnect
WITH records NUMERIC
WITH transaction COMPUND
connect, rollback
WITH append SIMPLE
INIT FALSE
WITH status STRING
WITH show error SIMPLE
WITH default error handling SIMPLE
INIT TRUE
WITH error message STRING
WITH native error NUMERIC
WITH trace file STRING

CLASS last alternative data
WITH name STRING

CLASS LastHighLevel
CLASS message box Inherits add on
WITH title STRING
WITH text STRING
WITH variable text STRING
WITH modal style COMPUND
app modal, sys modal, not modal
INIT app modal

WITH icon COMPUND
exclamation, information, question, stop sign
INIT exclamation
WITH button COMPUND
short retry ignore, ok, ok cancel, retry cancel, yes no
INIT ok
WITH default button COMPUND
one, two, three
INIT one
WITH button selected COMPUND
short, cancel, ignore, exit, ok, retry, yes
WITH put up SIMPLE

CLASS navigate Inherits add on
WITH dass attachment CLASS REFERENCE
WITH action COMPUND
first, previous, next, last
WITH instance number NUMERIC
WITH instance number NUMERIC

CLASS other alternatives
WITH professor STRING
ARRAY SIZE 1
WITH alternative change STRING
ARRAY SIZE 1

CLASS temporary current
WITH name STRING

SHARED ATTRIBUTE CASE_ID NUMERIC WHEN CHANGED BEGIN
delete A_CPMData = TRUE END

SHARED ATTRIBUTE project name STRING WHEN CHANGED BEGIN
data source Of Isocode 1 = project name action Of Isocode 1 IS connect = TRUE EXEC Isocode 1 SQL select DISTINCT(CASE_ID) FROM [A-CPM-Alternatives] END SQL INTO available change case id title Of main window = CONCAT("Project :") project name output Of main window = select_case display END

ATTRIBUTE i NUMERIC
ATTRIBUTE start calculations SIMPLE WHEN CHANGED BEGIN
delete A_CPMData = TRUE END

ATTRIBUTE ii NUMERIC
ATTRIBUTE y NUMERIC
ATTRIBUTE ii STRING
ATTRIBUTE delete A_CPMData SIMPLE WHEN CHANGED BEGIN
EXEC Isocode 1 SQL delete * FROM [A-CPM-Alternatives] WHERE CASE_ID = domain.CASE_ID END SQL

fill data elements = TRUE END

ATTRIBUTE fill data elements SIMPLE

WHEN CHANGED BEGIN
*** this is to create data elements that have each professor and all
*** the alternative possible (each professor and one
*** of its alternatives
*** is considered as a separate element (instance)
EXEC Isocode 1 SQL select PREDECESSOR, CHANGE, alternative CHANGE FROM [A-CPM-Alternatives] WHERE CASE_ID = CASE_ID END SQL INTO data elements
(predecessor, CHANGE, alternative CHANGE)

*** make current data alternative in use = alternative1(I) CHANGE
FOR (i = 1 TO number instances OF navigate)
BEGIN
instance number OF navigate = i
current data alternative in use OF data elements = CHANGE OF data elements
END

make first alternative = TRUE
END

ATTRIBUTE make first alternative SIMPLE WHEN CHANGED BEGIN
***** Define the field ALTERNATIVE in A
CPM data table in the database
alternative1 = 1
***** select the starting data elements
***** also put alternative #1 as the
current data alternative in use OF data elements
FIND data_element
WHERE professor OF data_elements = "NULL"
WHEN FOUND
current data alternative in use OF data_elements = CHANGE OF data_elements
current open Alternative OF data_elements = CONCAT("ALTERNATIVE", # STRING(alternative1))
MAKE current
WITH name = current data alternative in use OF data_elements
EXEC Isocode 1 SQL insert INTO [A-CPM-data] (PREDECESSOR, CHANGE, ALTERNATIVE, CASE_ID) VALUES (data_elements.professor, data_elements.current data alternative in use, data_elements.current open Alternative, domain.CASE_ID)
END SQL
FIND END

***** Repeat to fill all the A-CPM-data table in the
data base
***** select all the elements that come after the
previously used ones (called current)
***** the assumption here is that any particular
change will always result in the same changes
WHILE (name OF current = "STOP")
BEGIN
FOR (i = 1 TO number instances OF navigate)
 BEGIN
instance number OF navigate = i
FIND data elements
WHERE professor OF data_elements = name OF current
WHEN FOUND
current data alternative in use OF data_elements = CHANGE OF data_elements
current open Alternative OF data_elements = CONCAT("ALTERNATIVE", # STRING(alternative1))
MAKE temporary current
WITH name = CHANGE OF data_elements
EXEC Isocode 1 SQL insert INTO [A-CPM-data] (PREDECESSOR, CHANGE,
ALTERNATIVE, CASE_ID) values
(data_elements, predecessor,
data_elements.current data_alternative_in_use,
data_elements.current_cpm_alternative,
|domain,CASE_ID | END SQL |

FIND END
***** end of FOR END
***** make the temporary current as current after making sure that it has not been used before
FOR (ii := 1 TO number instances OF navigate

BEGIN
instance number OF navigate ii := ii
WHERE name OF finished = name OF temporary current
WHEN FOUND
yy := 0
WHEN NONE FOUND
yy := ii
FIND END

IF (yy = 1) THEN
BEGIN
MAKE current WITH name := name OF temporary current
MAKE finished
END

else put the current data_alternative_in_use
OF data_elements
WHERE predecessor OF data_elements = "NULL"
WHEN FOUND
current_cpm_alternative OF data_elements := CONCAT("ALTERNATIVE", TO STRING(alternative))
MAKE current
WITH name :=
current_data_alternative_in_use OF data_elements
EXEC Sodbc 1 SQL insert into [A-CPMdata] (PREDECESSOR, CHANGE, ALTERNATIVE, CASE_ID) values
(data_elements, predecessor,
data_elements.current_data_alternative_in_use,
data_elements.current_cpm_alternative,
|domain,CASE_ID | END SQL |

FIND END

***** repeat to fill all the A-CPMdata table in the data base
****** select all the elements that come after the previously used ones (called current)
****** check that any particular change will always result in the same changes

WHILE (name current = "STOP") BEGIN
FOR (ii := 1 TO number instances OF navigate

BEGIN
instance number OF navigate ii := ii
WHERE name OF finished = name OF temporary current
WHEN FOUND
current_cpm_alternative OF data_elements := CONCAT("ALTERNATIVE", TO STRING(alternative))
MAKE temporary current
WITH name :=
current_data_alternative_in_use OF data_elements
EXEC Sodbc 1 SQL insert into [A-CPMdata] (PREDECESSOR, CHANGE, ALTERNATIVE, CASE_ID) values
(data_elements, predecessor,
data_elements.current_data_alternative_in_use,
data_elements.current_cpm_alternative,
|domain,CASE_ID | END SQL |

FIND END
***** make the temporary current as current after making sure that it has not been used before
FORGET current
