A Model Checking Tool to detect and resolve potential Redundancies in Entity Relationship models.

A dissertation submitted in partial fulfilment
of the requirements for the Open University's
Master of Science Degree
in Computing for Software Development

Frédéric Maréchal
(T8813518)

11 March 2008

Word Count: **14,707**

Preface

I would like to thank my partner, Lucille Boquet, who has supported me relentlessly and patiently throughout these last two difficult years. My parents, Claude and Nicole Maréchal, who have given me the strength to succeed through hardship. My tutor, Simon Dugmore, who has guided my footsteps throughout this thesis. Finally, Jonathan Towler for his time and patience while editing this document.

Table of Contents

Preface ii

List of Figures vi

List of Tables vii

Chapter 1 Introduction 1

1.1 The Problem Domain 1

1.2 Definition of Terms 2

1.3 Background to the Research 3

1.4 Limitation of Existing Approaches 7

1.5 Aims and Objectives of the Research Project 8

1.6 Contribution to Knowledge 10

1.7 Overview of the Dissertation 11

Chapter 2 The Literature Review 12

2.1 The Literature Research 12

2.1.1 The Methodology 12

2.1.2 The Resource Evaluation 15

2.2 Review of Current Methodologies on Superfluous Relationships Removal 15

2.2.1 Relationships Elimination through Minimal Covering of FD-graphs 16

2.2.2 Relationships Elimination through the Real-World Knowledge 20

2.2.3 Relationship Elimination through the Analysis Cardinality and Participation Constraints 24

2.2.4 Relationship Elimination through the Analysis of Shared Attributes and Inclusion Constraints 27

2.2.5 Relationship Elimination - The Hybrid Approach 31

2.3 The Research Question 34

2.4 Summary 35

Chapter 3 Research Methods 38

3.1 Literature Review 38

3.2 The Scope of the Experiment 41

3.3 The Experiment Environment 43

3.4 The Experimental Framework 44

3.5 Controlled Experiments 45

3.5.1 'Discovery Quality' Methodology 46

3.5.2 Scalability Measurement Methodology 47

3.5.3 The Experiment Limitations 48

3.6 Experiment Implementation 48

3.6.1 The Prototype Objectives 49

3.6.2 High Level Functionality 49

3.6.3 High Level Component Structure 50

3.6.4 The Algorithm Architecture 52

3.6.4.1 Algorithm Class Hierarchy 52

3.6.4.2 Sequence Diagram 56

3.6.5 Enhanced Bowers (2002) Algorithm 59

3.6.5.1 The Algorithm Description 59

3.6.5.1.1 The deletion of unwanted paths 59

3.6.5.1.2 Ignore potential redundant paths that live on the equivalent composed path 65

3.6.5.1.3 Ensure uniqueness of potential redundant relationships 67

3.6.5.2 The Algorithm Implementation Logic 68

3.6.5.3 The Model Limitations 71

Chapter 4 Data Collection 72

4.1 Data Sources 72

4.2 Data Collection Process 73

4.3 Preliminary Analysis 75

4.3.1 Efficiency Analysis 75

4.3.2 'Quality Discovery' Analysis 75

Chapter 5 Results 78
5.1 Execution Time Measurement Result 78
5.2 Quality Discovery Result 81
5.2.1 Potential Redundant Path Analysis 81
5.2.2 Equivalent Composed Path Analysis 84
5.3 Validation 86
5.3.1 Discovery Capability Comparison 86
Chapter 6 Conclusions and Recommendations 88
6.1 Result Summary 88
6.2 Recommendations 89
6.2.1 Algorithm Implementation Improvements 89
6.2.1.1 Potential redundant relationship enhancement 89
6.2.1.2 Equivalent composed paths enhancement 90
6.2.2 Architecture Modification 91
6.2.3 Scalability Improvements 91
6.2.4 Performance Improvements 92
6.2.5 Complexity Analysis 92
6.3 Future Research 93
Bibliography 97
Index 98
Appendix A – Quality Discovery Test Cases 105
Appendix B – Quality Discovery Test Cases Input File 118
Appendix C – The Quality Discovery Collected Raw Data 119
Appendix D – The Quality Discovery Raw Data Transformation 138
Appendix E – The Execution Time Test Cases 139
Appendix F – The Execution Time Results 140
Appendix G – The ERD Schema Business Object 141

Appendix H – Plug-in Installation 142

Appendix I – Training Manual 142

Appendix J – Application Code 143

List of Figures

Figure 1- Potential Redundant Relationship 4

Figure 2 – Non-redundant relationship (Connolly and Begg, 2000) 5

Figure 3 - A relevance tree for the subject of 'Detection and resolution of potential arcs in ERDs' 13

Figure 4 – Example of an FD-graph (Ausiello, d'Atri and Sacca, 1983) 16

Figure 5 – Example of a redundant (a) and a "nonredundant" (b) FD-graph (Ausiello, d'Atri and Sacca, 1983) 17

Figure 6 – Minimum covering of a "nonredundant" FD-graph (Ausiello, d'Atri and Sacca, 1983) 18

Figure 7 - General method representing domain knowledge (Lloyd-Williams, 1997). . 22

Figure 8 – Example of a redundant relationship (R3) in a single binary cyclic 'n-relationship' path (Dullea and Song, 1997) 25

Figure 9 - Discovery of potential redundant relationships in vertical and horizontal shortcuts. 29

Figure 10 – Venn diagram that displays the current ability of the different approaches to analyse different types of relationships to search for potential redundant relationship. 35

Figure 11- A relevance tree for the subject of 'Implementation of an algorithm for the detection and resolution of potential arcs redundancies in ERDs 40

Figure 12 – PC Specification........ 43

Figure 13– The experiment framework 44

Figure 14– The experiment framework 47

Figure 15– High level interaction between the framework major components. 50

Figure 16– The algorithm class hierarchy........ 53

Figure 17– The graph operation class hierarchy 55

Figure 18 – Sequence Diagram for the discovery of potential redundant relationships in an ERD (from the user to the GraphOperation object) 58

Figure 19 – Path deletion conditions (assuming there is no recursive relationship on Entity A). One of these conditions suffices for the path to be deleted. 63

Figure 20– Schema representing a cycle added to 'Test I' 67

Figure 21 - Example of the discovery of a potential redundant path and its equivalent composed path for the Test R (for the order 2). 74

Figure 22 – Execution time in milliseconds (ms) for a sparse ERD (5 entities) 80

Figure 23 - Execution time in milliseconds (ms) for a dense ERD, where the number of relationships is double the number of entities (as there are two relationships per entity) 80

Figure 24 - Potential Path Redundancy Discovery Comparison 81

Figure 25 - Potential Equivalent Path Redundancy Discovery Comparison 85

List of Tables

Table 1 – List 'unambiguous' redundant relationships (in column RE1E4) based on the cardinality constraints of a cyclic four entities relationship diagram. 1 means the cardinality is one. M means the cardinality is many (Dullea and Song, 1997). 25

Table 2 – List 'unambiguous' redundant relationships the special of 'M-1, M-1, M-1' relationships between the entities R1, R2 and R3 and their individual participation. A plain circle (●) means the relationship is mandatory. (○) means the participation is optional (Dullea and Song, 1997). 26

Table 3 – Algorithms used for the test 45

Table 4– Matrices representation of paths allowing for cycles 61

Table 5– Matrices representation of paths allowing for cycles and loops 63

Table 6– Truncated matrices representation of paths allowing for cycles and loops and multiple relationships 64

Table 7– Matrices representation of 'Test I', for the orders 1 and 2 66

Table 8 – Initial Boolean matrix for an ERD 67

Table 9 - The success, failure and missed metrics definition for potential redundant paths and equivalent composed paths 76

Table 10– Summary of matrices representation of 'Test I', for the order1 and 2 83

Table 11 – Summary of the comparison of the studied approaches discovery abilities. 87

Abstract

A fundamental measure of database design success lies in the production of an accurate conceptual model that does not exhibit unnecessary relationships. Numerous approaches have been proposed to discover potential redundant relationships in an Entity Relationship Diagram (ERD). However, they reveal restrictions that limit their usage in real life models. The literature review shows that the functional dependency graph approach systematically removes longest paths - whether they are potentially redundant or not. The study of the ‘real-world’ knowledge demonstrates that the discovery strategy success rate is highly correlated to the quality of the relationship name dictionary definitions. Finally, the analysis of “cardinality constraints” and the “shared attributes and inclusion constraints” methodologies are restricted to single relationships in a cycle.

Although the ERD adjacency-matrix multiplication strategy is currently limited to single relationships without a loop or a cycle, it seems to be the most promising in systematically discovering potential redundant relationships with more complex ERDs. The Graph Edge Validator 2008 (GEV2008) prototype implements an enhanced version of the adjacency-matrix strategy, introduced by Bowers (2002). This version allows for diagonal matrix elements and multiple relationships. Therefore, GEV2008 can locate potential redundant relationships in loops or cycles, in the context of single and multiple relationships. Furthermore, the enhanced algorithm automatically removes ‘look back’ and ‘recursive look back’ paths. These types of paths are generated by the adjacency-matrix multiplication, although they do not exist in the original ERD.

The result of the test bench looks promising. GEV2008 detects most of the potential redundant relationships and equivalent composed paths, in dense and sparse ERDs. The failure rate is currently significant, i.e. 30% on average over the test case suite. However, it should be noted that this is due to only one reason: the loss of relationship context of the first entity in an equivalent composed path. This thesis proposes a simple solution to remove this limitation, which should ensure a significantly higher success rate. This study also demonstrates that the current execution time growth rate is proportional to $O(R^3)$, when the number of entities remains constant and there is no loop. However, the growth rate becomes exponential when the number of entities is doubled. $O(R^3)$ refers to the “big-Oh” notation, where O(…) is an asymptotic function that represents the “big-picture” of the execution time growth rate given an input size R.

Chapter 1 Introduction

1.1 The Problem Domain

The implementation of a relational, hierarchical or object-oriented database requires the representation of the problem domain in a complete, consistent and accurate manner. The difficulty in database design is to reconcile the designers, programmers and end-users view into a common representation of the problem domain, as pointed out by Connolly and Begg (2000).

A crucial measure of database design success lies in the production of an accurate model that does not store unnecessary information, Dullea and Song (1997) and Goelman and Song (2004). There are two flavours of redundancy: data redundancy and relationship redundancy. Redundant information has two major impacts in databases. First, on the theoretical side, it prevents the database implementation from being in Third Normal Form (3NF), as underlined by Connolly and Begg (2000). In particular, the existence of a redundant arc in a ternary (or '*n-ary*') relationship produces a transitive dependency across entities. Second, it makes management of the integrity constraints more complex. Each extra relation may support behaviour properties such as cascade insertion, deletion and update operations. Consequently, when denormalisation is not required, the designer should strive to minimise the duplication of large data items that require frequent amendment, as it will reduce the databases update efficiency (Burleson, n.d.).

Data redundancy has been studied in depth through the literature and the development of commercial databases (Dullea and Song, 1997). However, as Batra and Zanakis

(1994) and Siau and Wand and Benbasat (1995) demonstrated, relationships have not received the same degree of attention.

1.2 Definition of Terms

The Entity Relationship Diagram (ERD) analysis and design modelling methodologies are usually the *de facto* techniques employed to achieve this goal. An ERD is composed of four major elements:

- Entities (relations) model 'real world' objects (e.g. employee, company, client, etc.) or intangible/abstract concepts (e.g. investment strategy, return on investment, etc.). They contain attributes and tuples (i.e. records). To be in the First Normal Form (1NF), the table attributes must be unique, simple (i.e. not composed) and single-valued. Furthermore, each tuple must be unique.

- Attributes are properties of an entity or a relationship.

- Relationships (arcs) represent meaningful associations among entities, which can be recursive. In addition, more than one relationship may exist between two entities, as long as they have distinct semantics (otherwise some of them may be redundant).

- Constraints are consistency rules that the database is not permitted to violate.

Teorey, Yang and Fry (1986) show that a relationship contains four elements:

- The relationship cardinality. This defines the number of possible relationships for each participating entity (e.g. '*one-to-many*', '*many-to-many*').

- The Participation. This indicates whether all or some entity occurrences participate in the relationship, i.e. the relationship is mandatory or optional (Connolly and Begg, 2000).

- The relationship degree. This is the number of entities associated with one relationship. Unary: (one), binary (two), ternary (three) relationships are examples of the *n-ary* (*n*) relationships superset.

- The recursive nature of the relationship. A recursive relationship is an entity associated with itself. This is also referred as a 'loop'.

1.3 Background to the Research

In the context of an ERD, a relationship is redundant when both its "mapping and its semantics are implied by some alternative path" (Bowers, 2002; Bowers, 2000). Figure 1 shows examples of the three main types of potential relationship redundancies.

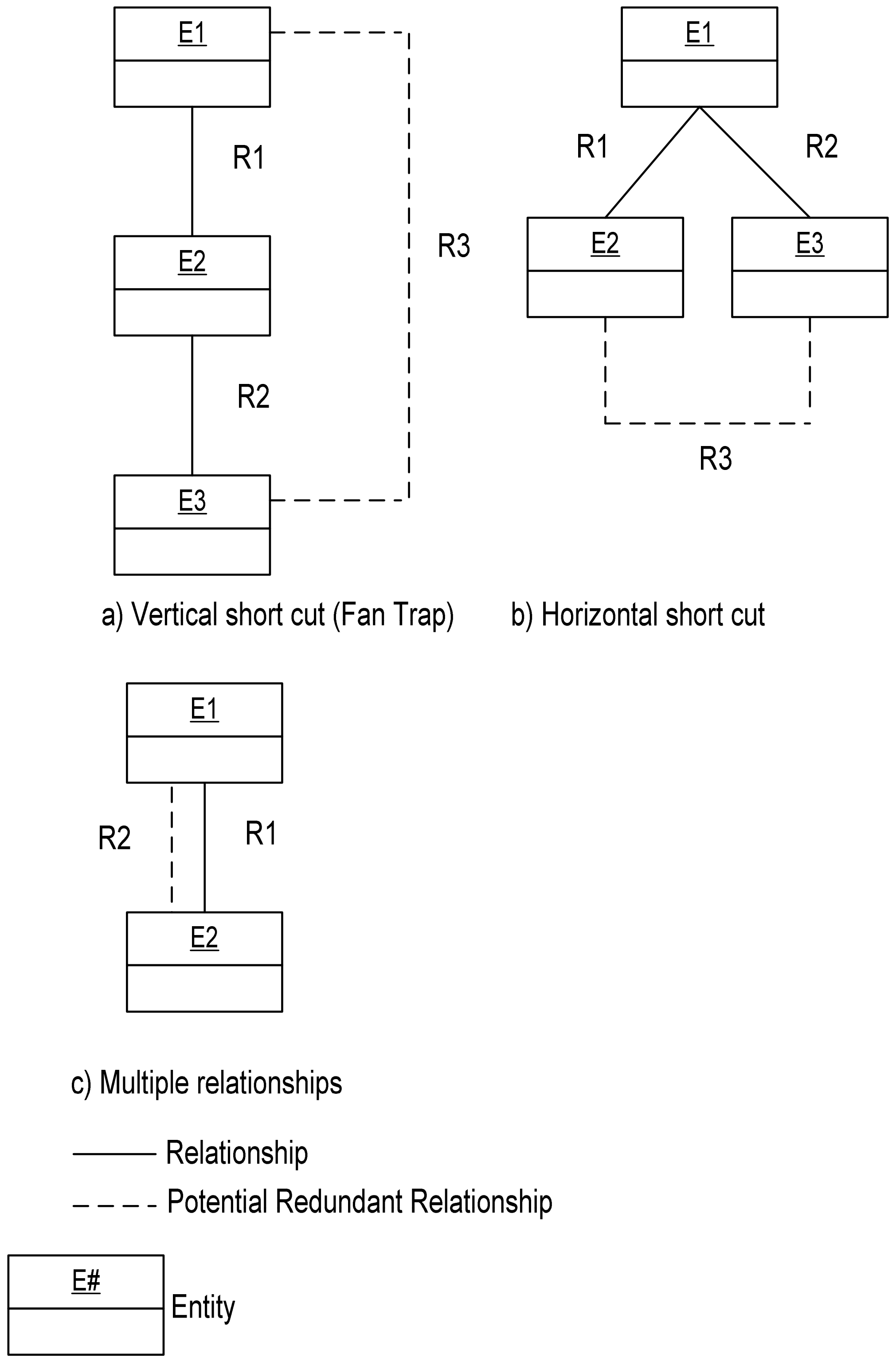

E1
R1
E2
R2
E3
R3
a) Vertical short cut (Fan Trap)
E1
R1
R2
E2
E3
R3
b) Horizontal short cut
E1
R2
R1
E2
c) Multiple relationships
Relationship
Potential Redundant Relationship
E#
Entity

Figure 1- Potential Redundant Relationship

Sections a) and b) of Figure 1 exhibits the “vertical shortcut” or the “horizontal shortcut” (Rosenthal and Reiner, 1994), where the R3 relationship may be implied by the R1 and R2 relationships. Section c) displays the case of multiple relationships between two entities; where relation R2 could be a duplicate of relation R1. This representation aims at demonstrating the redundant cases in a simple and an intuitive fashion. In reality, each of these archetypes could be combined to produce a more complex picture.

It is important to emphasise the concept of ‘potential’ redundant relationships.
Connolly and Begg (2000) raise the importance of the time dimension when assessing a relationship redundancy. Figure 2 illustrates this concept. The ERD is made of three entities; a Man, Woman and Child.

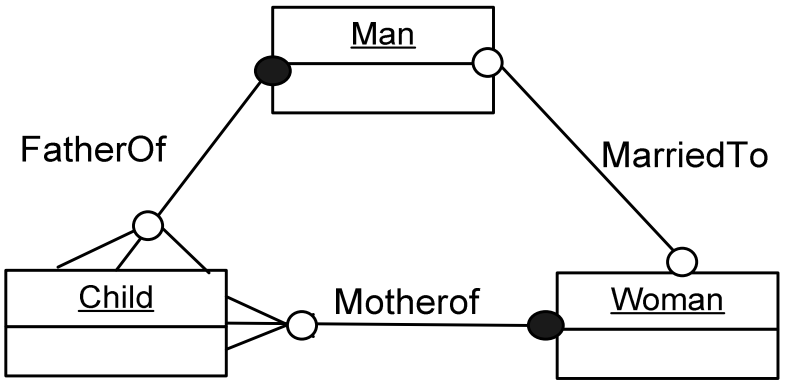


Figure 2 – Non-redundant relationship (Connolly and Begg, 2000)

This schema shows that the father may have children from a previous marriage, furthermore the father may not be married to the mother, or the father could be married to someone other than the mother. Therefore none of these relationships are redundant, although the user may intuitively think that one of these relationships is too many.

Morris (1999) considers that the normalisation process is a necessary step to improve the quality of the initial relational schema. However, it is unclear how much value this process may offer in the discovery of redundant arcs, as it suffers from two major shortfalls. On the one hand, this process takes place during the implementation phase, without the involvement of the business user. On the other hand, it purely focuses on attribute dependencies. Hence, it does not take into account the relationship semantics (Bowers, 2002).

Different methodologies have been proposed in the literature, which locate potential redundant arcs in an ERD. However, none of them proposes an all encompassing and systematic methodology to resolve this problem. Three types of approaches are available in the literature. The first approach formalises the ERD as a hyper graph (Vatanawood and Rivepiboon, 2004). The graph theory models ERDs into a set of vertices and edges, equivalent to entities and relationships in ERDs. However, it does not model the relationships cardinality/participation. The weighted graph methodology, described by Goodrich M. and Tamassia R (2006) can be used to find the shortest path from some starting vertex to every other vertex in a connected graph. However, it assumes prior knowledge of the weight of each edge.

The second approach relates to the study of some of the fundamental relationship elements. The most intuitive one is the analysis of the relationship names. The design of an ERD can lead to the creation of synonym relationships. That is the existence of relationships with the same or nearly the same meaning. For example, two entities, E1 and E2, could bear two relationships that have the same semantic meaning; R1, named 'Manage' and R2, named 'Supervise'. Therefore, one of these two relationships is redundant. Each synonym case should be reviewed and the user should establish whether the relationship should be transformed to an 'is-a' relationship type, or declared as redundant and eliminated (Storey 1993). An ‘is-a’ relationship between two entities, E1 and E2, indicates that the entity E1 is a ‘child’ of E2 (i.e. it shares all the properties of E1 and extends E1 with its own properties).

The method, formulated by Rosenthal and Reiner (1994), exposes a “composition theorem for relationships that share common attributes” and demonstrates how to utilise this algorithm to unveil “semantically-unnecessary” and “content-unnecessary” relationships. This theorem facilitates the discovery of redundant arcs in vertical and horizontal shortcuts, based on the analysis of shared common attributes within a three node path, where an attribute inclusion dependency exists between two entities exists.

The third type is a hybrid approach. Bowers (2002), drew on techniques from the graph theory to navigate through the entities and relationships, as a graph algorithm would navigate through nodes and vertices. The solution combines a dual approach. It mixes the use of an algorithm (based on adjacency-matrix multiplications) to compare relationships semantic, and detect potential redundant relationships. It also relies on user intervention to decide on the relevance of these relationships. The algorithm that detects potential redundant relationships starts by comparing each relationship with its equivalent composed path, and then flags paths that have the same signature, as being potentially redundant.

1.4 Limitation of Existing Approaches

The proposed approaches show a number of limitations:

- The Goodrich M. and Tamassia R. (2006) methodology requires the user to make a decision about the relevance of each relationship in the ERD. This is equivalent to deciding in advance which relationship is redundant. Furthermore, the shortest path is not necessarily the right answer, as the user may prefer a longer route to a shorter equivalent one.

- The approach suggested by Storey (1993) resolves the 'most obvious' case of potential redundancy. However, this technique is heavily dependent on the quality and/or relevance of the dictionary.

- The Rosenthal and Reiner (1994) methodology suffers from the requirement for designers and users to define inclusion constraints during the ERD design phase.

- The solution proposed by Bowers (2002) is incomplete. It does not take into account either loops or cycles. Sedgewick (2004) defines a cycle as a path that is simple (i.e. where the vertices and edges are distinct), except that the first and last vertices are the same. Furthermore, it is restricted to single relationships between two entities.

1.5 Aims and Objectives of the Research Project

The aim of this research was twofold. First, I intended to extend and implement the solution suggested by Bowers (2002), as it seemed to offer the most improvement opportunities for searching through loops, cycles and multiple relationships. The algorithm implementation was written in Java. It was encapsulated into Java plug-ins, compatible with the Java Eclipse framework.

Due to time constraints, the prototype implementation only enables the user to store the ERD in a basic way (i.e. defining the ERD in an XML format). The interaction between the user and the implemented algorithm is supported by a basic user interface. It enables the user to:

- select an ERD, defined in an XML document
- view the list of potential redundant relationships and their equivalent composed paths
- remove unwanted paths

The second aim of this research was to analyse the capacity of the enhanced algorithm to discover potential redundant relationships in simple and complex scenarios, and compare the improved algorithm strengths and weaknesses against other approaches. Finally, the execution time spanning from simple sparse to complex dense ERDs was analysed to anticipate how the enhanced algorithm would cope with real life ERDs.

I originally planned to study the algorithm conciseness, complexity, flexibility and reliability through the use of software complexity measures/metrics (such as the Volume and Cyclomatic number). However, these subjects were not central to the topic of this dissertation. Furthermore, the major drawback of this analysis is that, even "in the case of widely studied metrics (e.g. LOC, McCabe's Cyclomatic Complexity, etc.), it is not universally agreed what they measure" (Mills, 1988).

The result of this analysis exposed weaknesses of the improved algorithm in terms of its theoretical model and implementation. The analysis also provided sufficient evidence to formulate a set of heuristics to further improve the proposed solution.

1.6 Contribution to Knowledge

This thesis presents a useful extension to the field of ERDs analysis. It provides composite approaches for analysing relationships, which leveraged the study of the relationships fundamental elements and the graph theory framework. In addition, it offers conclusions that indicate the circumstances in which the ‘improved’ algorithm should be used. It also shows the limits of the improved algorithm and suggested a set of proposals to improve the model.

I aim at three audiences:

- Students who want to learn more about the type of potential redundant relationships in ERDs, the effect it has on the quality of the physical schema and how they could be removed.

- Database designers who need an accurate, consistent, reliable, efficient and scalable algorithm that can be used to identify unnecessary relationship(s).

- Object-Oriented analyst/developer who uses UML class diagrams. The UML class diagram representation shares many attributes of the ERD modelling. Therefore, conclusions found in this thesis could be integrated in part (or in full) into specialist designer tools (such as Microsoft Visio^{TM}) to discover potential redundant relationships in class diagrams. My review of the ‘Microsoft Visio^{TM} 2003 embedded user guide’ indicated that this product does not currently offer such features.

- A Larger Audience: that focuses on redundant relationships within graphs. The audience spans from the pure graph theory field, to the many practical fields such as biotechnology (e.g. the formation of protein chains), the electrical/logical circuits engineering field, etc.

1.7 Overview of the Dissertation

This thesis is divided into four sections. The first section introduces to the existing methodologies that allow for discovering potential redundant relationships. The second section discusses the research methods to build the proposed algorithm. This section also describes:

- the software architecture
- the improved searching algorithms, it explains the variation made on the existing approach
- the algorithms logic
- the test cases

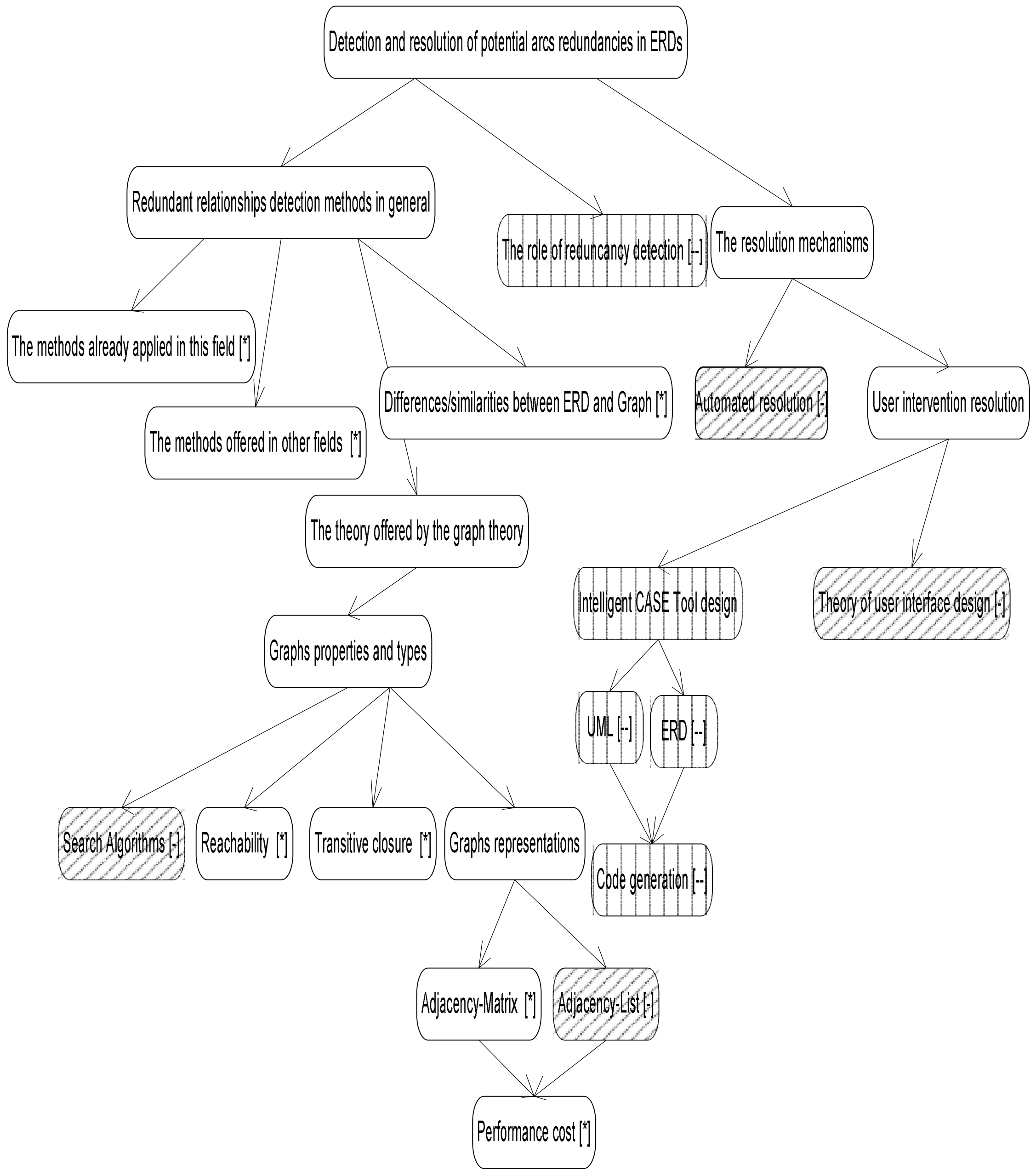
The third section focuses on the data collection. The data capture a set of metrics that establish the success rate of the discovery for different test case types. The final section concentrates on analysing the data and revealing the findings of the research.

Chapter 2 The Literature Review

2.1 The Literature Research

2.1.1 The Methodology

The literature review started with an analysis of the articles produced by Bowers (2002), which discussed the research topic. The focus of this thesis is the detection and resolution of potential arc redundancies in ERDs, by implementing and analysing the discovery capacity of the improved version of the algorithm initially proposed by Bowers (2002). Therefore, I created a relevance tree for this subject, as shown below (Sharp, Peters and Howard, 2002).


Detection and resolution of potential arcs redundancies in ERDs
Redundant relationships detection methods in general
The role of reduncancy detection [--]
The resolution mechanisms
The methods already applied in this field [*]
The methods offered in other fields [*]
Differences/similarities between ERD and Graph [*]
Automated resolution [-]
User intervention resolution
The theory offered by the graph theory
Intelligent CASE Tool design
Theory of user interface design [-]
Graphs properties and types
UML [--]
ERD [--]
Search Algorithms [-]
Reachability [*]
Transitive closure [*]
Graphs representations
Code generation [--]
Adjacency-Matrix [*]
Adjacency-List [-]
Performance cost [*]

[*] Area to form basis for research

[--] Area abandoned because better information available from other fields

[-] Area abandoned because deemed irrelevant

Figure 3 - A relevance tree for the subject of 'Detection and resolution of potential arcs in ERDs'

The above figure depicts the subjects/keywords that have been retained or abandoned. Subjects/keywords relating to the algorithmic detection: methods existing in the ERD field, other fields and the graph theory have been preserved, because they represent the foundation of knowledge on this topic. Abandoned subjects/keywords concern the creation of a rich user interface and/or automatic code generation. There is already an extensive literature base on these topics.

The next step was to follow the references listed in the text, and to read the abstract of each and every material found to decide whether it is in scope with the subjects/keywords selected above. Abstracts also enable to expand/restrict the scope and refine the relevance tree.

Citation indexes, such as *Web of Science*, *Cite Seer*, Scopus, *ACM Digital Library*, etc. were used to follow the article forwards, and gather literature that may confirm the article's approach and/or draw attention on potential limitations. It may also discuss the application of the article ideas in other fields.

General search engines (such *Google Scholar*, *Yahoo, Magellan, etc.)* and scientific databases (such as *ACM Digital Library*, *IEEE Xplore*, *Lecture notes in computer science, etc.*) were used to carry out searches based on the authors' name and/or selected subjects/keywords. In parallel, the functionality of specific search engines, such as *Web of Science,* were used to get email alerts on new resources, relating to the selected topic. This research was necessary to ensure that the latest available resources were available on the subject.

This process was reiterated for each of the resources found, until it was decided the research was maybe exhausted and the process could be suspended.

2.1.2 The Resource Evaluation

The literature review method consists of ensuring that each selected resources is authentic, trustworthy, complete and accurate. The method followed for this research is based on M801 (pp. 35-107) two ways approach:

- Use of internal criteria – the use of triangulation, i.e. cross checking each resource with another that independently supports the assertions of the first.
- Use of external criteria – checks that the method in use in a resource is sound, rigorous, does not contain ambiguities, contradictions, inconsistencies and that assumptions are clearly defined.

2.2 Review of Current Methodologies on Superfluous Relationships Removal

The following sections analyse the current body of knowledge existing on the topic. The ERD notation selected in this thesis is based on the notation proposed in M876 (Block 1, pp. 100-104):

- an entity is represented by a box
- a relationship is represented by a straight line that connects two entities
- the optional (mandatory) participation: is represented by a transparent (plain dark) circle
- '*: m*' degree of a '*one-to-many*' (or '*many-to-many*') relationship is represented by a 'crow's foot'.

2.2.1 Relationships Elimination through Minimal Covering of FD-graphs

This section examines the current Functional Dependency graph (FD-graph) methodology for eliminating duplicate relationships, via the elimination of functional dependencies graph amongst attributes that show Armstrong's (1986) dependencies:

- reflexivity: If $X \supseteq Y$, then $X \rightarrow Y$,
- transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$, or
- or union : If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

The duplication removal algorithm exploits the graph theory concepts and algorithms. This was introduced by Ausiello, d'Atri and Sacca (1983) to reduce functional dependencies on ERD attributes. The set of attributes existing in the ERD corresponds to the graph vertices, the ERD relationships correspond to edges between two vertices (i.e. attributes). Figure 4 shows an example of an FD-graph. Single vertices such as A, F, B contain a single attribute. Compound vertices such as FBD and BD characterise composite attributes. The edge between each vertex represents a relationship. The full edges denote dependencies between two single vertices. The dotted arcs represent relationships between a compound vertex back to its corresponding single vertex.

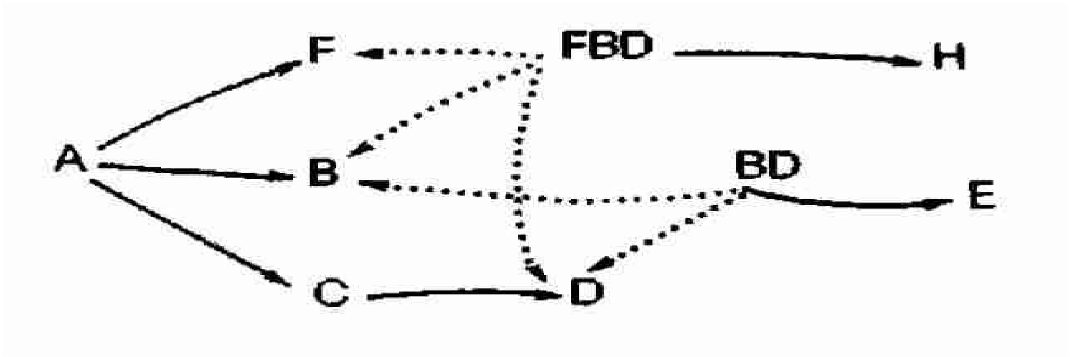


Figure 4 – Example of an FD-graph (Ausiello, d'Atri and Sacca, 1983)

The part of interest is the "non redundant coverings" of an FD-graph. Ausiello, d'Atri and Sacca (1983) suggest that a compound vertex is redundant if for each full edge between the vertices 'i' and 'j', there is a dotted path that starts from vertex 'i' and ends at vertex 'j'. Figure 5 shows a redundant and the equivalent "non redundant" coverings of an FD-graph.

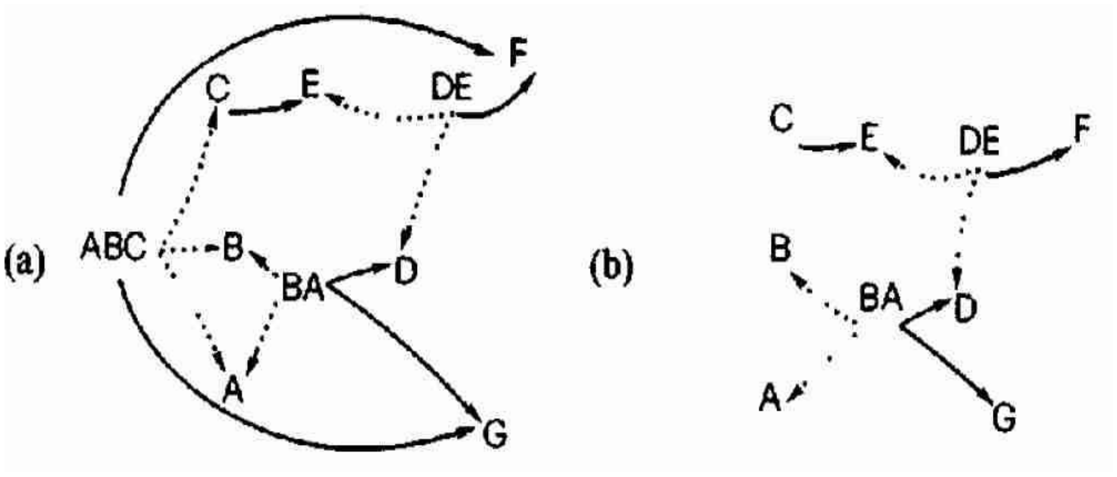


Figure 5 – Example of a redundant (a) and a "nonredundant" (b) FD-graph (Ausiello, d'Atri and Sacca, 1983)

The “nonredundant” covering is obtained by eliminating the redundant vertex ABC. This article also aims at finding the minimum covering of an FD-graph. Minimum covering is defined by covering the graph with the smaller number of vertices. A superfluous vertex, in a dotted path between two vertices, is a vertex that is equivalent to another one in this path. Therefore, the vertex will be eliminated to ensure the minimum covering of the path. However, the elimination of superfluous vertices requires moving their full outgoing vertices to the final destination. This means that there is a potential for creating redundant edges. This is shown in Figure 6. The deletion of the superfluous vertex CD, in Figure 6a), has the effect of eliminating the edges emanating from CD. However, it produces another edge between the edges AB and G, Figure 6b).

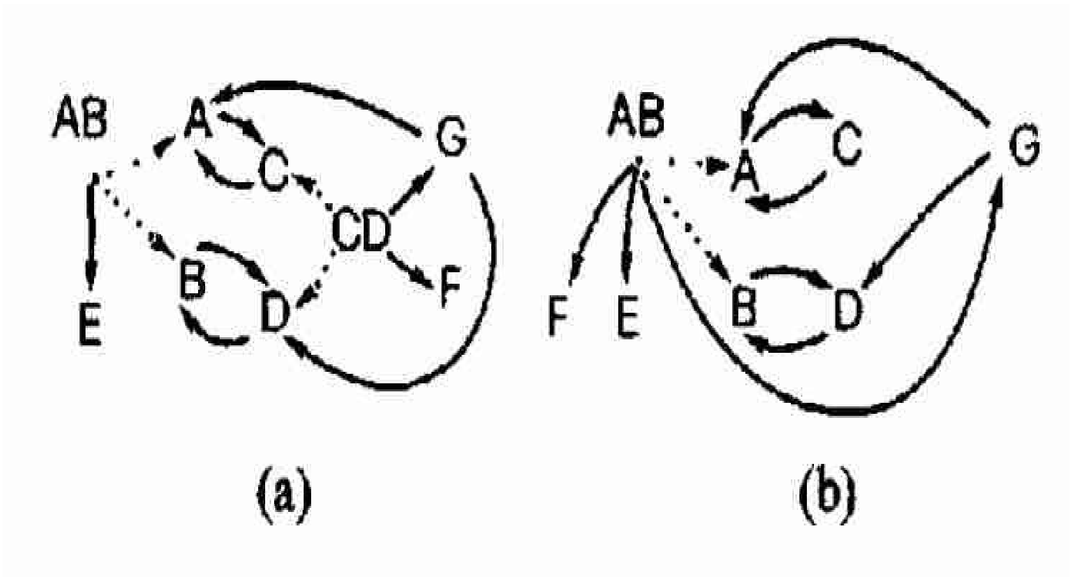


Figure 6 – Minimum covering of a “nonredundant” FD-graph ***(Ausiello, d’Atri and Sacca, 1983)***

The LR-minimum covering: resolves this problem by ensuring the minimum functional dependencies between two sets of attributes that share an FD. In other words, redundant full and dotted edges must be removed. The suggested algorithm is an extension of the transitive reduction of a directed graph. The steps are as follows:

- Determine the transitive closure of the graph. The transitive closure corresponds to the set of vertices that can be reached following each edge from each vertex in a graph (Sedgewick, 2004).

- Determine the strong connected components of the graph and choose a vertex as representative for every component. A component is strongly connected when there is a path between the vertex ‘a’ and ‘b’, and another from ‘b’ to ‘a’ (Sedgewick, 2004).

- Modify the graph by replacing every edge between vertices of different components by an edge between the representatives of such components, and by providing a Hamilton circuit among the vertices inside every component. A Hamilton circuit is a path from a vertex back to itself, i.e. a cycle (Sedgewick, 2004).

- Eliminate the redundant edges in the above mentioned graph following this mechanism:

    - Remove a redundant edge (i, j) in the minimum graph where there is a vertex k in a different strong component which forms two path (i, k) in the first strong component and (k, j) the second strong component, or when (i, k) and (k, j) belong to the same strong component.

- Eliminate dotted (full) edge (i, j) when there exists a dotted (full) FD-path starting from ‘i’ and ending at ‘j’, which does not contain the edge (i, j).

The resource above presents the first version of the algorithm. Since its publication, enhancements have been published in the literature, for example Yang, Li and Ann-Beng Ng (1988) and Wang (1996). This concept of removing redundant edges could be applied to ERDs that show entity reflexivity, transitivity or union. However it suffers from a number of drawbacks. First, the algorithm is very complex, and although it has been studied by many researchers, there does not seem to be a consensus on an all-encompassing algorithm that could be used to resolve the problem studied in this thesis. Second, it takes decisions on the deletion of attributes and relationships based on the minimum path to cover. Therefore, it is inclined to remove longer paths. This could lead to the incorrect deletion of some relationships/attributes. This approach could be helpful, if the user were able to review the list of relationships to delete for each minimum covering path iteration. However, this would require a large number of user interactions.

2.2.2 Relationships Elimination through the Real-World Knowledge

The following sections examine the current ‘real-world’ (i.e. domain-specific) knowledge methods. Storey (1993) introduces a dictionary that maps ‘real-world’ situations (e.g. classes of things) to a semantic relationship (such as ‘is-a’ relationship), and shows the database design implications (e.g. the inheritance from a super type to a subtype). The article defines a list of semantic relationship types (such as inclusion, possession, part-whole, etc.). The description of each individual semantic relationship type is not relevant to this thesis. The prototype, named *Semantic Relationship*

Analyzer, implements the rules described above. The system obtains 'real-world' knowledge: ' from the user in the form of a short sentence made of a subject (entity 1), a verb (the relationship) and the object (entity 2). For example, 'Companies have Employees'. Words such as 'have' are considered vague. Therefore, the system elicits the relationship refinement to an association or aggregation, etc. It also enquires for the minimum/maximum cardinalities at both ends of the relationship.

The key ideas lie in the rule based decision making process for potential redundant relationships:

- 'Is-a' or 'part of' relationships. When there is a three relationships cycle where 'Entity1 is-a Entity2', 'Entity2 part-of Entity3' and 'Entity3 part-of Entity1'. In that case, the relationship 'Entity3 part-of Entity1' is considered as potentially redundant.
- Synonym/antonym relationship name. When there are many relationships between two entities; for example, the relations 'teaches' and 'taught-by', in 'Professor teaches Course' and 'Course taught-by Professor'. The synonym relationship is suggested for removal. The same applies to antonyms.

Lloyd-Williams (1997) utilises the principles explained in the previous point and enhances the dictionary rule base decision engine with the representation of the domain knowledge. This is called the thesaurus approach. It allows the user to custom create a web of concepts that are linked with one another (c.f. Figure 7, p. 22)

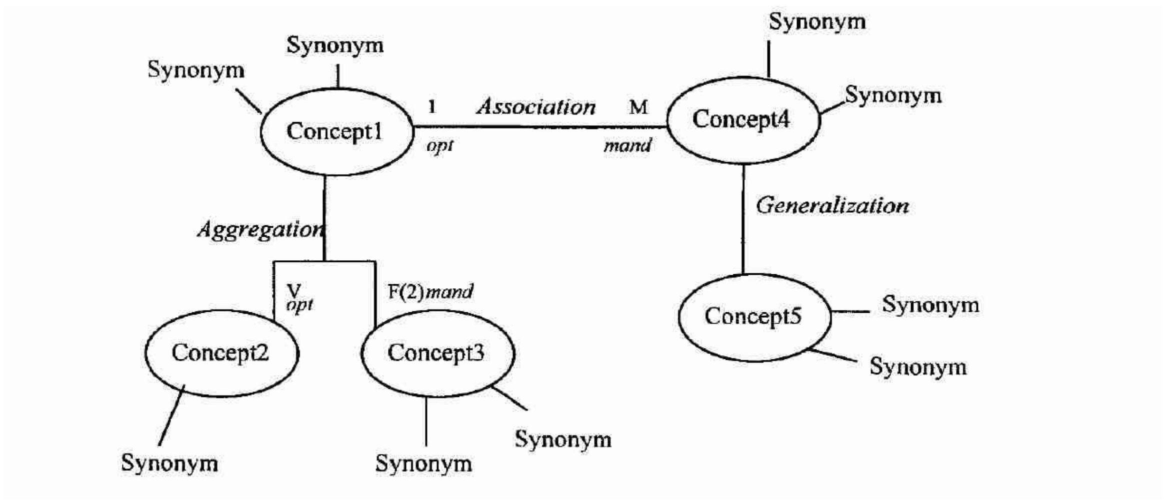


Figure 7 - General method representing domain knowledge (Lloyd-Williams, 1997).

The main advantages of these articles are:

- the study of the capacity to extend the domain knowledge, and consequently reduce the number of user interactions, as well as
- the analysis of domain specific knowledge approach limitations

Azman and Noah (2000) introduce an automated conceptual database design tool, named *Intelligent Object Analyser (IOA)*. This article provides a summary of the theoretical concepts defined in the previous materials, and reveals the ‘real-world’ knowledge usage limitations.

The system relies on a database of rules and facts. Rules encapsulate the knowledge on how to detect and resolve four types of errors in the model:

- Semantic inconsistencies: inconsistencies generated by missing relationships or the existence of a relationship transitivity.
- Inconsistent concepts: inconsistencies due to missing properties.

- Redundant inherited properties and relationships: redundancy occurring within a generalisation hierarchy where the parent class and its child classes include the same properties or participate in the same relationship.

- Redundant elements: inconsistencies produced by synonymous concepts or relationships.

Facts represent the user domain view as well as the generated conceptual view after detecting/resolving the above listed errors. The key idea in this article is the demonstration that the usage of synonyms can lead to inconsistency not being discovered. For example, “patient undergoes operation” and “patient underwent operation”. The ‘undergoes’ and ‘underwent’ relationships are redundant. Consequently, the dictionary had to be increased to allow for comparison of verbs in the past tense. This article complements the previous literature and illustrates an extra limitation on the ‘knowledge-based’ approach.

Noah and Williams (2000) tested the *IOA,* introduced by Azman and Noah (2000), against a number of fabricated cases. The aim of this analysis was to quantitatively evaluate the ability of these methods. The following criteria are analysed:

- the ability to detect model errors

- the number of errors generated by the system

- the processing time for the *IOA* to complete its analysis

- the number of interaction required by the user

The result of the above test establishes that, although it is theoretically possible to use such a methodology to find the complete set of potential redundant relationships in an ERD, it is rarely used in practice.

The review of the current literature on 'real-world' knowledge draws the same conclusions as the result above. Although these methods can help detect inconsistencies, the effectiveness of these methodologies depends "greatly on the accuracy and completeness of the system-help 'real-world' knowledge, and the results obtained from the tests may be influenced to certain extent by the variety and coverage of the generated test-cases" Noah and Williams (2000).

2.2.3 Relationship Elimination through the Analysis Cardinality and Participation Constraints

The approach taken by Dullea and Song (1997) uses occurrence diagrams and matrix algebraic techniques to generate a set of heuristics that qualifies the redundancy state of single binary relationships in a cyclic path. Dullea and Song (1997) characterise the existence of a redundant relationship as follows: "A single binary relationship occurring in a cyclic path is defined as a redundant relationship if there is a composite relationship that completes the cyclic path and represents the same concept through a semantic connection constraint that is semantically related, structurally unambiguous, and sufficiently complete." Figure 8 shows an 'n-relationship' path with a redundant relationship (R3) on the path X.

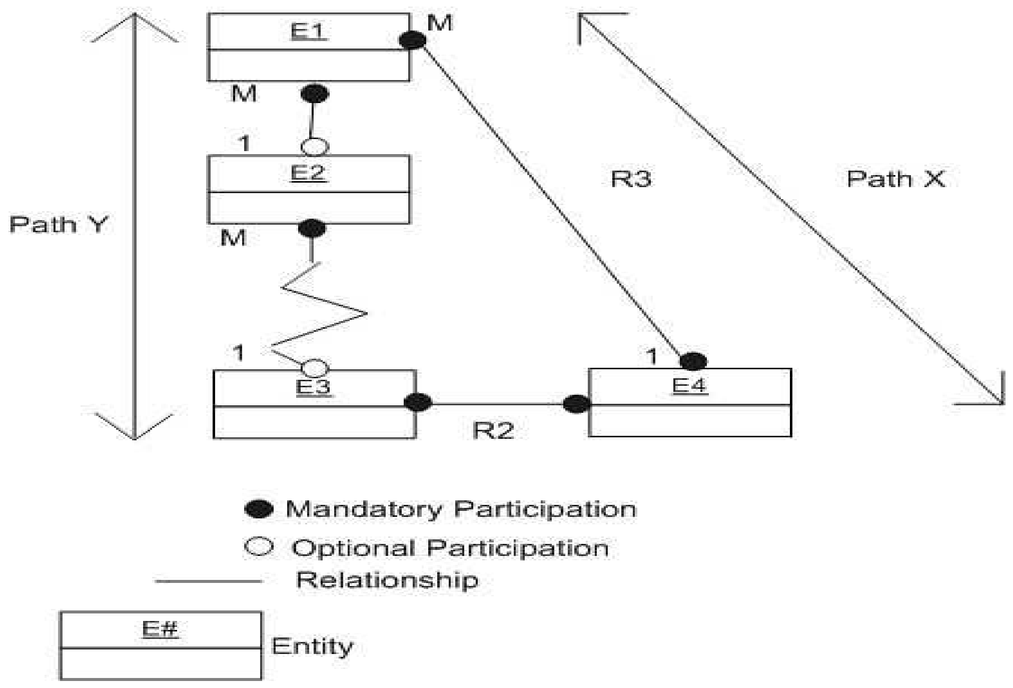


Figure 8 – Example of a redundant relationship (R3) in a single binary cyclic 'n-relationship' path (Dullea and Song, 1997)

The part of interest, for this thesis, is the elaboration of rule based 'unambiguous' redundant relationships, derived from the analysis of the cardinality and participation constraints. The tables below summarise these findings. The entity and relationships correspond to the ones listed in the Table 1 and Table 2.

R3	R1	R2
1-1	1-1	1-1
1-1	1-1	1-M
1-1	1-M	1-1
M-1	1-1	M-1
M-1	M-1	1-1
M-1	M-1	M-1
1-M	1-1	1-M
1-M	1-M	1-1
1-M	1-M	1-M

Table 1 – List 'unambiguous' redundant relationships (in column RE1E4) based on the cardinality constraints of a cyclic four entities relationship diagram. 1 means the cardinality is one. M means the cardinality is many (Dullea and Song, 1997).

R3	R1	R2
M-1	M-1	M-1
● ●	● ●	● ●
● ●	● ○	● ●
● ●	● ○	○ ●
● ○	● ○	● ●
● ○	● ●	● ○
● ○	● ○	● ○
● ○	● ○	○ ●
● ○	● ○	○ ○
○ ●	○ ●	● ●
○ ●	● ●	○ ●
○ ●	● ○	○ ●
○ ●	○ ●	○ ●
○ ●	○ ○	○ ●
○ ○	○ ○	● ●
○ ○	○ ●	● ○
○ ○	○ ○	● ○
○ ○	● ○	○ ●
○ ○	● ●	○ ○
○ ○	● ○	○ ○
○ ○	○ ●	○ ○
○ ○	○ ○	○ ○

Table 2 – List 'unambiguous' redundant relationships the special of 'M-1, M-1, M-1' relationships between the entities R1, R2 and R3 and their individual participation. A plain circle (●) means the relationship is mandatory. (○) means the participation is optional (Dullea and Song, 1997).

Table 1 lists the 'R3' 'unambiguous' redundant relationship based on cardinality constraints of the 'R1/R2' and 'R3' relationships. For example the first tree rows in the table reads; 'R3' is redundant when there is a 'one-to-one' relationship, as long as:

- 'R1' is a 'one-to-one' relationship and 'R2' is 'one-to-one' or '*one-to-many*', or
- 'R1' is '*one-to-many*' relationship and 'R2' is a 'one-to-one' relationship.

Table 2 lists the participations that are required to establish whether 'R3' is redundant in the case where the 'R3, R1, R2' relationships are of type '*many-to-one*, *many-to-one*, *many-to-one*'. This follows the same logic as the above example, this time the participation constraints are compared for this special case.

This rule based decision mechanism is attractive as it provides a set of rules based on the cardinality/participation constraints for finding 'unambiguous' redundant relationships. This approach is radically different from the methodologies demonstrated so far. However, this approach has some drawbacks. First, it requires that the user creates "semantic connection constraints" to discover unambiguous redundant relationships. Second, the study is only limited to single relationships, on a cycle with three entities.

This work has not been followed by any other researchers. The only citations available: Goelman and Song (2004) and Dullea and Song (1998) were produced by one of the same authors revisiting the topic. Therefore, I could not apply triangulation to evaluate the resource conclusions. Furthermore, the citing literature only applies the raw results obtained by Dullea and Song (1997) without providing a critical review of the original work.

2.2.4 Relationship Elimination through the Analysis of Shared Attributes and Inclusion Constraints

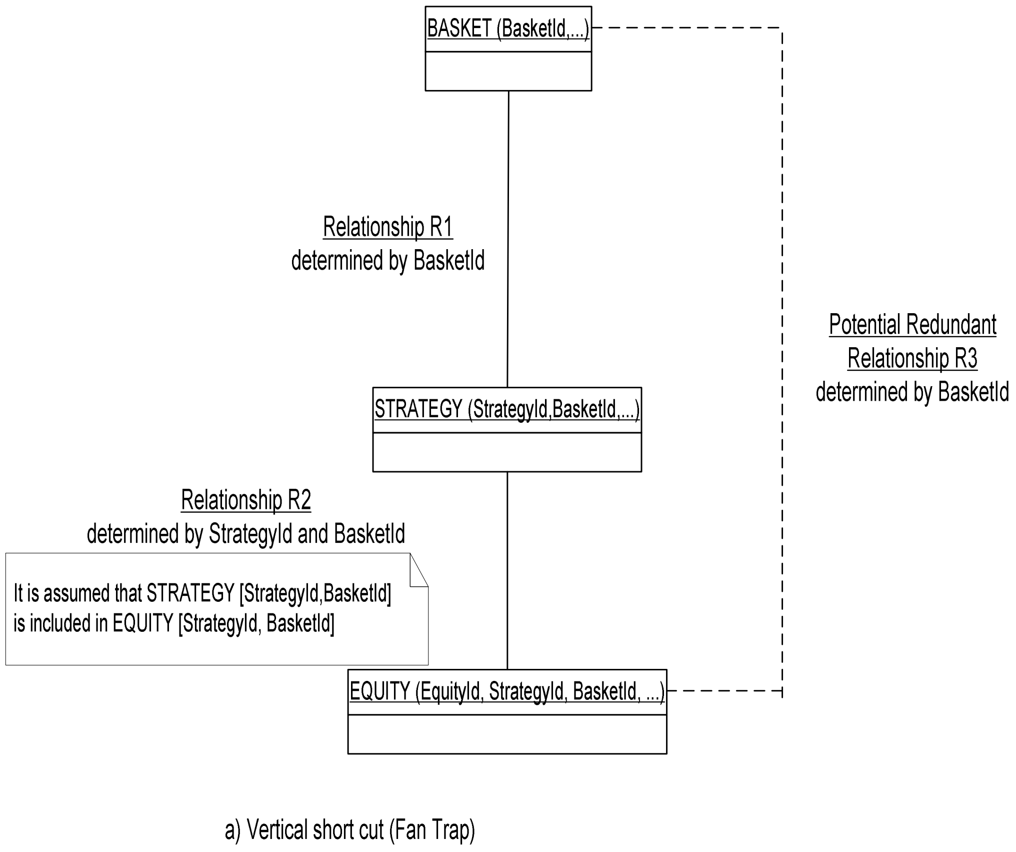
Rosenthal and Reiner (1994) present a database design tool, named the *Database Design and Evaluation Workbench (DDEW)*. This enables the on demand transformation of a data model into a conceptual, logical or hierarchical model. To achieve this, the tool uses an internal representation of the data model, and a set of algorithms to perform the transformation from one model type to the next. It also employs a user interface that allows the user to edit the data model. The article describes the embedded set of algorithms used by the data model engine. It also explains "heuristics rules" that attempt to improve the schema by adding or removing information. As this is not the main focus of this thesis, these points will not be analysed further.

The cornerstone of this article is the description of a methodology for identifying unnecessary relationships. One of the *DDEW* characteristics is the creation of relationships based solely on the entities and their attributes names. This may occur when the underlying model is represented as a set of unlinked entities. However, this generation of relationships may lead to the creation of meaningless or redundant relationships. Rosenthal and Reiner (1994) describe a redundant relationship as being a relationship that is both "semantically-unnecessary", i.e. it can be obtained by joining other relationships, and "content-unnecessary", i.e. it can be deleted without affecting the schema information.   The detection of potential redundant relationships based on i) the name of matching single or multiple attributes between three entities, and ii) the existence of inclusion constraints on one of the relationships in the cycle. The "Composition Theorem for Relationships that Share Common Attributes" can be summarised as follows. In a three entity cycle (E1, E2 and E3), linked by the relationships:

- R1 between E1 and E2
- R2 between E2 and E3
- R3 between E1 and E3.

When the attributes of the entity E1 matches the attributes of entity E2, and all tuples of the entity E1 are included in the entity E2, then the relationship R3 is a composition of the relationship R1 and R2.

Figure 9a) and Figure 9b) show an example of a vertical (i.e. cyclic path), and a horizontal shortcut.


BASKET (BasketId,...)
Relationship R1
determined by BasketId
Potential Redundant
Relationship R3
determined by BasketId
STRATEGY (StrategyId,BasketId,...)
Relationship R2
determined by StrategyId and BasketId
It is assumed that STRATEGY [StrategyId,BasketId]
is included in EQUITY [StrategyId, BasketId]
EQUITY (EquityId, StrategyId, BasketId, ...)
a) Vertical short cut (Fan Trap)

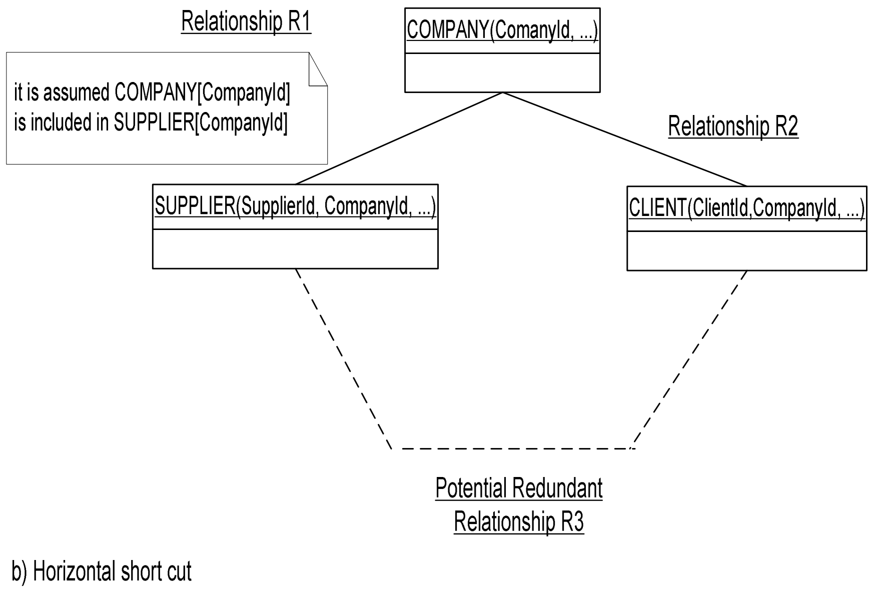

Relationship R1
COMPANY(ComanyId, ...)
it is assumed COMPANY[CompanyId]
is included in SUPPLIER[CompanyId]
Relationship R2
SUPPLIER(SupplierId, CompanyId, ...)
CLIENT(ClientId,CompanyId, ...)
Potential Redundant
Relationship R3
b) Horizontal short cut

Figure 9 - Discovery of potential redundant relationships in vertical and horizontal shortcuts.

This methodology is particularly relevant to the topic as it offers:

- Means of discovering potential redundant relationships based on the model attributes and inclusion constraints. In particular, this technique resolves the problem of the '*many:many:many'* cycle which complements the solution described by Dullea and Song (1997). The latter authors concluded that no rule could be produced for this particular case.

- An alternative to the analysis of the 'real-world' knowledge and the relationship cardinality/participationCardinality.

However, this technique, suffers from the same problems as the one proposed by Dullea and Song (1997). First, it requires extra data about the schema. In this case, there is a need for inclusion constraints. Second, the study is only limited to single relationships, on a cycle of three entities. Furthermore, it is closely related to a dictionary approach (c.f. section 2.2.2, p. 20), and therefore suffers from the same drawbacks. In this case, the semantic comparison is based on the attributes of the relationship, rather than the relationship name, as proposed by Storey (1993). This means that the foreign key must support the same attribute name as the parent entity primary or composite key, otherwise the discovery may fail. Consequently, this approach cannot be used in the case of multiple relationships. In this instance, the name of a posted foreign Key attribute must be unique, which breaks the model.

Although there is a number of articles citing this reference (the full list can be found using the Scopus on line search engine), none of them critically assessed the above mentioned solution to eliminate superfluous relationships. Therefore, I could not apply triangulation to evaluate the resource conclusions.

2.2.5 Relationship Elimination - The Hybrid Approach

Bowers (2002) applies techniques used in graph search algorithms to compare relationship semantics, and discover potential redundant relationships. The two parts of interest are:

- the formal and practical description of the model for discovering potential redundant arcs
- the discussion of the model limitations

Bowers (2002) proposes a three step approach to discover potential redundant relationships:

a) a process for "composing transitive relationships"

b) "a strategy for comparing each relationship with every alternative (composed) path between the entities it connects"

c) "a way of comparing the semantics of alternative paths that have the same signature".

Bowers (2002) presents a general algorithm, using Boolean logic, which enables the composition of transitive relationships in an ERD. There are two path signatures per relationship, from entity E1 to entity E2, and *vice-versa*. Equivalent single or composition paths between two entities are paths that have the same signature and same semantics. A composition path takes into account the relationship participation (i.e. mandatory or optional) and whether it is a function (i.e. one) or multifunction (i.e. many).

The composite path ($R_{1,n}$) from entity E_1 to entity E_n is represented as:

(1) $R_{1,n} =[E_1], R_{1,2}, [E_2], R_{2,3}, \dots , R_{n-1,n}, [E_n]$

The participation ($P_{1,n}$) of a composite path from entity E1 to entity En is characterised as:

(2) $P_{1,n} = P_{1,2} \wedge P_{2,3} \wedge \dots \wedge P_{n-1,n}$

The mandatory and optional participation are represented respectively by the Boolean values '1' and '0'.

The cardinality ($C_{1,n}$) of a composite path from entity E1 to entity En is defined as:

(3) $C_{1,n} = C_{1,2} \wedge C_{2,3} \wedge \dots \wedge C_{n-1,n}$

The one and many cardinality constraints: are represented respectively by the Boolean values '1' and '0'.

The signature of a binary relationship between two entities i and j is represented as

(4) $S_{i,j} = (C_{j,i} , P_{i,j} , P_{j,i} , C_{i,j})$

The complete signature ($S_{1,n}$) of the composition relationship path is characterised as:

(5) $S_{1,n} = S_{1,2} \wedge S_{2,3} \wedge \dots \wedge S_{n-1,n}$

The detection strategy follows this process. The initial ERD is represented by an adjacency-matrix. The x- and y-axes list the entities in the model. The element of the matrix contains the signature of each relationship. The signatures for path of length ‘n’ are obtained by pre-multiplying the adjacency-matrix of order ‘n-1’ by the initial matrix. At each stage of the process, the path signatures of order ‘n’ are compared to the initial matrix. Any equivalent composed paths are considered as potentially redundant.

However this strategy is based on two strong assumptions:

a) loops are not considered (i.e. recursive relationships)

b) any path that includes an entity more than once is discarded (i.e. cycles).

It also assumes that the ERD is likely to be sparse (i.e. a graph with only a few edges), and that most paths will be of order two or less. Bowers (2002) indicates that the algorithm can be improved in two ways:

a) the strategy performance can be improved by removing the redundant paths at the earliest opportunity

b) the complexity can be reduced by focusing only on relationships with a degree less or equal to ‘*one-to-many*’.

However, there are still a number of limitations:

- due to the nature of the search algorithm, the number of interactions with the user may be high, even if there is no redundant arc
- the elimination of redundant arcs, during each iteration, may delete longer cycles before they are even detected

- when the strategy assumptions (no loop/cycle and unique relationship) are relaxed, the transitive closure for the algorithm becomes more complex.

This methodology is central to this thesis as it enables the discovery of potential redundant relationships in an algorithmic fashion, which has the following advantages:

- it is relatively easy to implement
- it seems to be the most complete solution in terms of use case coverage.

Furthermore, the relaxation of the model assumptions should alleviate some of the model limitations. I could only find one resource citing this reference. Queralt and Teniente (2006) acknowledge that the approach proposed by Bowers (2002) could be used to verify the "schema satisfiability" of the UML class diagram. However, Queralt and Teniente (2006) discarded this solution because it did not offer a generic answer, because of the above mentioned limitations (c.f. section 2.2.5 p. 31).

2.3 The Research Question

The research hypothesis of this investigation is that the enhancement of the solution offered by Bowers (2002) should have a higher rate of success in discovering potential redundant relationship for complex ERD, in comparison to the methodologies mentioned in section 2.2. To achieve this, the output generated by the prototype is compared against the visual inspection of each test case. I measure success by counting the number of correct potential superfluous relationships that are discovered. I measure failure by counting the number of missed or incorrect potential superfluous relationships generated by the prototype (c.f. section 4.3.2 p. 75, and 5.2 p.81).
The processing time to complete the analysis should be 'relatively' efficient, especially for sparse schema. For this, I compare the proposed algorithm growth rate versus

empirical result obtained by other researchers in the field of graph algorithm (c.f. sections 4.3.1 p. 75, and 5.1 p.78).

2.4 Summary

The literature provides numerous methodologies to search for duplicate relationships in an ERD. Figure 10 uses a Venn diagram to summarise the ability of the different methodologies to analyse different types of relationships. I generated this visual representation from the conclusions provided in the literature review. The Venn diagram only offers a "big picture" of the capability of each approach, without describing the limitations of each model.

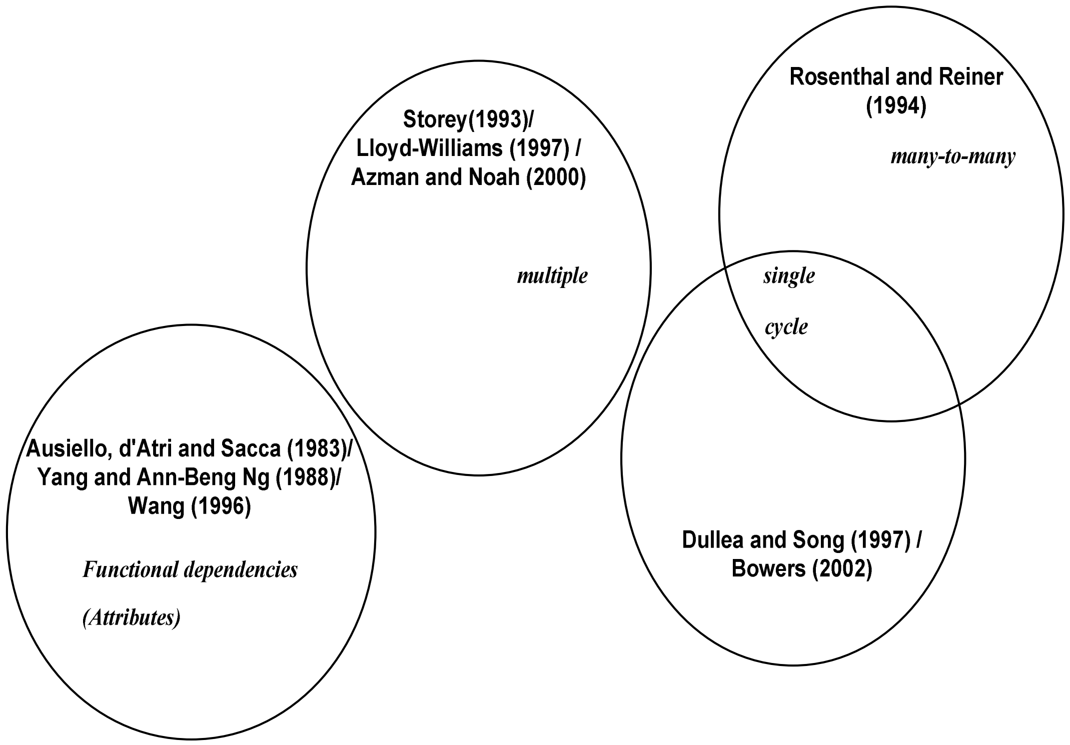


Figure 10 – Venn diagram that displays the current ability of the different approaches to analyse different types of relationships to search for potential redundant relationship.

The FD-graph uses minimum covering to find the shortest path within a graph. Although this approach is rigorous, it cannot be applied in its current form to solve the current problem, as it systematically removes longest paths. This is not necessarily the correct decision to make. At the other side of the spectrum, the ‘real-world’ knowledge approach proposes a method that analyses the relationship semantic against a dictionary. Relationship duplications are searched with the help of an algorithm that analyses the relationships name and eliminates synonymous relationships. The major drawback of this solution is the capacity of the dictionary to map synonymous concepts. Consequently, the success of the search can be inconsistent.

The analysis of cardinality constraints offers a middle ground solution. This empirical method concludes on the existence of ‘unambiguous’ redundant relationships for a defined set of relationships cardinality and participation. These results are only applicable to cycles of three entities linked by single relationships, when “semantic connection constraints” are defined. The limitations of this approach are coupled to the model assumptions. Users need to invest time defining “semantic connection constraints” for the algorithm to discover duplicate relationships. Instead, it could be argued that this time could be devoted to removing duplicate relationships. In the same vein, the “shared attributes and inclusion constraints” method suggests a means of discovering duplicate relationships, for cycles of three entities linked by single relationships, by analysing the attributes names and inclusion constraints. This solution offers the added value of analysing ‘*many:many:many*’ cycle, which is not supported in the previous method. However, as previously stated, it is limited to a three entity cycle (with single relationships). It also requires the inclusion constraint concept, and is implicitly founded on a dictionary.

The last approach, which was studied, bridges the gap between:

- the need for a complete (or almost complete), consistent and systematic approach to finding relationship duplications using a rigorous methodology (i.e. based on the graph theory principles)
- the analysis of the relationship signature, which consists of the cardinality and participation conditions

Although this approach seems to be the most appropriate to resolve the problem, its current form does not allow for the analysis of loops, cycles, multiple and '*many:many:many*' relationships. The remainder of this thesis aims at extending this solution to loops, cycles and multiple relationships. However, the search of potential redundant relationships in '*many:many:many'* relationships is out of scope.

Chapter 3 Research Methods

This chapter explains and justifies the choice of the research methods used to analyse the capacity of the proposed improved algorithm to discover potential redundant paths. The first section examines the literature review strategy. This step is compulsory to provide the necessary background knowledge of the problem domain. The second and third sections focus on the scope of the experiment, and the technical environment in which the experiment has been performed.

The fourth section describes in details the four steps of the evaluation methodology:

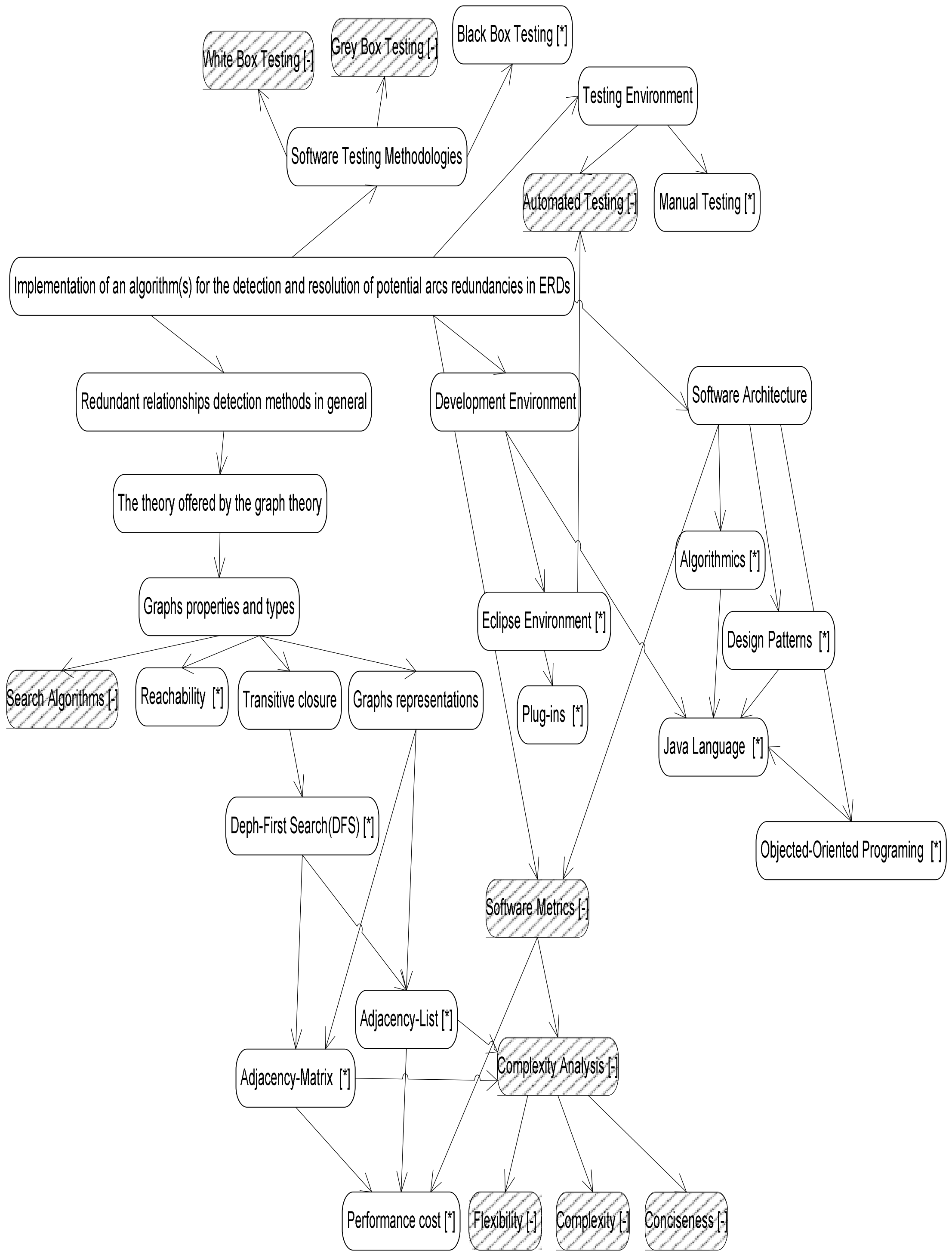
- the hypothesis to be tested
- the selected test cases
- the prototype architecture
- the logical structure of the proposed algorithm

3.1 Literature Review

The literature review aimed at investigating two areas; i) the Eclipse development environment and ii) the implementation of an adjacency-matrix algorithm. This research gave birth to the numerous concepts listed below. Figure 11 lists all concepts that were researched. The main key words/subjects of the literature research focus on:

- The graph theory - this provides the theoretical foundation for the adjacency-matrix implementation.

- The software architecture - this provides design patterns to build a flexible and extendible architecture.
- The development environment - this provides a rich environment to develop the prototype.
- The software testing methodologies - these offer a framework for maximising the software quality.
- The testing environment - this provides an extendible framework from which automated tests can be built, by implementing the software testing methodologies.



[*] Areas to form basis for research

[-]Area abandoned because deemed irrelevant

Figure 11- A relevance tree for the subject of 'Implementation of an algorithm for the detection and resolution of potential arcs redundancies in ERDs

3.2 The Scope of the Experiment

The second chapter described numerous approaches that were currently available to discover superfluous relationships. However, none of them offers a ‘complete solution’. Consequently, the first objective is to prove that the enhanced version of the algorithm introduced by Bowers (2002) can discover more potential redundant relationships than the approaches listed in the literature review (c.f. section 2.2 p. 15). The experiment uses careful human inspection from a list of fabricated test cases (c.f. Appendix A, p. 105). The test cases include loops, cycles and multiple relationships.

The second objective is to measure the processing time for the algorithm to complete its analysis. For this, another set of test cases has been used. Entities and relationships have been gradually added to a base case and execution timings have been recorded for each run. This has helped establishing to what extent the architecture is scalable.

I have abandoned a number of topics from this dissertation in order to focus on essential areas. The list of topics and reasons for their exclusion are listed below:

- The implementation of the approaches proposed by Rosenthal and Reiner (1994) and Dullea and Song (1997) were abandoned because I established that these two approaches did not yield satisfactory results in terms of discovery abilities. Therefore, I considered that implementing and comparing them against the algorithm proposed in this thesis would produce only little value.

- I discarded the review of the proposed algorithm code complexity because the complexity analysis benefits, in terms of the code flexibility and maintainability, are limited (and sometimes controversial).

- I did not pursue the generation of the SQL pseudo code from the final ERD because this has already been researched in details in the literature. Furthermore, this was not a focus point for the thesis topic. Therefore, this offered only a limited scope for this research and was abandoned.

3.3 The Experiment Environment

Figure 12 gives a summary of the hardware specification on which experiments were run. The code was written in Java, within the Eclipse SDK Eclipse platform 3.3 M 4 (JRE 1.5) environment. In order to minimise "white noise" due to external processes that may interfere unpredictably on execution timing test results, I shut down any unnecessary services (such as anti-virus, firewall, windows automatic updates, etc.). For the same reason, the PC was separated from the network.

ITEM	***VALUE***
OS Name	Microsoft Windows XP Professional
Version	5.1.2600 Service Pack 2 Build 2600
OS Manufacturer	Microsoft Corporation
System Name	D4Z6N82J
System Model	MXG061
System Type	X86-based PC
Processor 1	x86 Family 6 Model 14 Stepping 8 GenuineIntel ~1997 Mhz
Processor 2	x86 Family 6 Model 14 Stepping 8 GenuineIntel ~1997 Mhz
SMBIOS Version	2.4
Total Physical Memory	2,048.00 MB
Available Physical Memory	1.14 GB
Total Virtual Memory	2.00 GB
Available Virtual Memory	1.96 GB
Page File Space	3.85 GB

Figure 12 – PC Specification

3.4 The Experimental Framework

The experiment consists of creating a number of test schemas, and executes the proposed algorithm against each of them. The output offers quantitative information that describes the capacity of the algorithm to find potential redundant paths (c.f. Figure 13, p. 44). It also provides evidence about the algorithm scalability. The critical analysis of this data has helped formulating conclusions and recommendations about the thesis research question and hypothesis.

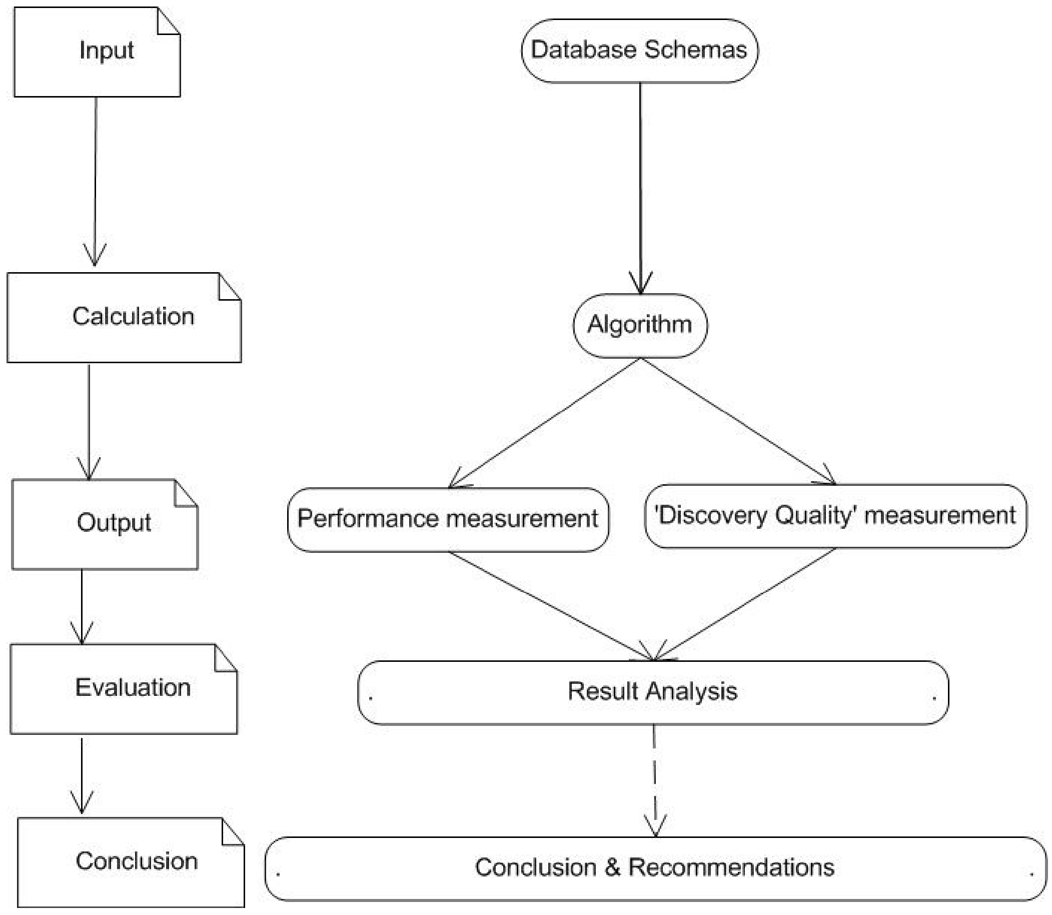


Figure 13– The experiment framework

3.5 Controlled Experiments

Due to the very large number of potential use cases and the graph complexity, it is necessary to control the graph types and the input variables. In other words, limit the number of entities, and the number of relationships between entities. Appendix A shows the use cases that were utilised to test the search algorithms ability to find correct superfluous relationships. Appendix D lists the test cases that were employed to test the scalability of the solution.

The quantitative data have been collected for the proposed algorithm as described in Table 3 below:

Algorithm Name	***Description***
EnhancedBowers	This is the implementation of the extended version of the algorithm proposed by Bowers (2002) that caters for loops, cycles and multiple relationships

Table 3 – Algorithms used for the test

The advantage of this method is the high level of control on the model input data. It reduces threats to validity of the inference chain between the hypothesis and the conclusion. On the ethical stand point, it helps complying with the 'Duty to Relevant Authority' section of The British Computer Society (2006), by minimising the potential for data misinterpretations.

3.5.1 ‘Discovery Quality’ Methodology

The objective is to establish in what cases the proposed algorithm delivers and where it fails. For this, the following measures have been selected:

- evaluate the number of correct potential redundant relationships found in an ERD, named ‘correct identification’,
- determine the number of incorrect potential redundant relationships produced by the algorithm, named ‘incorrect identification’,
- quantify the number of missed potential redundant relationships in the ERD, named ‘missed identification’

I have considered using statistical inference ‘right-tail’ tests to assess the hypothesis that the proposed algorithm can discover more than 80% of potential redundant paths, and more than 80% of the equivalent composed path, for a 5% level of significance. However, the collected data are fabricated (i.e. not random) and it is only limited to approximately 20 test cases, which is below the minimum required size of 30 random samples. These two constraints do not allow for the creation of a normal distribution (Salvatore and Reagle, 2002). Consequently, the analysis was focused on a qualitative (rather than quantitative) examination of data. However, I would recommend performing a statistical analysis against the proposed architecture when it is fed with a large sample of real life ERDs.

3.5.2 Scalability Measurement Methodology

The aim of this experiment is to record the execution time (in milliseconds) for each test case schema listed in Appendix E. As a result, it is be possible to deduce the growth rate of each algorithm, using the “Big-Oh” notation. It also helps discovering whether the solution is scalable with ERD schemas that are denser (in terms of entities and relationships) than the one studied in section 3.5.1.

The initial selected ERD is a bi-connected component ERD containing five entities (c.f. Figure 14, p. 47). This structure was chosen as this is a relatively complex graph that contains multiple cycles. It would have been interesting to study the execution time on ‘real life’ ERDs, to generate a precise view of the *enhancedBowers* scalability. However, after some investigation, I realised that gathering ‘real life’ ERDs and transforming them into schemas that can be read by the prototype was a time consuming exercise. Therefore, I decided to focus on a controlled experiment.

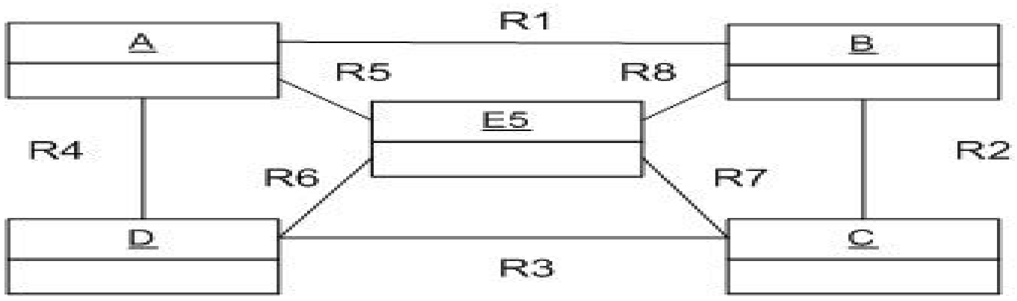


Figure 14– The experiment framework

The first test case suite is represented by a sparse ERD where the number of relationships is doubled (no loops), everything else being equal. The second test case is similar to the first test case suite. This time the new relationships are loops only, and added to all entities (bar the central one). The final test case suite doubles the number of entities, knowing that each entity has two relationships. It starts from a five entities/eight relationships diagram and grows to an 80 entities/160 relationships diagram.

3.5.3 The Experiment Limitations

There are two main limiting factors to this experiment. First, the collected data are derived from the fabrication of specific test cases, and a restricted number of variable combinations. Second, the execution time figures are tied to the platform used for this experiment (i.e. Microsoft Windows). It is possible that the generated figures would have been different if I had used a Mac OS or a UNIX operating systems. Therefore, conclusions can only apply to the specific cases and platform used for the purpose of this thesis.

3.6 Experiment Implementation

This section aims at prototyping the proposed algorithm. The prototype has been developed on the Eclipse platform (V3.3M4/Java SDK 1.5), as this is a rich 'Open Source' development platform. It allows for easy extension of third party components (plug-ins). It also provides an 'Integrated Development Environment' (IDE) with tools to manage workspaces, build, launch, debug and test applications (Holzner, 2004; Arthorne and Laffra, 2004). Eclipse supports a multitude of languages. I have selected Java, as this is an Object-Oriented language, and it is the closest fit to my language knowledge and experience. Due to time constraints, it was not possible to fully test the prototype. The framework was only tested partially.

3.6.1 The Prototype Objectives

The aims of the prototype are threefold:

- demonstrate that the proposed algorithm can be implemented
- collect qualitative data to validate the ‘quality of the discovery’
- collect quantitative data about algorithm execution time

3.6.2 High Level Functionality

The prototype, named Graph Edge Validator 2008 (GEV2008), provides the following functionalities:

- The user selects one of the ERD use cases and runs the application to discover potential redundant relationships, for a given adjacency-matrix order.
- When a potential redundant relationship is found, the system prompts the user.
- The user makes a selection on the relevance of the relationship.
- The system amends the ERD to reflect the choice made by the user.

The below class, sequence and component diagrams describe an open framework that combines specialised and abstract components. The current implementation focuses on the *EnhancedBowers* algorithm. However, other algorithms could be implemented; such as the ones suggested by Rosenthal and Reiner (1994) or Dullea and Song (1997). The below diagrams follow the UML standards defined by Booch, Rumbaugh and Jacobson (2005).

3.6.3 High Level Component Structure

A high-level component structure of the program is described in Figure 15.

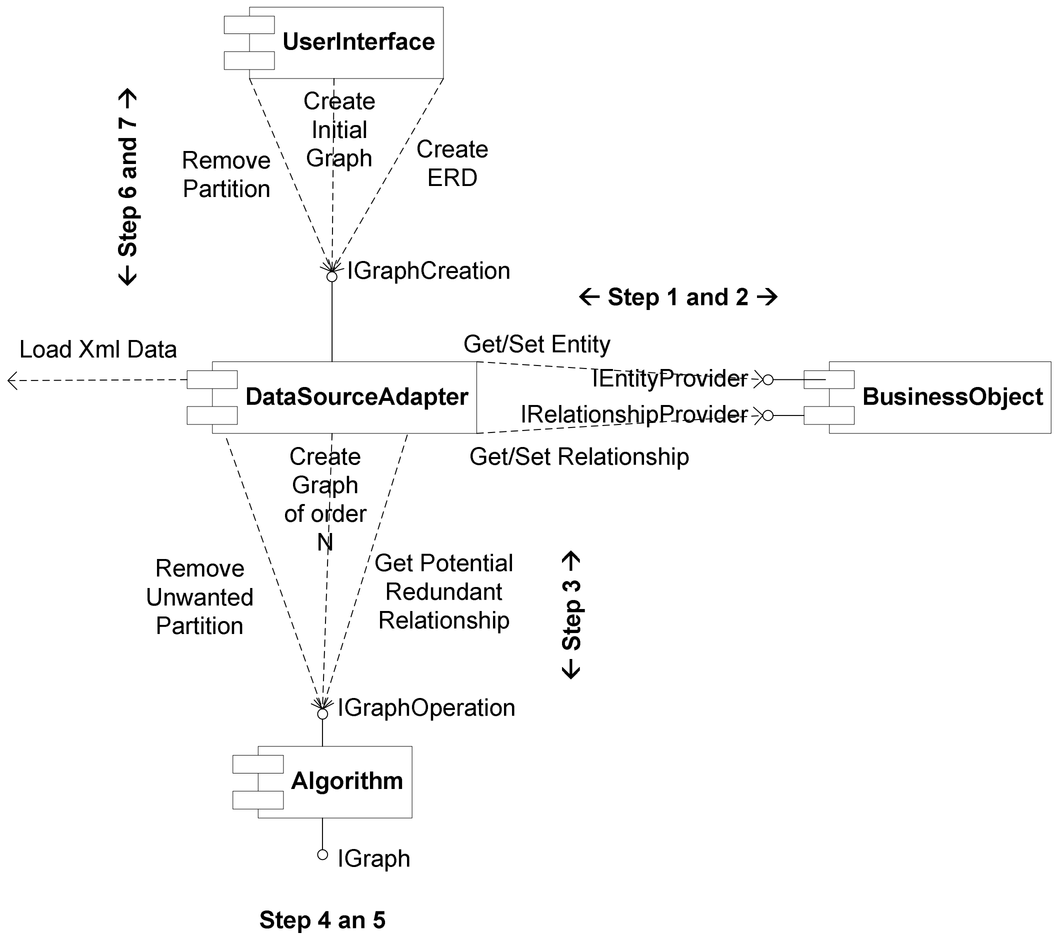


Figure 15– High level interaction between the framework major components.

Step 1 - the ‘DataSourceAdapter’ loads the ERD representation stored in a XML file. Each XML file contains one of the use cases defined in the previous section (c.f. Appendix B, p. 118 for the XML description)

Step 2 - the ‘DataSourceAdapter’ adapts the XML into an ERD business object by calling the ‘BusinessObject’ component (c.f. Appendix G, p. 141 for the ‘BusinessObject’ class diagram description)

Step 3 - the ‘DataSourceAdapter’ calls the relevant algorithm (e.g. *EnhancedBowers*) by delegating the request to the ‘Algorithm’ component

Step 4 - the ‘Algorithm’ component generates the relevant IGraph based on the selected algorithm and order (e.g. order 2)

Step 5 - the ‘Algorithm’ component computes the list of potential redundant relationships and their equivalent composed paths

Step 6 - the ‘DataSourceAdapter’ sends the results back to the ‘UserInterface’ component

Step 7 - the user’s modifications to the ERD object are recorded into the business object

Step 8 - steps 1) to 7) are repeated till the ERD analysis is complete

The focus of this thesis is not the design of the infrastructure. Therefore only a high level description of the logical framework is provided. Readers are invited to examine the code, provided with the CD-ROM, for more detailed information. Appendices H/I describe respectively i) how to install the software, ii) how to use the user interface and iii) how to run the automated test cases.

3.6.4 The Algorithm Architecture

The algorithm logic and implementation is the cornerstone of the prototype. The first section describes the class diagram and the main interfaces supported by each algorithm. The methods name relates to their behaviour. The methods description is available as part of the code provided on the CD-ROM. The second section describes the algorithms program structure.

3.6.4.1 Algorithm Class Hierarchy

The figures below describe the algorithm class hierarchy of the main objects used for the discovery of potential redundant relationships. The class diagrams follow the UML notation as illustrated in M878 (Unit 3, pp. 17-48). Figure 16 illustrates that the proposed algorithm derives from the `GraphOperation` base class, which implements the `IGraphOperation` interface. The `GraphOperation` object is responsible for the creation of an ERD XML schema matrix representation (c.f. section 2.2.5, p. 31). The `GraphOperation` object combines the `BooleanGraph` and `ElementGraph`, both implementing the `IGraph` interface to fabricate the final adjacency-matrix representation of type `ElementGraph`. Each specialised version of the algorithm (e.g. *EnhancedBowers*) overrides a list of methods, such as the `addEgdeToGraph(…), createElement(…), setPartition(…),` etc. methods. These methods fabricate the partitions (i.e. equivalent composed paths) contained in the matrix elements for a given algorithm. Based on this data, the selected algorithm can make a decision on whether to label (or not) a relationship as potentially redundant.

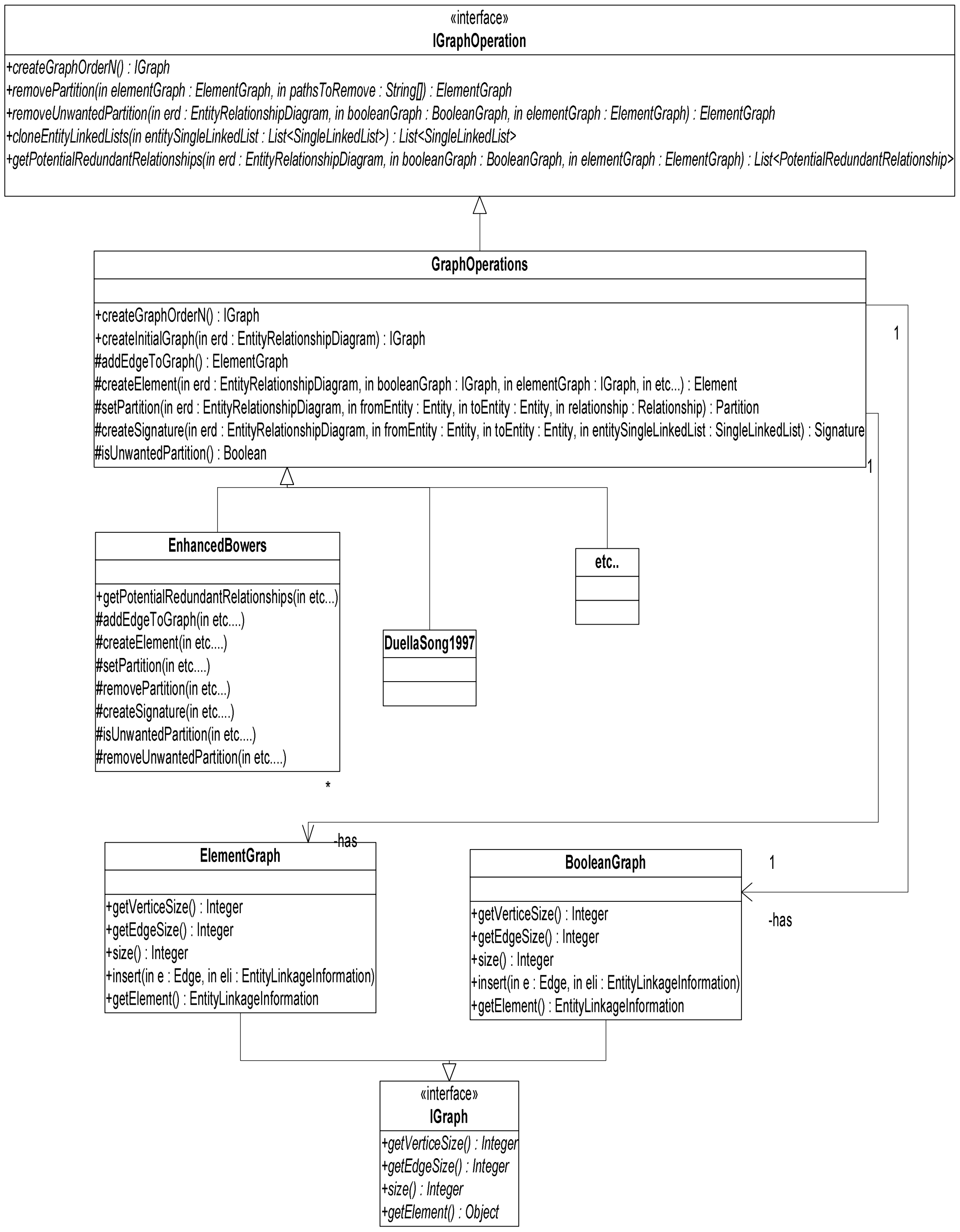

«interface»
IGraphOperation
+createGraphOrderN() : IGraph
+removePartition(in elementGraph : ElementGraph, in pathsToRemove : String[]) : ElementGraph
+removeUnwantedPartition(in erd : EntityRelationshipDiagram, in booleanGraph : BooleanGraph, in elementGraph : ElementGraph) : ElementGraph
+cloneEntityLinkedLists(in entitySingleLinkedList : List<SingleLinkedList>) : List<SingleLinkedList>
+getPotentialRedundantRelationships(in erd : EntityRelationshipDiagram, in booleanGraph : BooleanGraph, in elementGraph : ElementGraph) : List<PotentialRedundantRelationship>
GraphOperations
+createGraphOrderN() : IGraph
+createInitialGraph(in erd : EntityRelationshipDiagram) : IGraph
#addEdgeToGraph() : ElementGraph
#createElement(in erd : EntityRelationshipDiagram, in booleanGraph : IGraph, in elementGraph : IGraph, in etc...) : Element
#setPartition(in erd : EntityRelationshipDiagram, in fromEntity : Entity, in toEntity : Entity, in relationship : Relationship) : Partition
#createSignature(in erd : EntityRelationshipDiagram, in fromEntity : Entity, in toEntity : Entity, in entitySingleLinkedList : SingleLinkedList) : Signature
#isUnwantedPartition() : Boolean
1
1
EnhancedBowers
+getPotentialRedundantRelationships(in etc...)
#addEdgeToGraph(in etc....)
#createElement(in etc....)
#setPartition(in etc....)
#removePartition(in etc...)
#createSignature(in etc....)
#isUnwantedPartition(in etc....)
#removeUnwantedPartition(in etc....)
DuellaSong1997
etc..
*
-has
ElementGraph
+getVerticeSize() : Integer
+getEdgeSize() : Integer
+size() : Integer
+insert(in e : Edge, in eli : EntityLinkageInformation)
+getElement() : EntityLinkageInformation
BooleanGraph
+getVerticeSize() : Integer
+getEdgeSize() : Integer
+size() : Integer
+insert(in e : Edge, in eli : EntityLinkageInformation)
+getElement() : EntityLinkageInformation
1
-has
«interface»
IGraph
+getVerticeSize() : Integer
+getEdgeSize() : Integer
+size() : Integer
+getElement() : Object

Figure 16– The algorithm class hierarchy

Figure 17 illustrates the business object for the ERD XML schema matrix representation. The two main objects are the `BooleanGraph` and the `ElementGraph`, which implement the `IGraph` interface. The first one represents a Boolean adjacency-matrix: (i.e. the initial matrix), which lists the entities on the matrix x- and y- axes. The matrix intersection is set to a scalar value (e.g. 1) representing the number of relationships between two entities. If there is no relationship between two entities, then it is set to “0”. The second object represents a polymorphic adjacency-matrix. In other words, the object type contained in the matrix element is dependent on the selected algorithm. In the case of the *EnhancedBowers* algorithm it is an adjacency-matrix made of elements and partitions. However, this behaviour could be overridden for another algorithm type. The `BooleanGraph` and the `ElementGraph` objects contain an internal `Matrix` object that encloses a collection of `Element` objects, which represent the matrix elements. A matrix `Element` object is the intersection between a row on the x-axis and a column on the y-axis. In turn, each Element object contains a collection of Partition objects. The latter includes two types of objects:

- an entity `SingleLinkedList` object, which represents the equivalent composed path, excluding the source and destination entities listed on the matrix x- and y-axes, and
- a `SignatureComposition` object, of which purpose is to compute the equivalent path signature starting from the matrix source entity (x-axis) and ending at the matrix destination entity (y-axis), going through each `Partition` object of a given `Element` object.

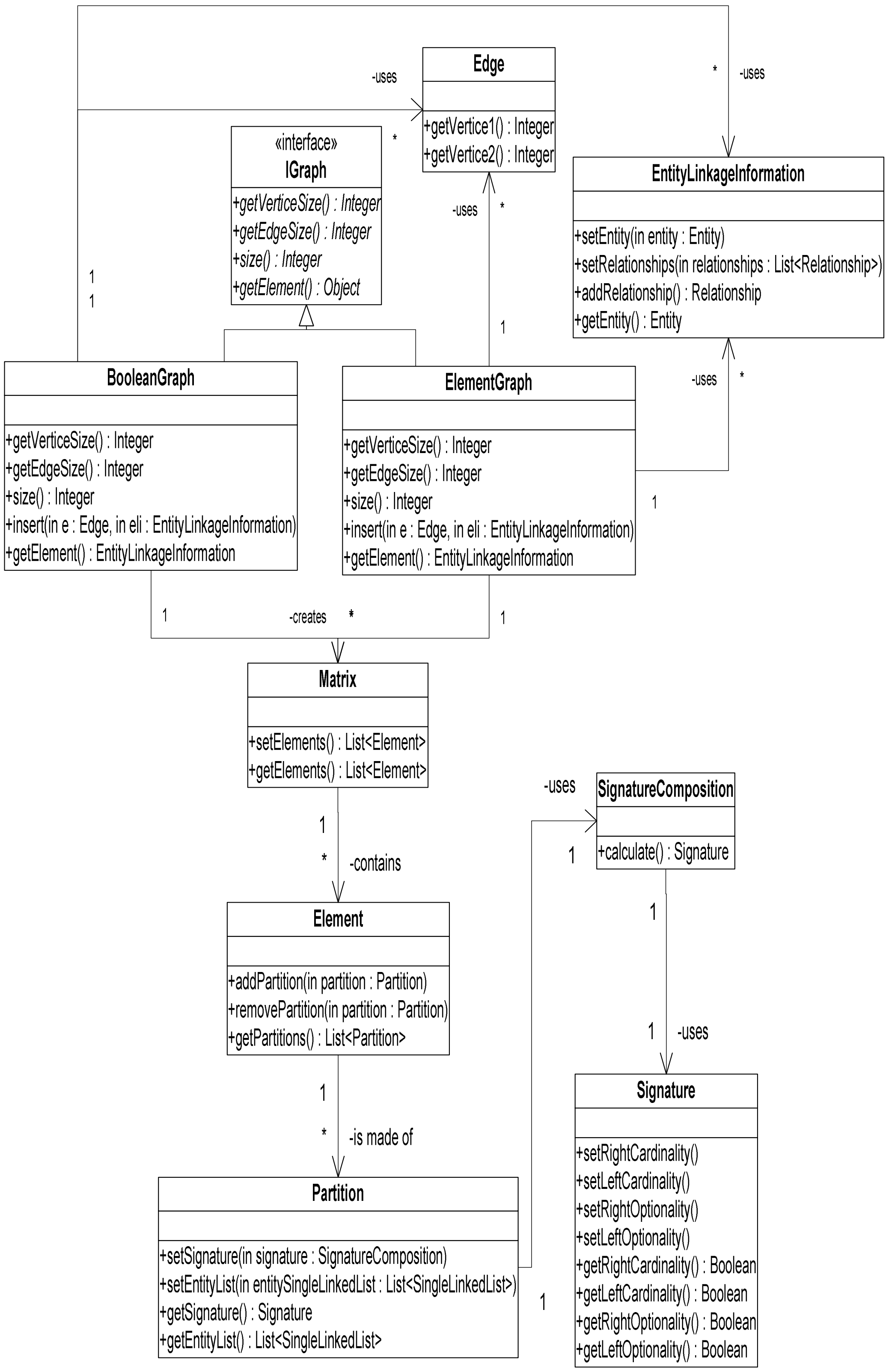

Edge
+getVertice1() : Integer
+getVertice2() : Integer
«interface»
IGraph
+getVerticeSize() : Integer
+getEdgeSize() : Integer
+size() : Integer
+getElement() : Object
EntityLinkageInformation
+setEntity(in entity : Entity)
+setRelationships(in relationships : List<Relationship>)
+addRelationship() : Relationship
+getEntity() : Entity
BooleanGraph
+getVerticeSize() : Integer
+getEdgeSize() : Integer
+size() : Integer
+insert(in e : Edge, in eli : EntityLinkageInformation)
+getElement() : EntityLinkageInformation
ElementGraph
+getVerticeSize() : Integer
+getEdgeSize() : Integer
+size() : Integer
+insert(in e : Edge, in eli : EntityLinkageInformation)
+getElement() : EntityLinkageInformation
Matrix
+setElements() : List<Element>
+getElements() : List<Element>
Element
+addPartition(in partition : Partition)
+removePartition(in partition : Partition)
+getPartitions() : List<Partition>
Partition
+setSignature(in signature : SignatureComposition)
+setEntityList(in entitySingleLinkedList : List<SingleLinkedList>)
+getSignature() : Signature
+getEntityList() : List<SingleLinkedList>
SignatureComposition
+calculate() : Signature
Signature
+setRightCardinality()
+setLeftCardinality()
+setRightOptionality()
+setLeftOptionality()
+getRightCardinality() : Boolean
+getLeftCardinality() : Boolean
+getRightOptionality() : Boolean
+getLeftOptionality() : Boolean
-uses
-creates
-contains
-is made of
1
*

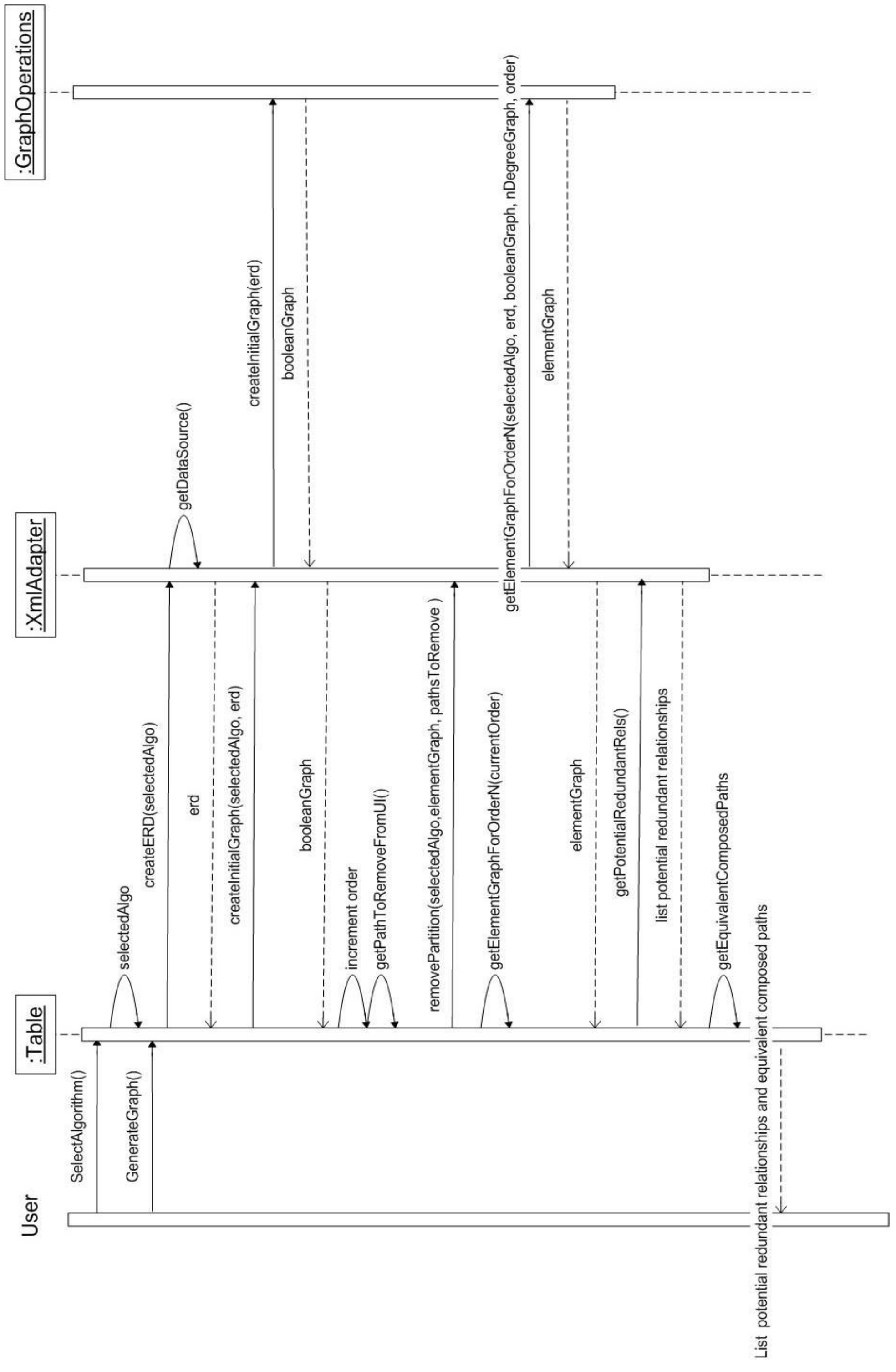
Figure 17– The graph operation class hierarchy

3.6.4.2 Sequence Diagram

Figure 18a) and 18b) describe the object flow relating to:

- the creation of the in-memory adjacency-matrix representing an ERD, stored in an XML file
- the discovery of potential redundant relationships in the ERD, for a given order

The creation of a `BooleanGraph` object is similar to the creation of an `ElementGraph` object. Therefore, Figure 18b) only illustrates the creation of the `BooleanGraph` when the `createInitialGraph(…)` method is called on the `GraphOperation` object. An equivalent flow would be generated for the creation of the `ElementGraph`. The sequence diagram follows the UML notation illustrated in M878 (Unit 5, pp. 29-33).



:GraphOperations
:XmlAdapter
:Table
User
SelectAlgorithm()
GenerateGraph()
selectedAlgo
createERD(selectedAlgo)
erd
getDataSource()
createInitialGraph(selectedAlgo, erd)
createInitialGraph(erd)
booleanGraph
booleanGraph
increment order
getPathToRemoveFromUI()
removePartition(selectedAlgo,elementGraph, pathsToRemove)
getElementGraphForOrderN(currentOrder)
getElementGraphForOrderN(selectedAlgo, erd, booleanGraph, nDegreeGraph, order)
elementGraph
elementGraph
getPotentialRedundantRels()
list potential redundant relationships
getEquivalentComposedPaths
List potential redundant relationships and equivalent composed paths

Figure 18 (a)

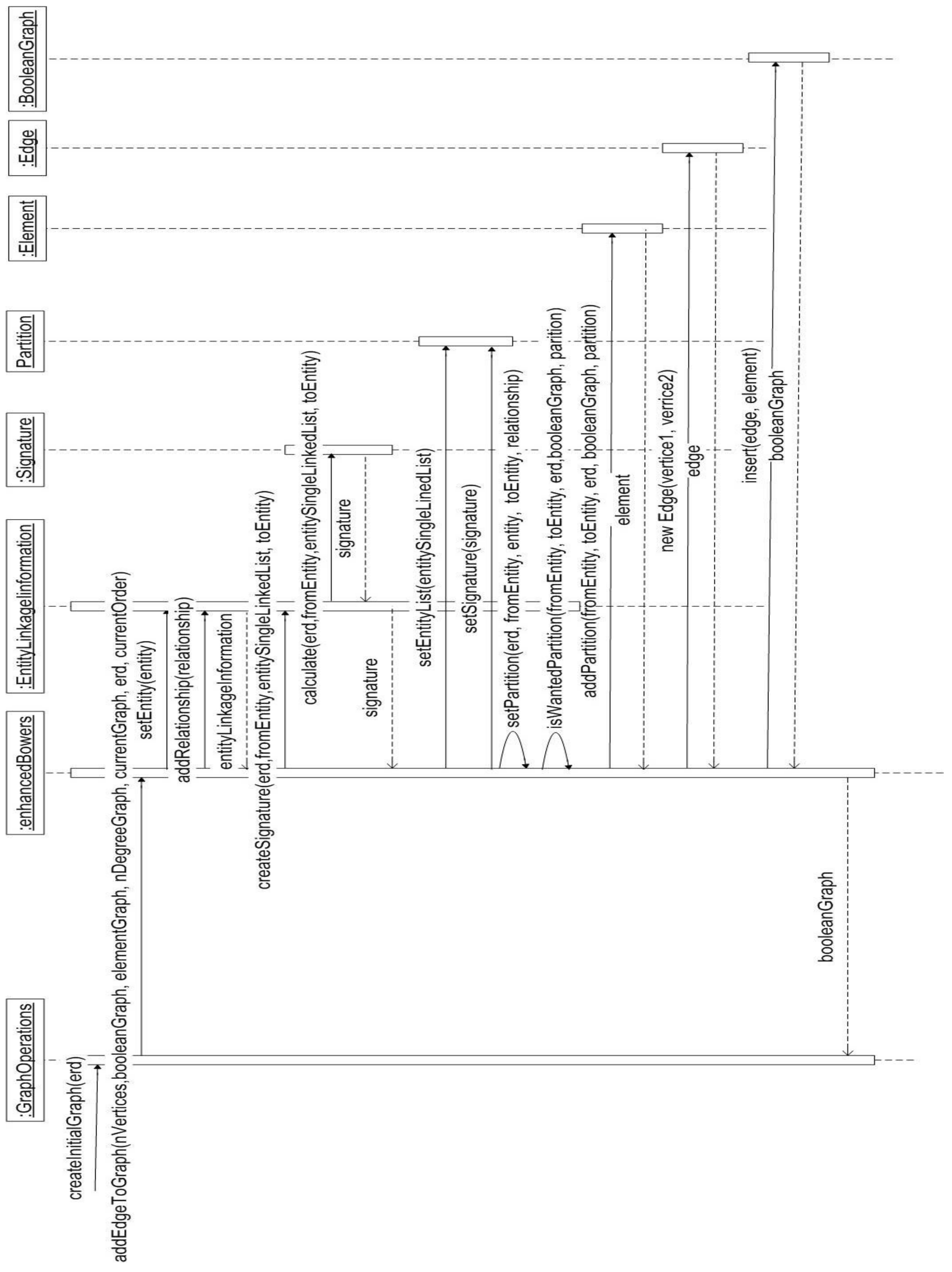

:GraphOperations
:enhancedBowers
:EntityLinkageInformation
:Signature
Partition
:Element
:Edge
:BooleanGraph
createInitialGraph(erd)
addEdgeToGraph(nVertices,booleanGraph, elementGraph, nDegreeGraph, currentGraph, erd, currentOrder)
setEntity(entity)
addRelationship(relationship)
entityLinkageInformation
createSignature(erd,fromEntity,entitySingleLinkedList, toEntity)
calculate(erd,fromEntity,entitySingleLinkedList, toEntity)
signature
signature
setEntityList(entitySingleLinedList)
setSignature(signature)
setPartition(erd, fromEntity, entity, toEntity, relationship)
isWantedPartition(fromEntity, toEntity, erd,booleanGraph, parition)
addPartition(fromEntity, toEntity, erd, booleanGraph, partition)
element
new Edge(verice1, verice2)
edge
insert(edge, element)
booleanGraph
booleanGraph

Figure 18 (b)

Figure 18 – Sequence Diagram for the discovery of potential redundant relationships in an ERD (from the user to the GraphOperation object)

3.6.5 Enhanced Bowers (2002) Algorithm

3.6.5.1 The Algorithm Description

3.6.5.1.1 The deletion of unwanted paths

The enhanced version of the Bowers algorithm uses the premises described in section 2.2.5. Furthermore, the ERD loops and cycles are obtained by allowing the diagonal element of the matrix to be non-zero (c.f. Table 4, p. 61). The matrix of order 2 is obtained by multiplying the initial Boolean matrix with itself. The order 3 and above are obtained by multiplying the matrix of precedent order to the initial matrix. As Bowers (2002) explained, the outcome of the matrix multiplication does not result in a list of scalar values, as shown by Lipschutz and Liposon (1997). Instead, each element of the matrix represents a possible path between two entities on the x- and y-axes of the matrix. The problem with introducing cycle is twofold. First, it produces path such as 'Manager-Client-Manager', as shown in Table 4/Order 2. This 'look back' cycle is not a truly order 2 cycle, as it necessitates moving between three entities on two different relationships. In the case shown in Table 4, there is only one relationship between the 'Client' and the 'Manager' entities. Therefore, all diagonal elements in the matrix of order 2 are eliminated (see the element highlighted in grey). The effect this has is to eliminate a number of other paths in matrices of higher order. These paths are also highlighted in grey in the matrix of order 3 of Table 4.

Order 1				
	M	C	P	CE
M	0	1	1	1
C	1	0	1	1
P	1	1	0	1
CE	1	1	1	0
Order 2				
	M	C	P	CE
M	C P CE	P CE	C CE	C P
C	P CE	M P CE	M CE	M P
P	C CE	M CE	M C CE	M C
CE	C P	M P	M C	M C P
Order 3				
	M	M	M	M
M	C.P C.CE P.C P.CE CE.C CE.P	M.C M.P M.CE P.C P.CE CE.C CE.P	M.C M.P M.CE C.P C.CE CE.C CE.P	M.C M.P M.CE C.P C.CE P.C P.CE
C	C.M C.P C.CE P.M P.CE CE.M CE.P	M.P M.CE P.M P.CE CE.M CE.P	M.P M.CE C.M C.P C.CE CE.M CE.P	M.P M.CE C.M C.P C.CE P.M P.CE

P	C.M C.CE P.M P.C P.CE CE.M CE.C	M.C M.CE P.M P.C P.CE CE.M CE.C	M.C M.CE C.M C.CE CE.M CE.C	M.C M.CE C.M C.CE P.M P.C P.CE
CE	C.M C.P P.M P.C CE.M CE.C CE.P	M.C M.P P.M P.C CE.M CE.C CE.P	M.C M.P C.M C.P CE.M CE.C CE.P	M.C M.P C.M C.P P.M P.C

M = Manager, C= Client, P = Project, CE = Contract Employee.

Table 4– Matrices representation of paths allowing for cycles

The inclusion of loops modifies the above behaviour slightly. Table 5 shows a recursive relationship on the ‘Manager’ entity. It is represented by an element set to 1 in the initial matrix. This increases the entries (also called partitions) in some of the elements of the order 2 matrix. The new ‘Manager’ partitions are shown with an asterisk (*). It is interesting to note that the path ‘Manager-Manager-Manager’ in the matrix of order 2. This path is not retained, as there is only one path from the ‘Manager’ to the ‘Manager’ entity (i.e. the recursion). This type of equivalent composed path is not truly a path of order 2, and is named a ‘recursive look back’ path. Therefore, although it is noted with an asterisk, this path is also flagged for deletion (i.e. in grey). The same logic applies for the matrix of order 3.

Order 1				
	M	C	P	CE
M	1	1	1	1
C	1	0	1	1
P	1	1	0	1
CE	1	1	1	0
Order 2				
	M	C	P	CE
M	M* C P CE	M* P CE	M* C CE	M* C P
C	M* P CE	M P CE	M CE	M P
P	M* C CE	M CE	M C CE	M C
CE	M* C P	M P	M C	M C P
Order 3				
	M	C	P	CE
M	M.M* C.P C.CE P.C P.CE CE.C CE.P	M.M* M.C M.P M.CE P.M* P.C P.CE CE.M* CE.C CE.P	M.M* M.C M.P M.CE C.M* C.P C.CE CE.M* CE.C CE.P	M.M* M.C M.P M.CE C.M* C.P C.CE P.M* P.C P.CE
C	M.M* C.M C.P C.CE P.M P.CE CE.M CE.P	M.M* M.P M.CE P.M P.CE CE.M CE.P	M.M* M.P M.CE C.M C.P C.CE CE.M CE.P	M.M* M.P M.CE C.M C.P C.CE P.M P.CE

P	M.M* C.M C.CE P.M P.C P.CE CE.M CE.C	M.M* M.C M.CE P.M P.C P.CE CE.M CE.C	M.M* M.C M.CE C.M C.CE CE.M CE.C	M.M* M.C M.CE C.M C.CE P.M P.C P.CE
CE	M.M* C.M C.P P.M P.C CE.M CE.C CE.P	M.M* M.C M.P P.M P.C CE.M CE.C CE.P	M.M* M.C M.P C.M C.P CE.M CE.C CE.P	M.M* M.C M.P C.M C.P P.M P.C

M = Manager, C= Client, P = Project, CE = Contract Employee.

Table 5– Matrices representation of paths allowing for cycles and loops

Figure 19 summarises the condition for deletion of a partition in matrix elements.

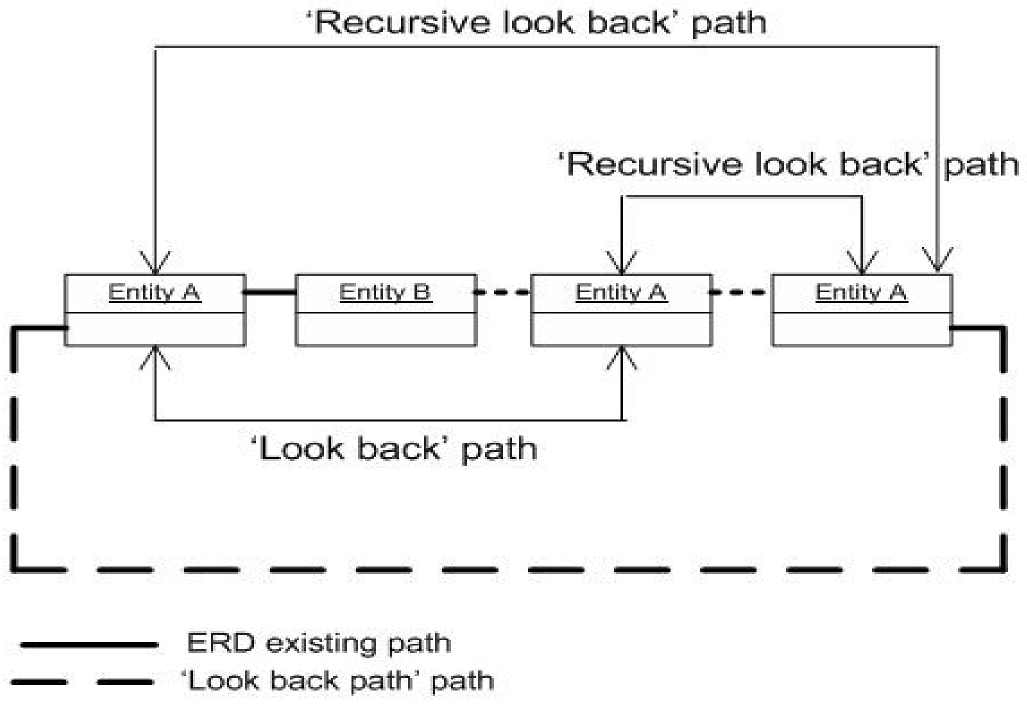


Figure 19 – Path deletion conditions (assuming there is no recursive relationship on Entity A). One of these conditions suffices for the path to be deleted.

Finally, the inclusion of multiple relationships is represented by an element in the initial matrix set to a value greater than one. Table 6 displays two relationships between the 'Manager' and 'Client' entities. In that case, both the 'Manager' and the 'Client' entities can be added as a partition to the matrix elements. They represent the extra possible path. These extra partitions are displayed by a double asterisk (**) in Table 6.

Order 1				
	M	C	P	CE
M	1	2	1	1
C	1	0	1	1
P	1	1	0	1
CE	1	1	1	0
Order 2				
	M	C	P	CE
M	M* C C** P CE	M* M** P CE	M* C C** CE	M* C C** P
C	….	….	….	….

M = Manager, C= Client, P = Project, CE = Contract Employee.

Table 6– Truncated matrices representation of paths allowing for cycles and loops and multiple relationships.

3.6.5.1.2 Ignore potential redundant paths that live on the equivalent composed path

Table 7 is the matrix representation of ‘Test N’ (c.f. Appendix A, p. 105). The paths highlighted in grey correspond to ‘look back’ equivalent composed path. The paths indicated with an asterisk (*) are ‘recursive look back’ paths. Therefore, they will be deleted. However, among the remaining order 2 equivalent paths, the two ‘Controller-Controller-Controller’ equivalent composed paths, indicated with a hash sign (#) will eventually be compared with the initial Boolean matrix paths ‘Controller-inspects-Controller’ and ‘Controller-trains-Controller’. Therefore, it is expected that when the signature composition of any equivalent composed paths are equal to one (or all) the recursive path(s) in the Boolean matrix, then the recursive relationship(s) will be flagged as redundant. In this example, the ‘Controller-inspects-Controller’ will be flag as redundant, because the ‘Controller-inspects-Controller’ and ‘Controller-trains-Controller’ equivalent paths have the same signature (for an adjacency-matrix of order 2). The other recursive path, ‘Controller-inspects-Controller’ would be flagged as potentially redundant for the same reason.

Order 1		
	C	I
C	2	2
I	2	1
Order 2		
C	C# C# I I	C C I*
I	C C I* I*	C C I*

C = Controller, I= Investment

Table 7– Matrices representation of 'Test I', for the orders 1 and 2

The temptation would be to define a rule that deletes equivalent composed paths which contain the equivalent path in the initial Boolean matrix. In the above example, it accounts to deleting the two 'C#' paths. However, this would create an extra problem in the case of cycles and the discovery of potential redundant relationship for orders greater than 2 (c.f. Figure 20, p. 67). If these two equivalent paths were to be deleted, then the equivalent composed path of order 4, represented by the path 'Controller-inspects-Controller' and 'Controller-trains-Controller' and 'Controller-audits-Controller' and 'Investment–is reviewed by–ExternalAuditors', would not be created and compared to the path 'ExternalAuditors-audits-Controller'existing in the original Boolean matrix. In this example, a potential redundant path would be missed. Therefore, the proposed solution is to keep these paths in the adjacency-matrix, for the reason mentioned above, however not consider them as potential redundant relationships.

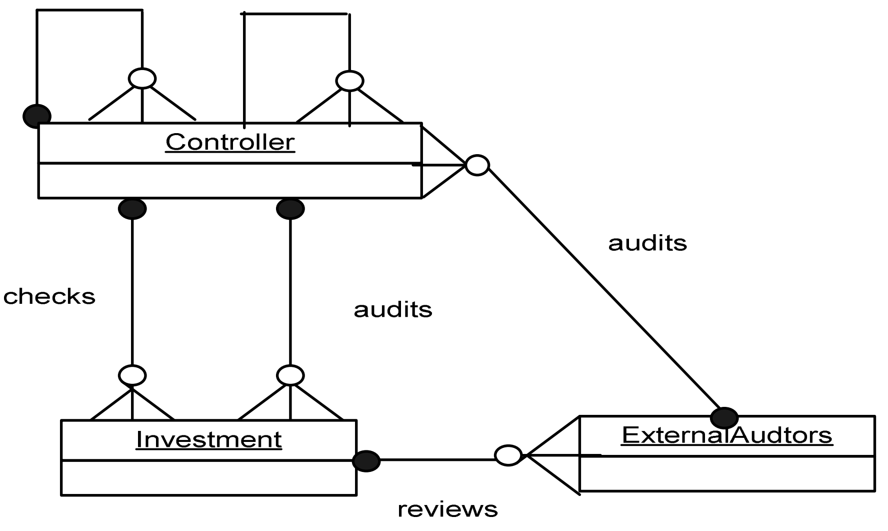


Figure 20– Schema representing a cycle added to 'Test I'

3.6.5.1.3 Ensure uniqueness of potential redundant relationships

By nature an adjacency-matrix compares relationships twice for elements existing on each side of the diagonal matrix elements ("mirror effect"). Table 8 illustrates that the relationships on the upper side of the matrix are repetitions of the relationships on the lower side of the matrix. The only difference is the signature. However, the creation of an order 2 adjacency-matrix or above could lead to the creation of equivalent composed paths that are identical in terms of path and signature. The only difference is the order in which the entity is shown in the equivalent composed path. In order to avoid repeating potential redundant paths and their equivalent composed paths, I have ignored any repetition of an existing potential redundant path.

Order 1			
	A	B	C
A	0	1	1
B	1	0	1
C	1	1	1

Table 8 – Initial Boolean matrix for an ERD

3.6.5.2 The Algorithm Implementation Logic

This section introduces the pseudo-code for the algorithm logic. The pseudo code standard follows the style presented by Dalbey J. (2003).

Algorithm Name: *EnhancedBowers*

High level algorithm logic and user interaction:

Step 1 – Create the initial adjacency matrix (i.e. initial matrix) that represents the relationship existence between two entities as well as their signature (Bowers 2002)

Step 2 - The initial matrix is multiplied with itself to obtain the 2nd order matrix (Bowers 2002)

Step 3 – Repeat Step 2 until the user decides to stop the search for the termination order selected by the user. The termination order is the last order for which potential redundant paths can be discovered. The termination closure is the highest order for which potential redundant paths can be discovered.

Model Assumptions:

The search for potential redundant relationships can be operated on

- binary and multiple relationships
- loops and cycles

Inputs: order – i.e. the maximum number of composition paths,
maxEntity – i.e. number of entities in the ERD,
elementGraph – i.e. the current `ElementGraph` object

Output: the matrix representation of the ERD indicating what relationships are potentially redundant (as well as the list of their equivalent composed paths) for a given order.

Algorithm pseudo-code body:

Step 1 – Create the initial graph, i.e. the matrix representation of the ERD. The entities stored in the business object are listed on the x- and y-axes of an in-memory object array (i.e. the Boolean matrix). The matrix intersection is set to a scalar value (e.g. 1) representing the number of relationships between two entities. If there is no relationship between two entities, then it is set to “0”.

Step 2 – For each order fabricate the element object

```
currentOrder =2, entity1 = 0, entity2 = 0

WHILE currentOrder < order

  WHILE entity1 < maxEntity

    WHILE entity2 < maxEntity

      IF (currentOrder = 2)

        Initialise the elementGraph object.

        Multiply the initial Boolean matrix with itself (i.e. matrix1 x matrix2) to
        create the matrix of order 2. The elementGraph matrix element is
        constituted of an array of entity linked lists. In this case, there is only one
        entity in the list. The element is obtained using the matrix multiplication
        mechanism described by Lipschutz and Liposon (1997) as well as Bowers
        (2002). The row by column matrix multiplication produces the matrix1
        row entity when the corresponding matrix2 column entity is set to "1".
        This result is stored at  a later stage in the elementGraph matrix element.

      ELSE

        Multiply the initial Boolean matrix with the elementGraph (i.e. matrix1 x
        matrix2) to create the matrix of order currentOrder. All other things are
        equal; except the elementGraph matrix element which contains a list of
        entities.

     ENDIF

     FOR each entity linked list in the previously generated entity linked list
     array

        Create a Partition object, named partition.

        Generate the signature of the path. This is achieved by performing
        Boolean Algebra (Lipschutz and Liposon, 1997) on the entity
        relationships along the equivalent composed path of a given relationship,
        in the initial Boolean matrix.
        An equivalent composed path is composed of the starting entity on the
        matrix y-axis, each entity in the entity linked list and the ending entity on
        matrix x-axis entity. This result is obtained by using the signature
        algorithm generation of the composition relationship path (c.f. 2.2.5, p.
        31).

        Add entity linked list to the partition.

        Add signature to the partition.

       IF (the partition path is not a 'look back' or a 'recursive look back' path)
       THEN

             Add partition to the corresponding matrix element (i.e.
             elementGraph).
```

```
            ENDIF (c.f. 3.6.5.1.1, p. 59)

            Add elementGraph to nDegreeGraph. The nDegreeGraph is the
            adjacency- matrix representation of the graph corresponding to a given
            order.

ENDFOR

      entity2 = entity2 +1

   ENDWHILE

     entity1 = entity1 +1

 ENDWHILE; currentOrder = currentOrder + 1

 ENDWHILE

 Step 4 – Get the list of potential redundant relationships

     FOR each elementGraph in the nDegreeGraph

          FOR each partition  in an elementGraph

                  IF the partition's signature equals the equivalent initial Boolean
                 graph's signature THEN

                            Label the initial BooleanGraph relationship as potentially
                            redundant
                           Attach the partition to the initial BooleanGraph potential
                           redundant relationships
          END FOR

END FOR

Step 5 – Get the list of equivalent potential redundant relationships

Create the equivalent composed path for each relationship labelled as potentially
redundant.

FOR each potential redundant relationships

   FOR each partition attached to the potential redundant relationships

             Fabricate a human readable path between the starting entity on the
             matrix y-axis and the first partition entity. The readable path looks like
             Entity1-Relationship Name-Entity2 (e.g. Controller-checks-Investment)

             Add the fabricated path between the first partition entity and the next

             Add the fabricated path between the last partition entity and the ending
          entity on the matrix x-axis.
  END FOR
END FOR
Step 6 – Display the potential redundant relationships in a user interface, when
the initial matrix path is not contained in its equivalent composed path (c.f.
3.6.5.1.2, p. 65). Furthermore ensure uniqueness of the potential redundant
relationships (c.f.3.6.5.1.3, p. 67).
Step 7 – The user flags a relationship as being redundant
Step 8 – Delete the redundant relationship(s) from elementGraph object.
```

3.6.5.3 The Model Limitations

- There is no support for the correct detection of '*many:many:many'* cycles. It is expected that the search will indicate that each relationship on the cycle is potentially redundant when they are compared to their equivalent composed path.

- The algorithm relies on the user to indicate the order at which the search should be stopped. There is no internal mechanism that generates the termination order (i.e. the transitive closure). In the case of an ERD with no loop, the termination condition is 'E-1', for an 'E'-entity ERD (Bowers, 2002). It will be '(E-1) + (RR)'; 'RR' representing the number of recursive relationships.

- As mentioned by Bowers (2002), the algorithm detects cycles in order of increasing length. Therefore, the early deletion of shorter cycles can potentially eliminate the existence of longer cycles, before they are even detected.

Chapter 4 Data Collection

4.1 Data Sources

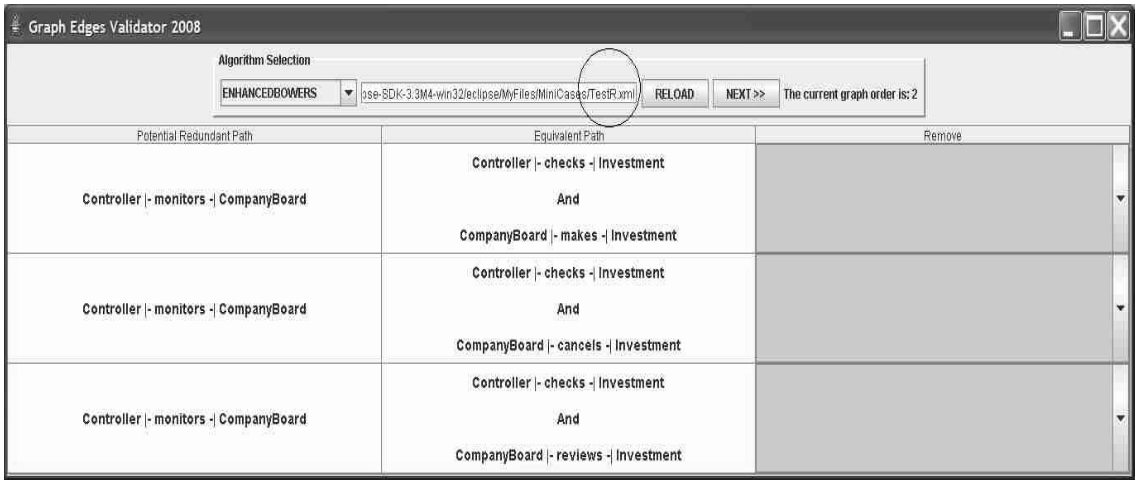
The data source has been generated by collecting data about the prototype. Two types of data have been collected:

- The execution time: this refers to the CPU time required to perform a single design action (such as finding all potential redundant relationships in the entire ERD). Execution time is not impacted by human factors, as it is measured from the point at which the tool commences an action until that action is complete. Processing time is however influenced by the complexity of the design input; the complexity of the implemented algorithm; and the specification of the processor of the PC in use (Noah and Williams, 2000). Precisely, it is obtained by implementing Java timers in the program, which reads code paths execution times, given a data set input.
- The quality of the search: this is achieved by counting the number of 'correct/incorrect and missed identifications' for each test case, against the expected results.

4.2 Data Collection Process

The execution time data collection is conducted using the Graph Edge Validator 2008 (GEV2008) prototype against each of the sparse and dense graphs represented in the XML test case files listed in Appendix E. The transitive closures are defined manually for each test case and follow the '(E-1) + (RR)' definition (c.f. section 3.6.5.3, p. 71). For each test case, the execution time required to complete a generation of the '(E-1) + (RR)' order adjacency-matrix is stored to a text file. This data are used without alteration or aggregation for the analysis (c.f. Appendix F, p. 140).

The discovery quality data collection is conducted using the GEV2008 prototype against each of the XML test case files in Appendix B. For each use case, the GEV2008 prototype produces a set of potential redundant relationships and equivalent composed path. These results are recorded and form the base of the raw data that will serve for the quality comparison analysis. Figure 21 illustrates an example of two pieces of information provided by the prototype. The left column lists potential redundant relationships, whereas the right column displays their equivalent composed path. This is obtained by selecting the *EnhancedBowers* algorithm, the test case (e.g. the Test R, circled Figure 21) and the order (e.g. order 2). This data are recorded as one potential redundant relationship and three discovered equivalent composed paths. All the test files are available in the Appendix B.


Graph Edges Validator 2008
Algorithm Selection
ENHANCEDBOWERS
ose-SDK-3.3M4-win32/eclipse/MyFiles/MiniCases/TestR.xml
RELOAD
NEXT >>
The current graph order is: 2

Potential Redundant Path	Equivalent Path	Remove
Controller |- monitors -| CompanyBoard	Controller |- checks -| Investment And CompanyBoard |- makes -| Investment	
Controller |- monitors -| CompanyBoard	Controller |- checks -| Investment And CompanyBoard |- cancels -| Investment	
Controller |- monitors -| CompanyBoard	Controller |- checks -| Investment And CompanyBoard |- reviews -| Investment	

Figure 21 - Example of the discovery of a potential redundant path and its equivalent composed path for the Test R (for the order 2).

Each test case result raw data (c.f. Appendix D, p. 138) is recorded in a table that indicates:

- the order: the number of relationships required to move from the path initial entity to the path final entity
- the relationship type: whether the test case contains multiple or only single relationships
- the existence of a recursion: whether the test case contains at least one recursive relationship
- the existence of a cycle: whether the test case contains at least one cycle
- the number of potential redundant paths that are correctly or incorrectly identified and paths that have been missed
- the number of equivalent composed paths that are correctly and incorrectly identified and paths that have been missed

When no potential redundant path is found, and no potential redundant path is expected (e.g. Test A, p. 105); a special tag named *'Not Available'* is recorded in the column *'# Correct Potential Redundant Paths Discovery'* (c.f. Appendix D, p. 138).

4.3 Preliminary Analysis

4.3.1 Efficiency Analysis

The execution time analysis is be represented by a graphic that plots the algorithm execution time (y-axis) against an increase of i) vertices, ii) edges, iii) edges and vertices (x-axis). This data are used to deduce the algorithm growth rate. These results also demonstrate whether the algorithm is more efficient in a dense or sparse graph.

4.3.2 'Quality Discovery' Analysis

Table 9 illustrates the first step of the process applied to the raw data (c.f. 4.2, p. 73) in order to obtain the percentage of the number of correct, incorrect or missed potential redundant relationship redundancies, and equivalent composed path discoveries across all the ERD paths (c.f. Appendix D, p.138). The pseudo code standard follows the style presented by Dalbey J. (2003).

C	It corresponds to the number of discovered correct potential superfluous relationships (or equivalent composed paths)
M	It corresponds to the number of missed potential superfluous relationships (or equivalent composed paths)
IC	It corresponds to the number of discovered incorrect potential superfluous relationships (or equivalent composed paths)

Correctness Metric (CM)	IF C = ‘NOT AVAILABLE’ THEN
	    CM = 100%
	ELSEIF C = 0 THEN
	    CM = 0%
	ELSE
	    CM = 100 * C / (C+ IC +M)
	ENDIF

Incorrectness Metric (ICM)	IF IC = 0 THEN
	    ICM = 0%
	ELSEIF
	    ICM = 100 * IC / (C+ IC +M)
	ENDIF

Missed Metric (MM)	IF M = 0 THEN
	    MM = 0%
	ELSE
	    MM = 100 * M / (C+ IC +M)

Table 9 - The success, failure and missed metrics definition for potential redundant paths and equivalent composed paths

The second step of the process is to group the result per ‘Type’ and generate the metrics of Table 9 for each type. A ‘Type’ corresponds to a combination of the relationship, recursive and cycle. Four types have emerged from the test cases:

- ‘Type 1’ contains test cases that have only single relationships, at least one recursion and no cycle
- ‘Type 2’ contains test cases that have only single relationships, no recursion and at least one cycle
- ‘Type 3’ contains test cases that have multiple relationships, at least one recursion and at least one cycle
- ‘Type 4’ contains test cases that have multiple relationships, no recursion, and at least one cycle
- ‘Type 5’ contains test cases that have single relationships, recursions, and at least one cycle

Chapter 5 Results

This chapter presents the analysis of the results deriving from the experimental procedure outlined in section 3.5 and Chapter 4.

5.1 Execution Time Measurement Result

Figure 22 and Figure 23 show the execution times in milliseconds for a sparse and dense ERD, which do not support recursive relationships. The *enhancedBowers* algorithm running time tends to grow by a factor of 8 when the number of relationships is doubled. This supports the conclusion that the algorithm is time proportional to $O(R^3)$, where R stands for the number of relationships. However, the picture is not as clear when the number of entities is doubled. The growth rate seems to be proportional to E^3 up to a 40 entities ERD, with E representing the number of entities. When it is doubled to 80 entities, then the time is multiplied by a factor of 15. These results are very different from the ones obtained by Bowers (2002) and Sedgewick (2004). Bowers (2002) showed that the growth rate of his algorithm could potentially be of type $O(E^{E+1})$. Sedgewick (2004) illustrated that a digraph Deep-First Search algorithm based on an adjacency-matrix should be proportional to E^3. One cause of the slow down could be the extra processing required for the deletion of unwanted paths (c.f. 3.6.5, p. 59).

The test on doubling recursive relationships is interesting as the application throws a “java.lang.OutOfMemoryError: Java heap space”, for an ERD containing five entities and eight recursive relationships. This error indicates that the program consumes more resources than the ones that are allocated for its successful execution, Atkey (2006). The test, which generates this exception, is based on a schema represented in Figure 14 (p. 47), where entities A, B, C and D initially contain two loops each. According to the formula described in section 3.6.5.3 (p. 71), the transitive closure corresponds to an adjacency-matrix of order 12 (i.e. 5 entities -1 entity + 8 recursive relationships). However, the program failed the execution on the initial schema, at the order 9. This is caused by the number of partitions which need to be created in the element. Each partition is represented by a Java *hashtable* object which has a number of entries limited to the test machine memory. In this case, the algorithm creates more entries than can be handled, which in turn generate this exception. This shortcoming is not due to the algorithm design, but to the way the ERD matrix representation is stored in memory. This problem may not have occurred that early in the process if the schema had been persisted, and only a subset of schema had been loaded into memory to compute the adjacency-matrix. More details on this topic is available in section 6.2.2, p. 91.

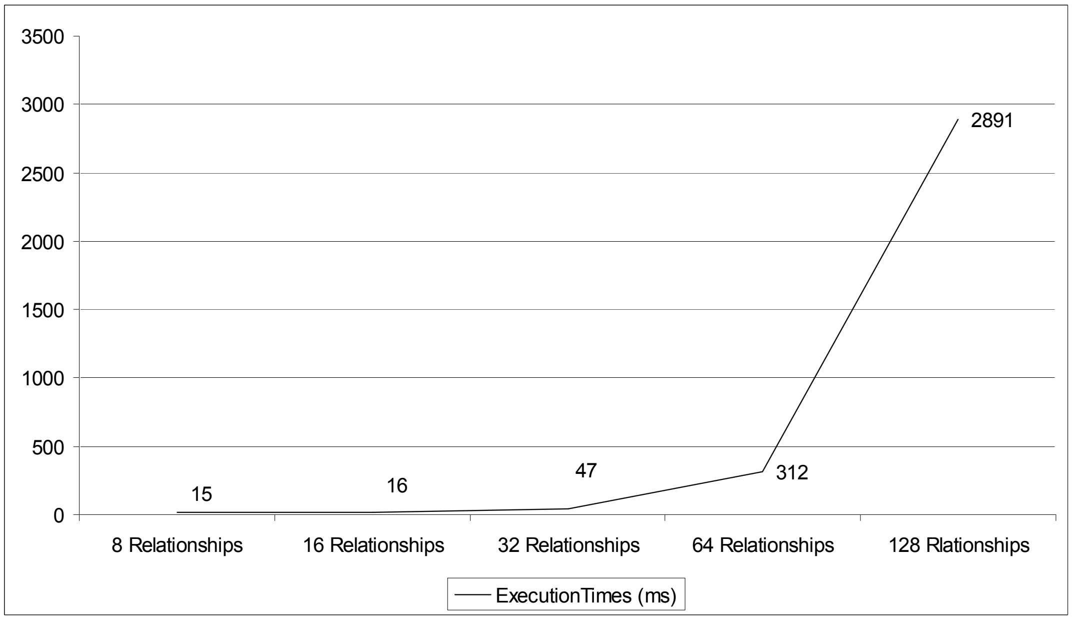


Figure 22 – Execution time in milliseconds (ms) for a sparse ERD (5 entities)

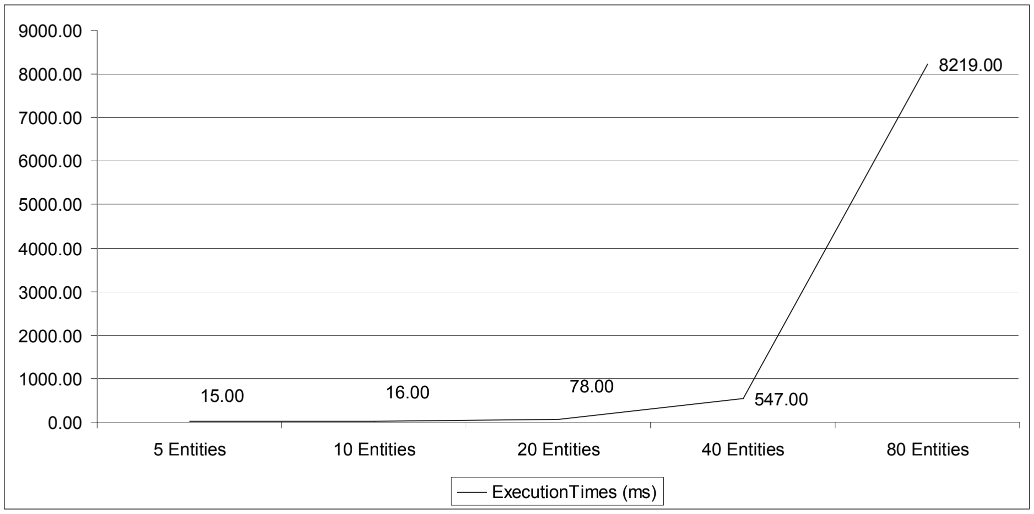


Figure 23 - Execution time in milliseconds (ms) for a dense ERD, where the number of relationships is double the number of entities (as there are two relationships per entity)

5.2 Quality Discovery Result

5.2.1 Potential Redundant Path Analysis

Figure 24 presents the success, failed and missed potential redundant paths discovery grouped by types. It appears that the proposed algorithm indicates incorrect results in 25%, 35% and 17% of cases for respectively 'Type 2', 'Type 3' and 'Type 5'. While 'Type 1' and 'Type 4' show a 100% success rate. The following paragraphs aim at explaining these errors.

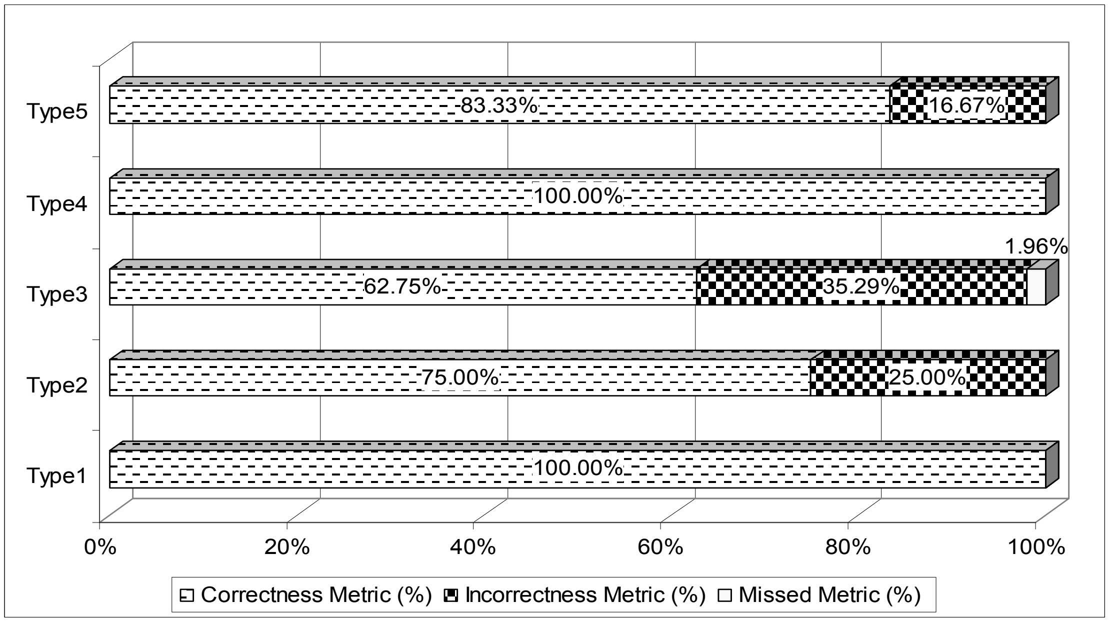


Figure 24 - Potential Path Redundancy Discovery Comparison

The raw result (c.f. Appendix D, p.138) indicates that the source of 'Type 2' incorrect potential redundant paths appears in 'Test K'. This is a '*many:many:many*' cycle (c.f. Appendix A, p.105). According to the conceptual model (c.f. 3.6.5.1, p. 59), the three relationships should be displayed in the

GEV2008 prototype, instead the latter displayed the relationship ‘chairs’ twice and ignore the relationship ‘takes’. This is due to the implementation of the `getPotentialRedundantRelationships(…)` method, which selects incorrectly potential redundant relationships in this particular case. Even if the implementation of this method was modified, the problem of *‘many:many:many’* cycle relationships would still not be resolved. The model and implementation limitations (c.f. 3.6.5.3, p. 71) mention this shortcoming. Bowers (2002) indicates that the algorithm could draw the user’s attention to this type of structures, and could offer the designer the choice to alter them, when required.

The *EnhancedBowers* algorithm produces 35% of incorrect potential redundant path for ‘Type 3’. This is caused by several factors. The ‘Test I’ produces two incorrect potential redundant relationships out of the four displayed:

- ‘Controller-inspects-Controller’, and
- ‘Controller-trains-Controller’

This is due to the limitation in the implementation of *EnhancedBowers* algorithm proposed in section 3.6.5.1. Each entity in the entity linked list that forms a partition, for a given element of the matrix, contains the definition of its relationship with the next entity. The last entity in the list also contains the definition of its relationship with the destination entity on the x-axis of the matrix. However, there is a loss of relationship context between the first entity of the linked list and the source entity on the y-axis of the matrix. This is the root cause of the creation of the two extra potential redundant relationships. In fact, the potential redundant relationship ‘ignore rule’ (c.f. 3.6.5.1.2, p. 65) should have eliminated them, however this did not happen. Table 10 illustrates only the two equivalent composed paths, made of the recursive relationship. The thin plain arrow represents the ‘Controller-inspects-Controller’ partition. The large plain arrow represents the ‘Controller-trains-Controller’. When the ‘Controller-inspects-Controller’ partition is compared against the initial Boolean matrix ‘Controller-inspects-Controller’ path, the potential redundant relationship is ignored, as mentioned in (c.f. 3.6.5.1.2, p. 65). However, when it is compared against the ‘Controller-trains-Controller’ path, the ‘ignore rule’ fails. This is because although the theoretical equivalent composed path contains the ‘Controller-trains-Controller’ arc, it is not captured in the relationship partition. This is represented with the dotted arrow in Table 10. The same principle applies to the ‘Controller-trains-Controller’ partition and initial Boolean matrix ‘Controller-trains-Controller’ path. This is why these two incorrect potential redundant relationships are created.

Order 1		
	C	I
C	2	2
I	2	1
Order 2		
C ---------> --------->	C C ….	….
I	….	….

C = Controller, I= Investment

Table 10– Summary of matrices representation of ‘Test I’, for the order1 and 2

‘Test O’ and ‘Test E’ display the same symptoms as the one explain in ‘Test I’. In ‘Test O’, the ‘Manager-chairs-CompanyBoard’ relationship is compared to the equivalent composed path ‘Manager-chairs-CompanyBoard’ and ‘CompanyBoard-elects representative- CompanyBoard’ path. The loss of relationship context between the entity “Manager” on the matrix y-axis and the first entity in the entity linked list, ’CompanyBoard’ fails the potential redundant relationship ‘ignored rule’, mentioned in (c.f. 3.6.5.1.2, p. 65).

‘Type 3’ also indicates that in 2% of the constructed cases, relationships are missing. The problem appears in ‘Test S’. This is caused by two effects; the implementation limitation highlighted in test cases ‘I’ and ‘O’ coupled with the rule that ensures uniqueness of potential redundant relationships (c.f. 3.6.5.1.3, p. 67). As a partition misses the initial relationship connection, the partition ‘ContractEmployee-WorksOn-Project’ is flagged as a duplicate. The reason is that the partition was already met when the path ‘Manager-supervises-ContractEmployee’ path was compared with the equivalent composed path ‘Manager-manages-Project’ and ‘ContractEmployee-WorksOn-Project’.

The *EnhancedBowers* algorithm produces 17% of incorrect potential redundant path for ‘Type 5’. The problem appears in ‘Test N’. The cause has already been explained in details for ‘Test I’ and ‘Test O’.

5.2.2 Equivalent Composed Path Analysis

Figure 25 presents the success, failed and missed equivalent composed paths discovery grouped by types.

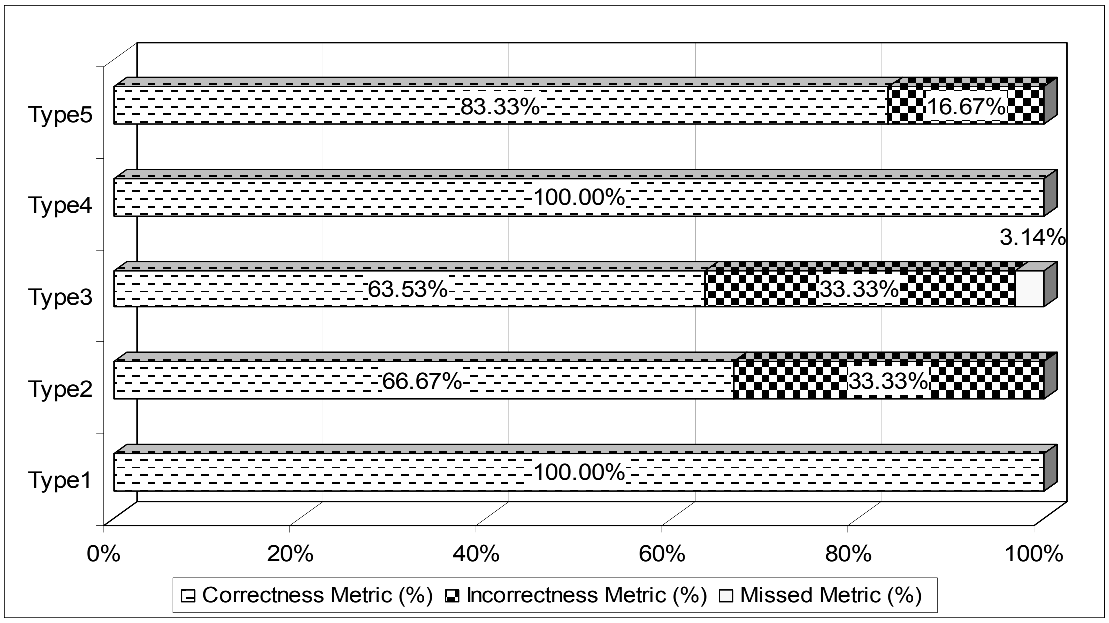


Figure 25 - Potential Equivalent Path Redundancy Discovery Comparison

The raw result (c.f. Appendix D, p.138) indicates that the source of ‘Type 2’ incorrect equivalent composed path appears in ‘Test K’ and ‘Test R’. The error induced in ‘Test K’ is due to the creation of the incorrect potential redundant relationships, as mentioned in section 5.2.1. The ‘Test R’ incorrect equivalent composed path, ‘Controller – checks – Investment’ and ‘CompanyBoard – reviews – Investment’ is due to a limitation in the implementation of the `pathBuilder(…)` method. This implementation does not operate correctly in the context of multiple relationships. The search only matches the first relationship found between two entities and skips any other existing relationships, between these two nodes.

All ‘Type 3’ incorrect/missed equivalent composed paths are due to incorrect/missed potential redundant relationships to which they are attached, except for ‘Test I’. The ‘Controller – trains – Controller and Controller – checks – Controller’ as well as the ‘Controller – trains – Controller and Controller –

audits – Controller' equivalent composed paths are respectively missing for the potential redundant relationships 'Controller –audits- Investment' and 'Controller –checks- Investment'. This is the same issue as above; the implementation does not operate correctly in the context of multiple relationships departing from the first entity.

5.3 Validation

5.3.1 Discovery Capability Comparison

Table 11 compares the 'quality discovery' ability of each approach described in the literature review (c.f. section 2.2 p. 15), and the solution that was the focus of this thesis, namely the *EnhancedBowers* algorithm. The comparison assumptions are as follows:

- There was not any attempt to code solutions proposed in the literature review, due to time constraints. Instead, I have used the results provided by the literature to predict the ability of each methodology to discover potential redundant relationship for the cases listed in Table 11.
- The 'Minimal Covering of FD-graphs' approach is not included in this qualitative comparison. It is virtually impossible to deduce whether it can find potential redundant relationships in the proposed test cases 'intuitively', due to the inherent algorithm complexity.

This analysis suggests the hypothesis that the *EnhancedBowers* algorithm is the most suitable solution for discovering potential redundant relationships in ERD diagrams, as it discovers most of potential redundant relationships in single/multiple relationships, loops, cycles and implied cycles (c.f. Test G, p. 108).

				HYBRID APPROACH	
Discover potential redundant relationship in …	***Real-World Knowledge (c.f. section 2.2.2, p. 20)***	***Analysis Cardinality and Participation Constraints (c.f. section 2.2.3, p. 24)***	***Analysis of Shared Attributes and Inclusion Constraints (c.f. section 2.2.4, p. 27)***	***Bowers2002 (c.f. section 2.2.5, p. 31)***	***Enhanced Bowers2002***
Single Relationships (with no cycle)	***Inconclusive***(1)	***No***	***No***	***Yes***	***Yes***
Multiple Relationships (with no cycle)	***Inconclusive***(1)	***No***	***No***	***No***	***Yes***(5)
Recursion	***Inconclusive***(1)	***No***	***No***	***No***	***Yes***(5)
Cycle (except many-to-many)	***Inconclusive***(1)	***Yes***(2)	***Inconclusive***(3)	***Yes***(4)	***Yes***(5)
Many-to-many Cycle only	***Inconclusive***(1)	***No***	***Inconclusive***(3)	***No***	***No***
Comments	(1) ***Potential Redundant Relationships are only discovered when two or more relationships have the same meaning.*** (2) ***For the cases defined by this methodology*** (3) ***Potential Redundant Relationships are only discovered when the primary / and posted foreign key attributes name are not identical in all entities in the 'many:many:many' cycle, in a single relationship structure. It fails in multiple relationship structures.*** (4) ***Except in the case of implied cycle.*** (5) ***Except for the few cases listed in section 5.2.***				

Table 11 – Summary of the comparison of the studied approaches discovery abilities

Chapter 6 Conclusions and Recommendations

This final chapter summarises the research results and propose a set of recommendations to improve the current outcome. It also suggests lines of inquiry that may be carried out in a future research.

6.1 Result Summary

The first objective was to assess whether the *EnhancedBowers* algorithm can discover more potential redundant relationships for a list of fabricated test cases, than the approaches studied in the literature review. The result of the analysis showed that this was the case. The analysis revealed that the cause of the discovery failures was due to an oversight in the creation of the partitions in the matrix element. It is expected that solving this issue should ensure a significantly higher success rate for the fabricated test case. However, there are multiple points of failure when searching for equivalent composed paths. The second objective of the thesis was to verify whether the algorithm was scalable. The result of the analysis indicated that the algorithm is scalable with sparse ERDs. However, this was not the case for dense graphs or graphs with many loops. The following section formulates a set of heuristics to improve the current solution in terms of scalability and discovery capability.

6.2 Recommendations

6.2.1 Algorithm Implementation Improvements

6.2.1.1 Potential redundant relationship enhancement

The current implementation of the algorithm is modelled on an adjacency-matrix representation. The `Matrix` object contains `Element` objects, which in turn contains `Partition` objects. The main drawback is that the `Partition` object contains a linked list of entities that form a path that does not contain a linkage between the first entity in the linked list and the starting entity (on the matrix y-axis). This is a limiting factor in the case of single recursive and multiple relationships. This is currently the cause of most of the test cases failure. Therefore, I recommend either:

- storing the matrix y-axis entity in the entity linked list, or
- adding a relationship on the first entity in the entity linked list that qualifies the linkage between it and the matrix y-axis entity

Furthermore, the algorithm cannot discover potential redundant relationships in the '*many:many:many*' cycle. One of the recommendations, as suggested by Bowers (2002) would be to flag these cycles in the user interface as non-supported structures, and let the user decide on whether to remove or keep them in the ERD. Another solution could be to leverage from the result of Queralt and Teniente (2006), and analysing whether verifying the OCL of a schema could help in this case. I also propose that the application should remove as much as the routine work as possible. The more complex potential redundancies should be left to the user to resolve by visual inspection.

6.2.1.2 Equivalent composed paths enhancement

The failures encountered during the fabrication of equivalent composed paths are due to the inability of the current implementation of the *EnhancedBowers* algorithm to search through multiple relationships between two entities, and reconstruct the equivalent paths (c.f. section 5.2.2 p. 84). I can propose two approaches to improve the current situation:

- Extend the current implementation of the 5th step (c.f. section 3.6.5.2, p. 68) by fabricating the number of equivalent composed paths matching the number of relationships between two entities in a path.

- Remove the step 5 logic and re-implement the logic in step 4 (c.f. section 3.6.5.2, p. 68). In step 4, the list of relationships between entities is already known. Therefore the list of equivalent composed paths could already be created. This seems to be the best solution, as this would reduce significantly the code complexity and maintainability. This should also improve the execution run time efficiency, as the logic of step 5 requires the parsing of the ERD. This could be avoided as all the necessary data should be available in step 4.

6.2.2 Architecture Modification

The current architecture (c.f. 3.6.3, p. 50) relies on loading the entire ERD from an XML file, and transforming the XML file into a business object that is understood by the algorithm. This is a limiting factor in terms of memory foot print and the ability of the object collection to store increasing elements/partitions for the adjacency matrix. This solution is not scalable. This was shown in particular in section 5.1. Therefore, I recommend:

- loading the initial Boolean matrix
- generating the next adjacency-matrix of order E+1
- storing the latter and reload it for the next iteration, if and when required

Furthermore, this solution has the added advantage of improving the resiliency of the solution. If for any reason, the application crashes, then the user could start from the last saved point (assuming there is no hardware failure). This is currently not the case. The analysis work would need to be started again from the beginning.

6.2.3 Scalability Improvements

Currently, the adjacency-matrix generation is implemented on a single thread. Therefore, each element of the matrix is generated one after the other. The execution time efficiency could be greatly improved in a multi-threaded environment. In this case, a ‘divide and conquer’ approach could be used to compute the matrix elements. The matrix could be divided in sections, for example each column would represent a section. Each thread could work independently and in parallel on its given section. Once all the matrix elements have been generated on a section, the section would be marked as complete. The next available thread could work on any section that is not currently being processed or complete.

6.2.4 Performance Improvements

I originally selected the Java^{TM} language as it was the closest fit to my programming knowledge. The main advantage of the Java^{TM} language is that it is compiled by the compiler into byte code, which can be executed on any computer. However, it requires an interpreter to execute the program. That is different from other language such as C++, where only a compiler is required to transform the code into the processor/operating system object code (Spell, 2000). This has a performance impact in term of algorithm execution performance, as illustrated by Stern (2001). Although the impact would be minor with relative small and sparse diagrams, it could be much greater in the case of very large and dense ERDs. I recommend moving the matrix computation to C++.

6.2.5 Complexity Analysis

It would propose to review algorithm complexity, using measures/metrics such the Line of Code metrics (Nurminen 2003; Pressman 2000), the Volume size (Mills, 1988; Hamer and Frewin, 1982; Nurminen 2000), Cyclomatic Number

(Elshoff and Marcotty, 1978; McCabe 1976; Nurminen 2000). This would help establish where the algorithm could be improved, to enhance its flexibility and maintainability, as well as reduce its overall complexity. That could also have a positive impact on the performance, as illustrated in section 5.1.

6.3 Future Research

This experiment focuses on relatively small ERDs, in terms of number of entities and relationships. Therefore, I would recommend studying the *EnhancedBowers* algorithm in the context of 'real life' ERDs. The researcher could verify whether the growth rates discussed in this thesis hold with larger diagram.

References

Arthorne, J. and Laffra, C. (2004), *Official Eclipse 3.0 FAQs*, Boston, USA, Addison-Wesley.

Atkey, R (2006), 'Specifying and Verifying Heap Space Allocation with JML and ESC/Java2, *In 8th Workshop for Formal Techniques for Java-like Programs*, Nantes, France.

Ausiello G., d'Atri A. and Sacca D. (1983), 'Graph Algorithms for Functional Dependency Manipulation', *Journal of the Association for Computing Machinery*, *Vol 30, No 4*, pp 752-766.

Azman, S. and Noah M. (2000), 'Intelligent System for Conceptual Modelling of Databases', *TENCON Proceedings*, *Vol2*, pp. 504-508.

Batra, D. and Zanakis, S. (1994), 'A conceptual database design approach based on rules and Heuristics', *Eur. J. Inf. Syst. 3*, pp. 228–239.

Booch, G., Rumbaugh J. and Jacobson I. (2005), *Unified Modelling Language User Guide*, *2nd edition*, Amsterdam, Holland, Addison-Wesley.

Bowers D.S. (2000), 'Database schema Improvement Techniques for CASE Tools', *proc. UKAIS conference*, Cardiff, UK.

Bowers D.S. (2002), 'Detection of Redundant Arcs in Entity Relationship Conceptual Models', *proc. Int. Workshop on Conceptual Model Quality*, Tampere, Finland.

Burleson D. (n.d.), Chapter 2 Physic Entity Design for Oracle [on line], *http://www.remote-dba.net/tp_data_warehouse_relationships_pointers.htm,* [Accessed 15 March 2007].

Connolly, T. and Begg, C. (2000), *Database Solution – A step-by-step guide to building databases,* Harlow, UK, Addison-Wesley.

Dalbey J. (2003), Pseudocode Standard, *http://www.csc.calpoly.edu/~jdablbey/SWE/pdl_std.html,* [Accessed 2 July 2007].

Dullea, J. and Song, I-Y. (1997), 'An Analysis of Cardinality Constraints in Redundant Relationships', *Proc. 6th CIKM*, pp. 270-277.

Dullea, J. and Song, I-Y. (1998), 'An Analysis of the structural validity of ternary relationships in entity relationship modelling', *Proc. 7th CIKM*, pp. 331-339.

Elshoff, J.L. and Marcotty, M. (1978), 'On the use of the Cyclomatic Number to Measure Program Complexity', *ACM*, New York, USA, pp. 29-40.

Goelman, D. and Song, I-Y. (2004), 'Entity-Relationship Modelling Re-revisited', *Springer-Verlag Berlin, LNCS 3288*, Berlin, pp. 43–54.

Goodrich, M.T. and Tamassia R. (2006), *Data Structures & Algorithms 4th edition*, Hoboken, USA, Wiley.

Hamer, P.C. and Frewin G.D. (1982), 'Halstead's Software Science – A Critical Examination', *IEEE Computer Society,* Los Alamitos, USA, pp. 197–206.

Holzner, S. (2004), *Eclipse,* Sebastopol, USA, O'Reilly.

Lloyd-Williams M. (1997), 'Exploiting Domain Knowledge During the Automated Design of Object-Oriented Databases', *Springer-Verlag Berlin, LNCS1331*, Berlin, pp. 16–27.

McCabe, T.J. (1976), 'A Complexity Measure', *IEEE Transactions on Software Engineering, VOL. SE2, No4,* pp. 308-320.

Noah, S. and Williams M. (2000), 'Exploring and Validating the Contributions of Real-World Knowledge to the Diagnostic Performance of Automated Database Design Tools', IEEE Computer Society Washington, USA, p177-189.

Nurminen, J.K. (2003), 'Using software complexity measures to analyze algorithms – an experiment with the shortest-paths algorithms'*, Computers & Operations Research 3,* Elsevier Science*,* Finland, pp. 1121-1134.

Pressman, R. (2000), *Software Engineering a practitioner's approach 5th edition*, Maidenhead, England, McGraw-Hill, pp. 494-519.

Queralt A., Teniente E. (2006) 'Reasoning on UML Class Diagram with OCL Constraints', , *Springer-Verlag Berlin, LNCS 4215*, Berlin, pp. 597-512–54.

Rosenthal, A. and Reiner, D. (1994), 'Tools and Transformation – Rigorous and Otherwise – for Practical Database Design', *ACM Transactions on Database Systems 19(2)*, pp. 167-211.

Salvatore, D. and Reagle D. (2002), *Schaum's Outline of Theory and Problems of Statistics and Econometrics 2nd Edition, USA,* McGraw-Hill, pp. 67-94.

Sedgewick, R. (2004), *Algorithm in Java Third Edition,* USA, Addison Wesley, pp. 1-31 and pp. 149-227.

Sharp J.A., Peters J. and Howard K. (2002), *The Management of a Student Research Project 3rd edition*, UK, Gower.

Siau, K., Wand, Y. and Benbasat, I. (1995), 'A psychological study on the use of relationship concept - Some preliminary findings', In *Proceedings of the 7th*

International Conference on Advanced Information Systems Engineering, Springer-Verlag, Vienna, Austria, pp. 341–354.

Spell, B. (2000), *Professional Java Programming,* Birmingham, UK, Wrox, pp. 6-20.

Stern, U. (2001), Java vs. C++, http://verify.stanford.edu/uli/java_cpp.html, [Accessed 20 October 2007].

Storey, V. (1993), 'Understanding Semantic Relationships', *Very Large Data Bases (VLDB) Journal, vol. 2, no. 4*, pp. 455-488.

Teorey, T.J., Yang, D. and Fry, J.P. (1986), 'A logical design methodology for relational databases using the extended entity-relationship model', *ACM Computing Surveys 18(2).*

The British Computer Society (2006), The Code of Conduct [on line], *http://www.bcs.org/server.php?show=conWebDoc.1588*, [Accessed 15 April 2007].

The Open University (2002), *M876 Relational Database System - Block 1 Information Systems*, Milton Keynes, Open University.

The Open University (2005), *M801 Research Project and Dissertation*, Milton Keynes, Open University.

The Open University (2000), *M878 Object-Oriented Software Development - Unit 3 Domain Modelling,* Milton Keynes, Open University.

The Open University (2000), *M878 Object-Oriented Software Development - Unit 5 Software Specification,* Milton Keynes, Open University.

Vatanawood W. and Rivepiboon W. (2004), 'Formal Specification Synthesis for Relational Database Model,' *International Journal of Intelligent Systems*, Vol. 19.

Wang L. (1996), 'Thorough Investigation into "An Improved Algorithm Based on Subset Closures for Synthesizing a Relational Database Scheme"', *IEEE Transactions On Software Engineering, VOL. 22, NO. 4*, pp271-274.

Yang C. C, Li G., Ann-Beng Ng P. (1988), 'An Improved Algorithm Based on Subset Closures for Synthesizing a Relational Database Scheme', *IEEE Transactions On Software Engineering, Vol. 14, NO. II,* pp1731-1738.

Bibliography

Appice, A., Ceci, M. and Rawles S., Flash P. W. (2004), 'Redundant Feature Elimination for Multi-Class Problems,' *proc. of the 21st International Conference on Machine Learning*, Banff, Canada.

Cook, M. (1982), 'Software Metrics: An Introduction to Annotated Bibliography', *Software Engineering Notes*, Vol7 N2, pp41-60.

Dar, S. and Ramakrishnan, R. (1994), 'A Performance Study of Transitive Closure Algorithms,' *Proc. ACM - SIGMOD – 94.*

Fowler, M. (2003), *Patterns of Enterprise Application Architecture*, Hoboken, USA, Addison-Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides J (2002), *Design Patterns Element of Reusable Object-Oriented Software*, USA, Addison-Wesley.

Lee, D. (2003), *Displaying a UML Diagram with Draw2D,* IBM, pp1-12.

Lipschutz, S. and Liposon, M. (1997), *Theory and Problems of Discrete Mathematics 2nd edition*, USA, McGraw-Hill.

Norman, D.A. (1988), *The Psychology of Everyday Things*, New York, Basic Books, p17.

The Open University (2001), M873 *User Interface Design and Evaluation Design*, Milton Keynes, Open University, Part 2 - Unit 3 p53.

Yen-Ting, C. and Ping-Yu, H. (2005), 'An Efficient and Grain Preservation Mapping Algorithm: From ER Diagram to Multidimensional Model', F.*F. Ramos et al. (Eds.): ISSADS 2005*, LNCS 3563, pp. 331–346.

Index

—A—

Aggregation *See* Relationship type

Algorithm 9, 10, 13, 14, 17, 21, 27, 39, 44, 46, 47, 50, 59, 60, 61, 63, 65, 67, 69, 70, 71, 73, 94, 95, 102, 106, 107, 114, 115, 118

Algorithm complexity 120

Algorithm component 69

Algorithm logic 15

Algorithm type 73

Algorithmic detection 19

Enhanced version of the Bowers algorithm 79

enhancedBowers 63, 67, 69, 71, 73, 97, 102, 107, 110, 112, 114

Antonym 29

Arc 22, *See* Relationship

Architecture 54

Association 2, *See* Relationship type

Attribute 2, 10, 21, 22, 26, 27, 28, 39, 40, 42

Attribute dependency 8

Composite Attributes 22

AttributeAttribute name 42, 51

—B—

Boolean logic 44

Business object 73

—C—

Cardinality 4, 29, 35, 45

Cardinality constraint 37, 45

Relationship cardinality 50

Relationship cardinality/participation 9

Cardinality Cardinality condition 51

CardinalityRelationship cardinality/participation 42

Class

Child classes 31

Class diagram 14, 69, 70

Parent class 31

Class hierarchy *See* Class diagram

Composition Theorem for Relationships that Share Common Attributes 40

Conceptual model 107

Constraint 3

Inclusion constraint 11, 40, 42, 51

Integrity constraint 1

Semantic connection constraint 34, 38, 50

Shared attributes and inclusion constraints 51

Covering

LR minimum covering 26

Minimum Covering 24, 28

Non redundant covering 22, 24

CPU time See Execution time

Cycle 11, 26, 29, 38, 40, 42, 46, 47, 50, 51, 52, 56, 79, 88, 93, 94, 98, 101, 107, 112, 115

Longer cycle 94

Multiple cycles 63

Shorter cycle 94

—D—

Data model 39

Database 3, 31

Database design 1, 28

Database designer 14

Hierarchical database 1

Object-oriented database 1

Relational database 1

Scientific database 19

Denormalisation 1

Design and Evaluation Workbench 39

Design pattern 54

Detection strategy 46

Dictionary 11, 28, 29, 33, 42, 50

Discovery of potential redundant relationship 70

Duplicate relationship 48, 50, 51, *See* Redundant Relationship

—E—

Eclipse platform 65

Eclipse development environment 53

Java Eclipse framework 12

Eclipse platformEclipse SDK 58

Edge 9, 11, 21, 22, 24, 26, 27, 46, *See* Entity

Element 73, 104, 118, 119

Entity 2, 3, 4, 7, 8, 9, 10, 20, 29, 35, 38, 39, 40, 42, 43, 44, 45, 46, 50, 51, 60, 63, 73, 74, 79, 90, 97, 102, 108, 109, 115

Parent entity 42

Entity relationship diagram 2

Equivalent composed path 61, 69, 74, 87, 88, 89, 90, 93, 96, 97, 98, 99, 108, 109, 111, 114

Equivalent path 88, 116, *See* Equivalent composed path

ERD 5, 7, 8, 9, 10, 13, 14, 17, 19, 20, 21, 27, 46, 61, 62, 63, 65, 66, 67, 69, 70, 73, 76, 79, 94, 95, 99, 102, 104, 112, 114, 116, 118, 119, *See* Entity relationship diagram

Execution time 95, 96, 102, 119

—F—

FD*See* Functional Dependency Graph

FD-graph 22, 24, *See* Functional Dependency Graph

FD-path *See* Functional Dependency Graph

Functional dependency 21, *See* Functional dependency graph

Functional Dependency graph 21

—G—

Graph 10, 21, 26, 27, 46, 50, 60, 63, 114

Connected graph 9

Dense graph 96, 99, 102, 114

Directed graph 26

Graph search algorithms 43

Graph theory 8

Graph type

Hyper graph 8

Minimum graph 27

Sparse graph 46, 96, 99, 102

Graph theory 14, 15, 19, 21, 51, 54

Growth rate 63, 102

—H—

Hamilton circuit 26

Heuristic 39

Heuristics 14, 34, 114

Horizontal shortcut 7, 40

—I—

Inheritance 28

Generalisation hierarchy 31

Intelligent Object Analyser 31

Interface 70, 71, 73

—J—

Java 11, 58

—K—

Key

Composite key 42

Foreign key 42

Primary key 42

Knowledge 31

'Real-world' knowledge 28, 31, 34, 42, 50

Domain knowledge 29, 31

—L—

Level of significance 61

Loop 11, 46, 47, 51, 52, 56, 65, 79, 82, 94, 112, 114, *See* Recursive Relationship

—M—

Matrix 70, 73, 74, 79, 80, 82, 84, 87, 89, 115, 119

Adjacency-matrix 10, 46, 53, 54, 66, 71, 73, 76, 87, 89, 96, 102, 115, 118, 119

Boolean adjacency-matrix 73

Boolean matrix 79, 87

Diagonal matrix 89

Initial Boolean matrix 87, 88, 108, 109, 118

Initial matrix 46, 73, 79, 82, 86, *See* Initial Boolean matrix

Matrix element 71, 73, 86, 114

Matrix multiplication 79

Polymorphic adjacency-matrix 73

Methodologies

Design modelling methodologies 2

Metric

Cyclomatic Number 120

Line of Code 120

Volume size 120

Minimal Covering of FD-graphs 112

Minimum path 27

Model 1, 31

Conceptual model 39

Conceptual view 33

Hierarchical model 39

Logical model 39

Model errors 33

Modelling

ERD Modelling 15

—N—

Node *See* Edge

Normal distribution 62

Normalisation 8

—O—

Object 70, 71, 73, 76

Occurence diagram 34

OCL 116

Original Boolean matrix *See* initial Boolean matrix

—P—

Participation 4, 38, 44, 50

Mandatory and optional participation 45

Optional (mandatory) participation20

Relationship participation 44

ParticipationParticipation constraint 35, 38

Partition 71, 73, 82, 84, 86, 104, 108, 110, 114, 118

Relationship partition 108

Path 10, 11, 22, 24, 27, 28, 44, 45, 46, 80, 82, 86, 87, 88, 89, 97, 98, 99, 108, 109, 110

‘Look back’ path 87

‘Recursive look back’ path 82, 87

Alternative path 5, 44

Composed Path 82

Composite path 44, 45

Composition path 44

Cyclic path 34

Equivalent composed path 10, 110, 111, 116

Shortest path 9, 50

Shortest Path 11

Path signature 46, *See* Signature

Plug-ins 12

Potential arc redundancy*See* Potential relationship redundancy

Potential redundancy *See* Potential redundant relationship

Potential redundant arc43, *See* Potential redundant relationship

Potential redundant path 53, 61, 88, 90, 98, 106, 107, 110, *See* Potential redundant relationship

Potential redundant relationship 7, 10, 13, 14, 15, 29, 33, 40, 42, 43, 47, 61, 66, 67, 69, 76, 88, 89, 95, 96, 97, 107, 108, 109, 111, 112, 114, 115

Problem domain 53

Processing time33, See Execution time

Prototype 13, 28, 66, 70, 95, 96

GEV2008 prototype 96

Graph Edge Validator prototype 96

Prototype, GEV2008 prototype 107

Pseudo code 90, 99

—R—

Record *See* Tuple

Recursion101, *See* Recursive relationship

Recursive 3, 101, *See* Recursive Relationship

Recursive path 87

Redundancy 34, *See* Redundant

Redundant 3, 5, 7, 8, 9, 10, 22, 29, 33, 37, 38, 46, 71, 87, 93

Redundant arc 8, 10, 47, *See* Redundant relationship

Redundant edge 27, *See* Redundant relationship

Redundant path *See* Redundant relationship

Redundant relationship 5, 10, 15, 34, 39

‘Unambiguous’ redundant relationship 35, 37, 38, 50

Redundant vertex *See* Redundant relationship

Reflexivity 27

Relation 2, *See* Entity

Relationship 2, 3, 4, 7, 8, 9, 10, 11, 14, 20, 21, 22, 27, 28, 29, 31, 32, 35, 38, 39, 40, 42, 44, 46, 48, 50, 60, 63, 65, 67, 73, 79, 86, 89, 93, 97, 101, 102, 107, 108, 109, 110, 115

‘Is-a’ relationship 28, 29

‘Many many many’ cycle 42, 51, 93, 106, 107, 115

‘Many many many’ relationship 52, *See* ‘Many many many’ cycle

‘Many-to-many’ relationship 4, 21

‘One-to-many’ relationship 4, 21, 37, 38

‘One-to-one’ relationship 37, 38

Binary relationship 4, 34, 45

Composing transitive relationship 43

Composite relationship 34

Composition of the relationship 40

Composition relationship 45

Crow’s foot 21

Multiple relationship 7, 11, 43, 51, 52, 56, 86, 101, 111, 115

'Part-of' relationship 29

Potential redundant relationship 99

Recursive relationship 43, 46, 82, 87, 94, 98, 102, 104, 108

Recursive Relationship 4

Relationship degree 4

Relationship name 9

Relationship transitivity 31

Relationship type 9, 97

Single relationship 38, 42, 50, 51, 97, 101

Single/multiple relationship 112

Ternary Relationship 4

Unary Relationship 4

Relationship diagram 65, *See* Entity relationship diagram

Relationship duplication 51, *See* Redundant relationship

Relationship redundancy 1, *See* Potential redundant relationship

Relationship semantic 10, 43, 50, *See* Semantic

Route *See* Path

—S—

Scalability 60, 63, 114

Scalable 14, 56, *See* Scalability

Schema 8, 63, 116

Physical schema 14

Relationial schema 8

Search engines 19

Searching algorithm *See* Algorithm

Semantic 3, 5, 9, 44

Semantic Relationship Analyzer28

Semantic relationship type 28

Semantic relationship*See* Relationship

SemanticSemantic comparison 42

Signature 10, 44, 45, 46, 87, 89

Equivalent path signature 74

Relationship signature 51

signature composition 87

Software architecture 15, 54

Strong component 26, 27

Subtype *See* Type

Super type *See* Type

Superfluous relationship, *See* Redundant relationship

Superfluous vertex 24, *See* Redundant relationship

Synonym 9, 33

Synonym relationship 9, *See* Redundant relationship

Synonymous concept 50

Synonymous relationship *See* Redundant relationship

—T—

Termination condition *See* Transitive closure

Termination order *See* Transitive closure

Test case 15, 60, 96, 97, 98, 114, 115

Test schema 59

Thesaurus approach 29

Third Normal Form*See* Normal Form

Tool See Prototype

Transitive closure 26, 47, 94, 96

Transitive dependency 1

Transitive reduction 26

Transitive relationship 44

Transitivity 27

Tuple 2, 40

—U—

UML 14, 67

  UML class diagram 47

  UML notation

Unambiguous redundant relationship *See* Redundant relationship

Union 27

Unnecessary relationship 14, *See* Redundant relationship

Unwanted path 103, *See* Redundant Relationship

Use case 60, 96

User 28, 67, 94

  Business user 8

User interface 39

—V—

Venn diagram 48

Vertex 9, 10, 11, 21, 22, 24, 26, 27, *See* Relationship

Vertical shortcut 7, 10, 40

—W—

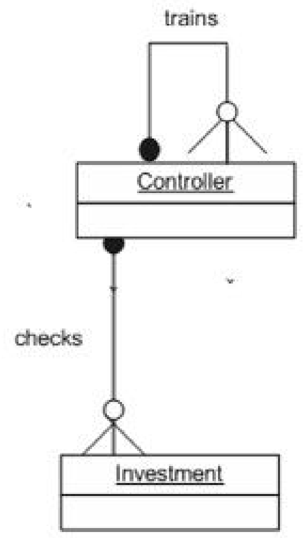
Weighted graph *See* Graph

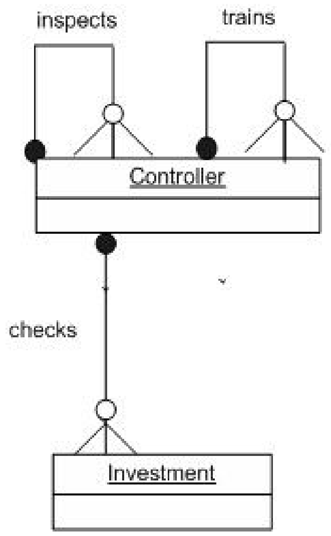
—X—

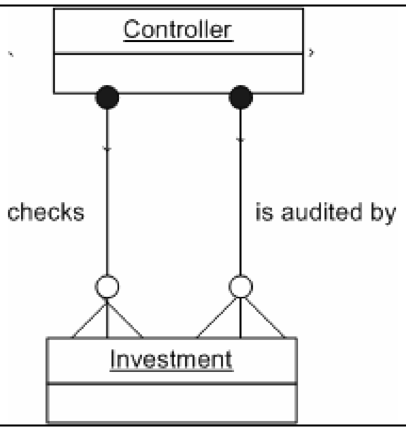
XML 13, 69, 76

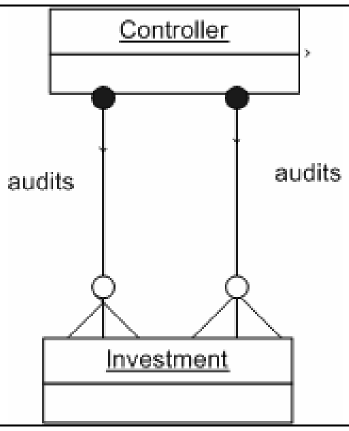
  XML file 118

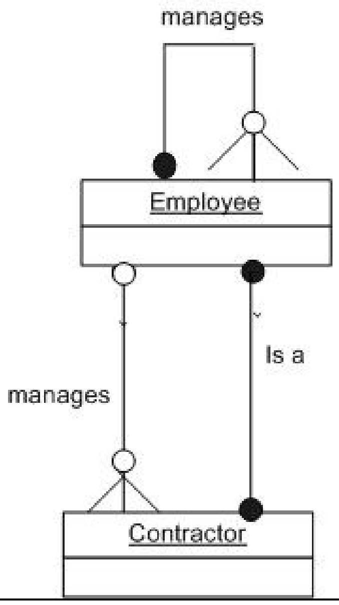
Appendix A – Quality Discovery Test Cases

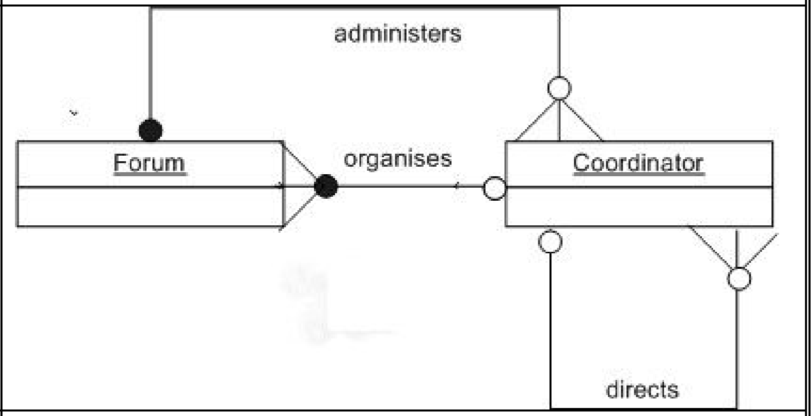
Test Case Details	
Name	Test A
File Name	TestA.xml
Input Schema	
Total Number of Relationships	2
Types	Single/Multiple Relationship (S/M)– S Recursion – Yes Cycle – No

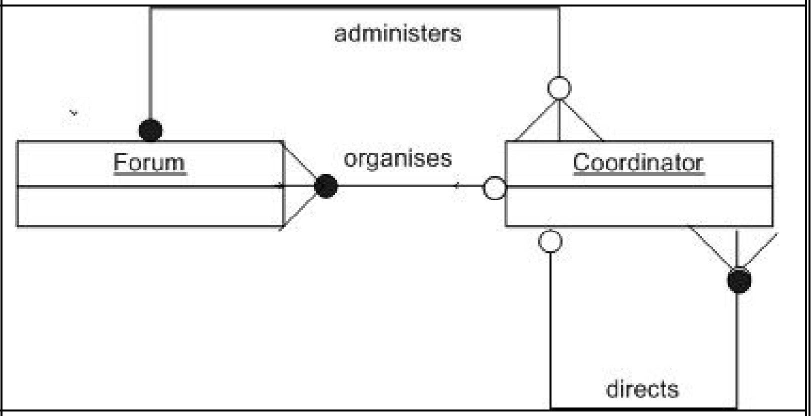
Test Case Details	
Name	Test B
File Name	TestB.xml
Input Schema	
Total Number of Relationships	3
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

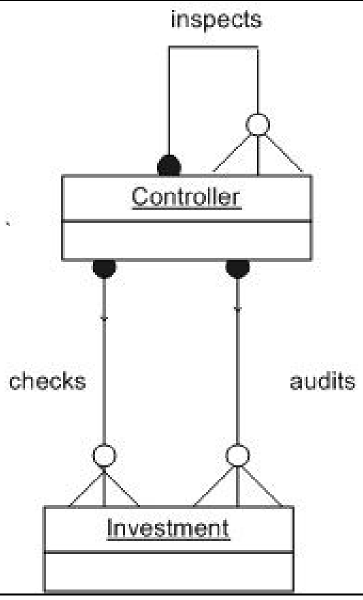
Test Case Details	
Name	Test C
File Name	TestC.xml
Input Schema	
Total Number of Relationships	2
Types	Single/Multiple Relationship (S/M)– M Recursion – No Cycle – Yes

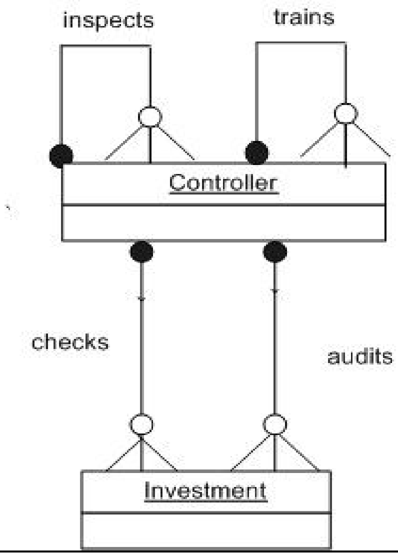
Test Case Details	
Name	Test D
File Name	TestD.xml
Input Schema	
Total Number of Relationships	2
Types	Single/Multiple Relationship (S/M)– M Recursion – No Cycle – Yes

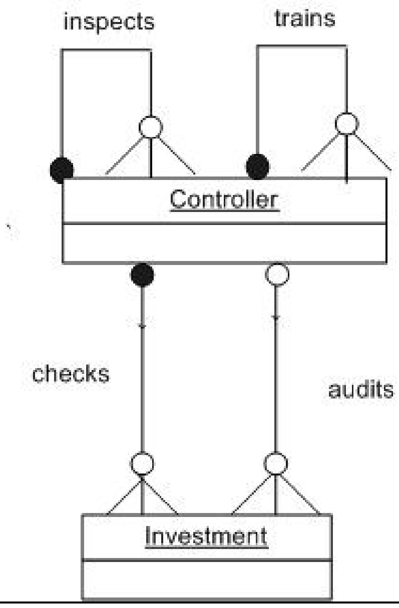
Test Case Details	
Name	Test E
File Name	TestE.xml
Input Schema	
Total Number of Relationships	3
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

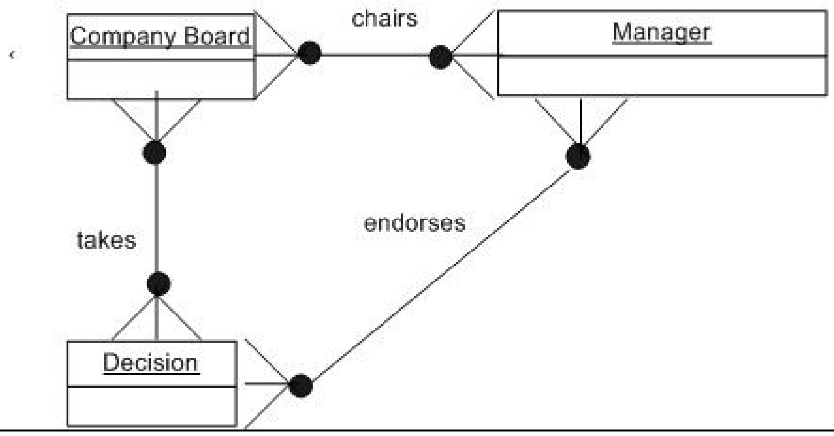
Test Case Details	
Name	Test F
File Name	TestF.xml
Input Schema	
Total Number of Relationships	3
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

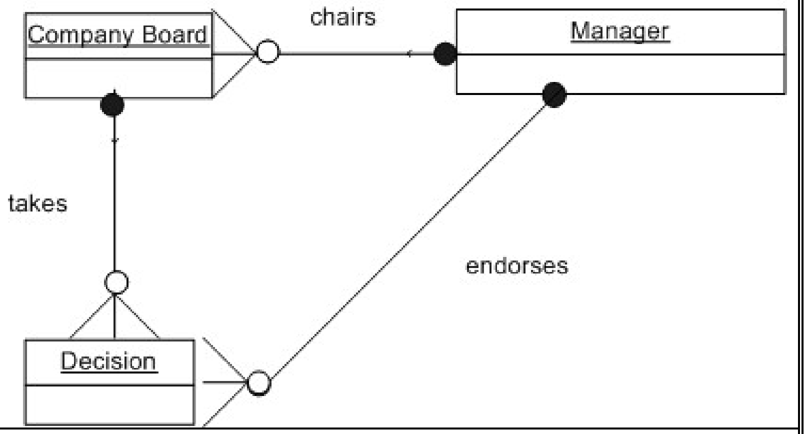
Test Case Details	
Name	Test G
File Name	TestG.xml
Input Schema	
Total Number of Relationships	3
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

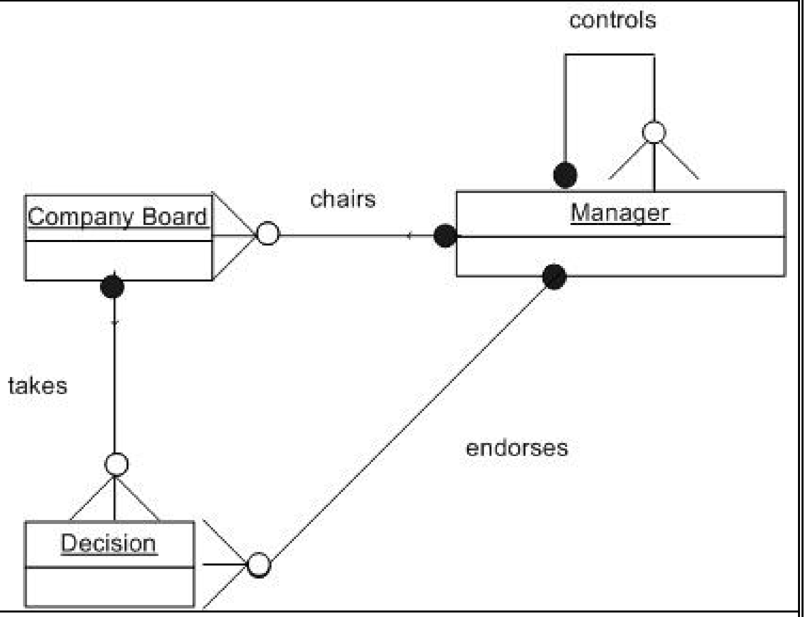
Test Case Details	
Name	Test H
File Name	TestH.xml
Input Schema	
Total Number of Relationships	3
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

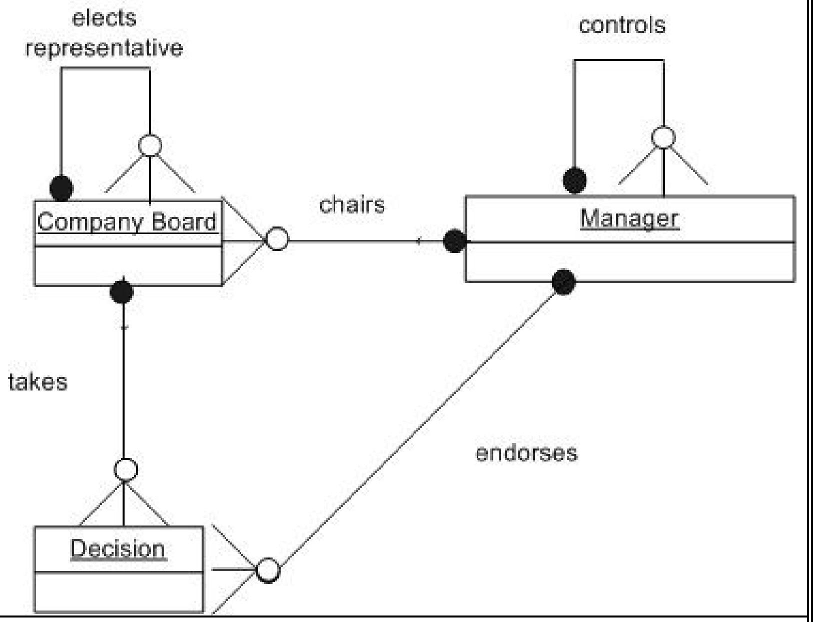
Test Case Details	
Name	Test I
File Name	TestI.xml
Input Schema	 inspects trains Controller checks audits Investment
Total Number of Relationships	4
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

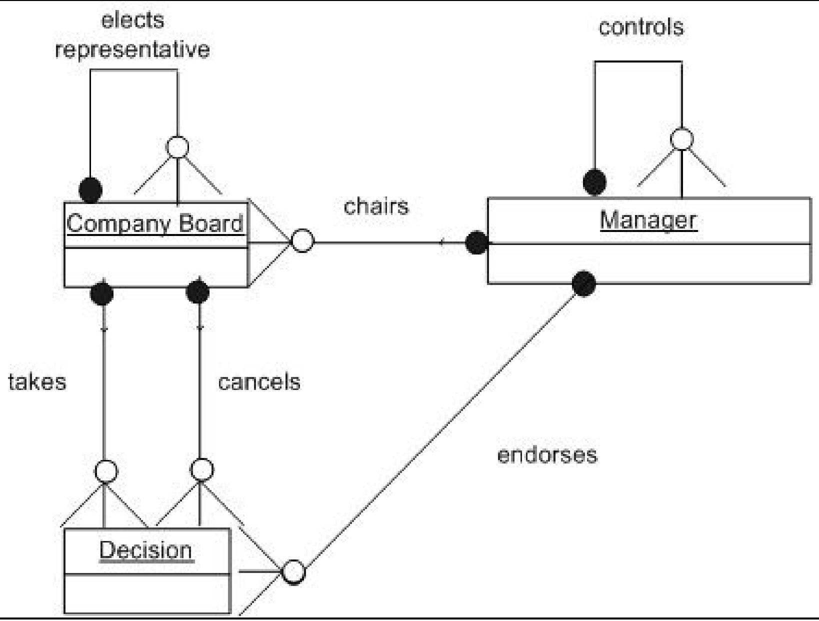
Name	Test J
File Name	TestJ.xml
Input Schema	 inspects trains Controller checks audits Investment
Total Number of Relationships	4
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

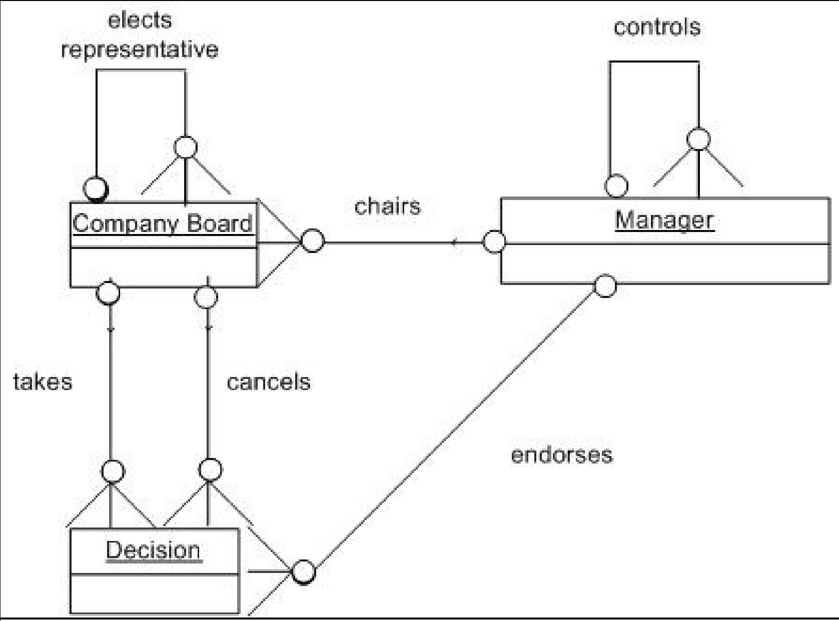
Test Case Details	
Name	Test K
File Name	TestK.xml
Input Schema	 Company Board chairs Manager takes endorses Decision
Total Number of Relationships	3
Types	Single/Multiple Relationship (S/M)– S Recursion – No Cycle – Yes

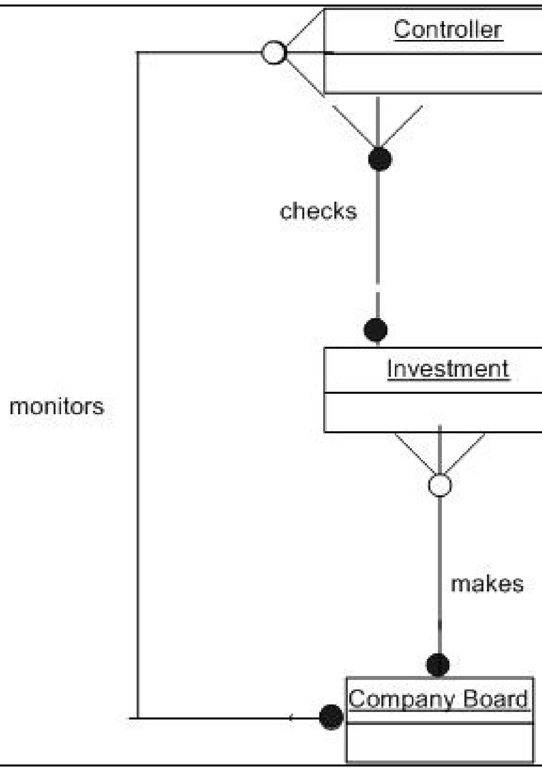
Test Case Details	
Name	Test L
File Name	TestL.xml
Input Schema	 Company Board chairs Manager takes endorses Decision
Total Number of Relationships	3
Types	Single/Multiple Relationship (S/M)– M Recursion – No Cycle – Yes

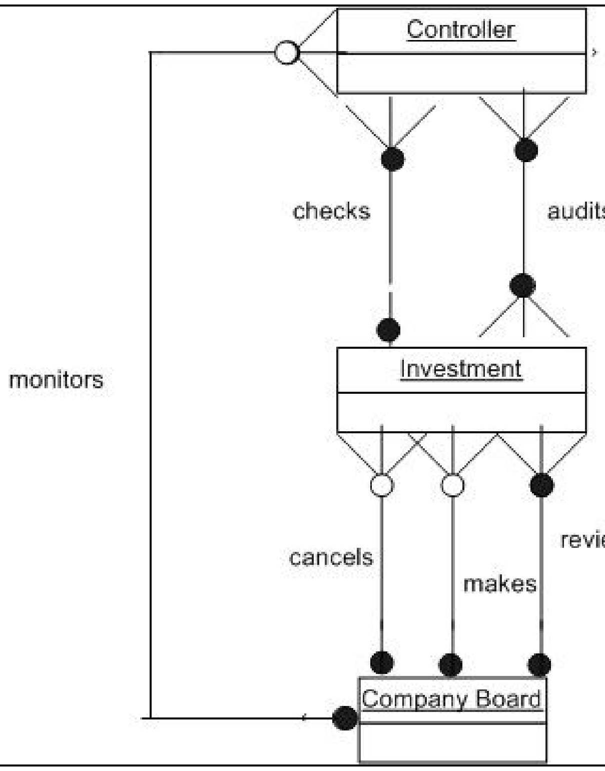
Test Case Details	
Name	Test M
File Name	TestM.xml
Input Schema	 controls Company Board chairs Manager takes endorses Decision
Total Number of Relationships	4
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

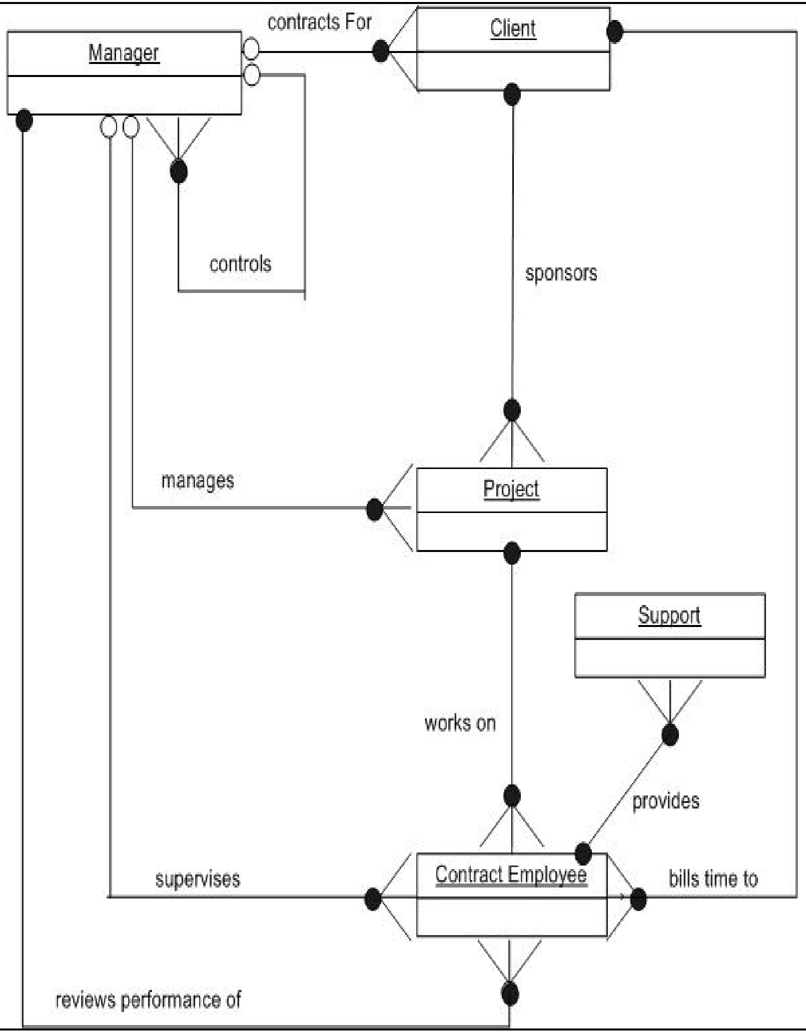
Test Case Details	
Name	Test N
File Name	TestN.xml
Input Schema	 elects representative controls Company Board chairs Manager takes endorses Decision
Total Number of Relationships	5
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

Test Case Details	
Name	Test O
File Name	TestO.xml
Input Schema	 elects representative controls Company Board chairs Manager takes cancels endorses Decision
Total Number of Relationships	5
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

Test Case Details	
Name	Test P
File Name	TestP.xml
Input Schema	 elects representative controls Company Board chairs Manager takes cancels endorses Decision
Total Number of Relationships	5
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

Test Case Details	
Name	Test Q
File Name	TestQ.xml
Input Schema	 Controller checks Investment monitors makes Company Board
Total Number of Relationships	3
Types	Single/Multiple Relationship (S/M)– S Recursion – No Cycle – Yes

Test Case Details	
Name	Test R
File Name	TestR.xml
Input Schema	 Controller checks audits monitors Investment cancels makes reviews Company Board
Total Number of Relationships	6
Types	Single/Multiple Relationship (S/M)– S Recursion – No Cycle – Yes

Test Case Details	
Name	Test S
File Name	TestS.xml
Input Schema	 Manager Client contracts For controls sponsors Project manages Support works on provides Contract Employee supervises bills time to reviews performance of
Total Number of Relationships	9
Types	Single/Multiple Relationship (S/M)– M Recursion – Yes Cycle – Yes

Appendix B – Quality Discovery Test Cases Input File

This section lists the ERD XML schema storage test cases. An ERD is made of a list of entities (<entities>) and relationships (<relationships>).
An entity (<entity>) has:

- a name (<entity name = “”>), and
- contains a number of attributes (<attribute name =””)

A relationship has a name (<relationship name =””>). It contains a description of the optionality and cardinality for both sides of the relationship. This information is stored in the <relationshipEnd> element. It indicates:

- the name of the source (destination/destination) entity,
- the direction (source entity = “from”, destination entity =”to”),
- the optionality (“0” means optional and “1” means mandatory), and
- the cardinality (degree =”0” means many, degree =”1” means one)

The below XML snippet represent an example of the xml schema storage representation.

Only ‘TestA’ is provided below. For the full test detail list, please refer to the list of XML files from ‘TestA.xml’ to ‘TestS.xml’ in CD-ROM path:

…\Documents and Settings\All Users\Start Menu\Programs\a_Thesis\Eclipse\Eclipse SDK\eclipse-SDK-3.3M4-win32\eclipse-SDK-3.3M4-win32\eclipse\MyFiles\MiniCases\

```
<?XML VERSION="1.0" ENCODING="UTF-8"?>
<DIAGRAM NAME = "TESTA" TYPE ="ERD">
        <ENTITIES>
                <ENTITY NAME="CONTROLLER" POSITION = "0">
                        <ATTRIBUTE NAME = "ID" TYPE="INT"/>
                </ENTITY>
                <ENTITY NAME="INVESTMENT" POSITION = "1">
                        <ATTRIBUTE NAME = "ID" TYPE="INT"/>
                </ENTITY>
        </ENTITIES>
        <RELATIONSHIPS>
                <RELATIONSHIP NAME = "CHECKS">
                        <RELATIONSHIPEND ENTITY="CONTROLLER"
        DIRECTION="FROM"    OPTIONALITY="1" DEGREE="1"/>
                        <RELATIONSHIPEND ENTITY="INVESTMENT"
        DIRECTION="TO"                  OPTIONALITY="0" DEGREE="0"/>
                </RELATIONSHIP>
                <RELATIONSHIP NAME = "TRAINS">
                        <RELATIONSHIPEND ENTITY="CONTROLLER"
        DIRECTION="FROM"    OPTIONALITY="1" DEGREE="1"/>
                        <RELATIONSHIPEND ENTITY="CONTROLLER"
        DIRECTION="TO"        OPTIONALITY="0" DEGREE="0"/>
                </RELATIONSHIP>
        </RELATIONSHIPS>
</DIAGRAM>
```

Appendix C – The Quality Discovery Collected Raw Data

Experience Results Vs Expected Results (Test A)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	None
Correct Potential Redundant Paths Discovery	
# Correct Equivalent Paths Discovery	None
Correct Equivalent Paths Discovery	
# Incorrect Potential Redundant Path Discovery	0
Incorrect Potential Redundant Path Discovery	
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test B)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	0
Correct Potential Redundant Paths Discovery	
# Correct Equivalent Paths Discovery	0
Correct Equivalent Paths Discovery	
# Incorrect Potential Redundant Path Discovery	2
Incorrect Potential Redundant Path Discovery	Controller – inspects – Controller Controller – trains – Controller
# Incorrect Equivalent Path Discovery	2
Incorrect Equivalent Path Discovery	Controller-trains-Controller-checks-Controller Controller-inspects-Controller-checks-Controller
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test C)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	None
Correct Potential Redundant Paths Discovery	
# Correct Equivalent Paths Discovery	None
Correct Equivalent Paths Discovery	
# Incorrect Potential Redundant Path Discovery	0
Incorrect Potential Redundant Path Discovery	
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test D)	
User Interface Output	
Order 2	
Order	2
# Correct Potential Redundant Paths Discovery	None
Correct Potential Redundant Paths Discovery	
# Correct Equivalent Paths Discovery	None
Correct Equivalent Paths Discovery	
# Incorrect Potential Redundant Path Discovery	0
Incorrect Potential Redundant Path Discovery	
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	1
Missed Potential Redundant Path Discovery	Audits
# Missed Equivalent Path Discovery	1
Missed Equivalent Path Discovery	Audits

Experience Results Vs Expected Results (Test E)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	0
Correct Potential Redundant Paths Discovery	0
# Correct Equivalent Paths Discovery	0
Correct Equivalent Paths Discovery	0
# Incorrect Potential Redundant Path Discovery	1
Incorrect Potential Redundant Path Discovery	Employee – manages - Contractor
# Incorrect Equivalent Path Discovery	1
Incorrect Equivalent Path Discovery	Employee –manages – Employee and Contractor – is – Employee
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test F)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	None
Correct Potential Redundant Paths Discovery	None
# Correct Equivalent Paths Discovery	0
Correct Equivalent Paths Discovery	
# Incorrect Potential Redundant Path Discovery	0
Incorrect Potential Redundant Path Discovery	
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test G)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	1
Correct Potential Redundant Paths Discovery	Coordinator – organises –Forum
# Correct Equivalent Paths Discovery	1
Correct Equivalent Paths Discovery	Coordinator – directs – Coordinator and Coordinator – administers – Forums
# Incorrect Potential Redundant Path Discovery	0
Incorrect Potential Redundant Path Discovery	
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test H)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	2
Correct Potential Redundant Paths Discovery	Controller – audits – Investment Controller – checks – Investment
# Correct Equivalent Paths Discovery	2
Correct Equivalent Paths Discovery	Controller – inspects – Controller and Controller – checks – Controller Controller – inspects – Controller and Controller – audits – Controller
# Incorrect Potential Redundant Path Discovery	0
Incorrect Potential Redundant Path Discovery	
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test I)		
User Interface Output		
Order	2	3
# Correct Potential Redundant Paths Discovery	2	0
Correct Potential Redundant Paths Discovery	Controller – audits – Investment Controller – checks – Investment	
# Correct Equivalent Paths Discovery	2	0
Correct Equivalent Paths Discovery	Controller – inspects – Controller and Controller – checks – Controller Controller – inspects – Controller and Controller – audits – Controller	
# Incorrect Potential Redundant Path Discovery	2	1
Incorrect Potential Redundant Path Discovery	Controller – trains – Controller Controller – inspects – Controller	Controller – inspects – Controller
# Incorrect Equivalent Path Discovery	2	1
Incorrect Equivalent Path Discovery	Controller – inspects – Controller and Controller – inspects – Controller Controller – inspects – Controller and Controller – trains – Controller	Controller – inspects – Controller and Controller – trains – Controller and Controller – trains – Controller
# Missed Potential Redundant Path Discovery	0	0
Missed Potential Redundant Path Discovery		
# Missed Equivalent Path Discovery	2	0
Missed Equivalent Path Discovery	Controller – trains – Controller and Controller – checks – Controller Controller – trains – Controller and Controller – audits – Controller	

Experience Results Vs Expected Results (Test J)		
User Interface Output		
Order	2	3
# Correct Potential Redundant Paths Discovery	0	0
Correct Potential Redundant Paths Discovery		
# Correct Equivalent Paths Discovery	0	0
Correct Equivalent Paths Discovery		
# Incorrect Potential Redundant Path Discovery	3	1
Incorrect Potential Redundant Path Discovery	Controller – trains– Controller Controller – inspects– Controller Controller – checks– Investment	Controller – inspects– Controller
# Incorrect Equivalent Path Discovery	3	1
Incorrect Equivalent Path Discovery	Controller – inspects– Controller and Controller – inspects– Controller Controller – inspects– Controller and Controller – trains– Controller Controller – inspects– Controller and Controller – audits – Investment	Controller – inspects– Controller and Controller – trains– Controller and Controller – inspects– Controller and Controller – trains– Controller
# Missed Potential Redundant Path Discovery	0	
Missed Potential Redundant Path Discovery		
# Missed Equivalent Path Discovery	0	
Missed Equivalent Path Discovery		

Experience Results Vs Expected Results (Test K)	
Order	2
# Correct Potential Redundant Paths Discovery	0
Correct Potential Redundant Paths Discovery	
# Correct Equivalent Paths Discovery	0
Correct Equivalent Paths Discovery	
# Incorrect Potential Redundant Path Discovery	3
Incorrect Potential Redundant Path Discovery	Manager – chairs – CompanyBoard Manager – chairs – CompanyBoard Manager - endorses - Decision
# Incorrect Equivalent Path Discovery	3
Incorrect Equivalent Path Discovery	Manager – endorses – Decision and CompanyBoard – takes – Decision CompanyBoard – takes – Decision and Manager – endorses – Decision Manager – chairs – CompanyBoard and CompanyBoard – takes – Decision
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test L)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	1
Correct Potential Redundant Paths Discovery	Manager - endorses - Decision
# Correct Equivalent Paths Discovery	1
Correct Equivalent Paths Discovery	Manager – chairs – CompanyBoard and CompanyBoard – takes – Decision
# Incorrect Potential Redundant Path Discovery	0
Incorrect Potential Redundant Path Discovery	
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test M)		
User Interface Output		
Order	2	3
# Correct Potential Redundant Paths Discovery	1	1
Correct Potential Redundant Paths Discovery	Manager – endorses – Decision	Manager– endorses – Decision
# Correct Equivalent Paths Discovery	1	1
Correct Equivalent Paths Discovery	Manager – chairs – CompanyBoard and CompanyBoard – takes – Decision	Manager – controls – Manager and Manager – chairs – CompanyBoard and CompanyBoard – takes – Decision
# Incorrect Potential Redundant Path Discovery	0	0
Incorrect Potential Redundant Path Discovery		
# Incorrect Equivalent Path Discovery	0	0
Incorrect Equivalent Path Discovery		
# Missed Potential Redundant Path Discovery	0	0
Missed Potential Redundant Path Discovery		
# Missed Equivalent Path Discovery	0	0
Missed Equivalent Path Discovery		

Experience Results Vs Expected Results (Test N)			
User Interface Output			
Order	2	3	4
# Correct Potential Redundant Paths Discovery	1	1	1
Correct Potential Redundant Paths Discovery	Manager-endorses-Decision	Manager-endorses-Decision	Manager-endorses-Decision
# Correct Equivalent Paths Discovery	1	2	1
Correct Equivalent Paths Discovery	Manager-chairs-CompanyBoard and CompanyBoard – takes-Decision	Manager-controls-Manager and Manager-chairs-CompanyBoard and CompanyBoard-takes-Decision Manager-chairs-CompanyBoard and CompanyBoard-elects representative – CompanyBoard and CompanyBoard – takes-Decision	Manager-controls-Manager and Manager-chairs-CompanyBoard and CompanyBoard-elects representative – CompanyBoard and CompanyBoard-takes-Decision
# Incorrect Potential Redundant Path Discovery	1	0	0
Incorrect Potential Redundant Path Discovery	Manager-chairs-CompanyBoard		
# Incorrect Equivalent Path Discovery	1	0	0
Incorrect Equivalent Path Discovery	Manager-chairs-CompanyBoard and CompanyBoard – elects representative – CompanyBoard		
# Missed Potential Redundant Path Discovery	0	0	0
Missed Potential Redundant Path Discovery			
# Missed Equivalent Path Discovery	0	0	0
Missed Equivalent Path Discovery			

Experience Results Vs Expected Results (Test O)			
User Interface Output			
Order	2	3	4
# Correct Potential Redundant Paths Discovery	1	1	1
Correct Potential Redundant Paths Discovery	Manager-endorses-Decision	Manager-endorses-Decision	Manager-endorses-Decision
# Correct Equivalent Paths Discovery	2	4	2
Correct Equivalent Paths Discovery	Manager-chairs-CompanyBoard and CompanyBoard –takes-Decision Manager-chairs-CompanyBoard and CompanyBoard –cancels-Decision	Manager-controls-Manager and Manager-chairs-CompanyBoard and CompanyBoard –takes –Decision Manager-controls-Manager and Manager-chairs-CompanyBoard and CompanyBoard –cancels –Decision Manager-chairs-CompanyBoard and CompanyBoard –elects representative – CompanyBoard and CompanyBoard –takes –Decision Manager-chairs-CompanyBoard and CompanyBoard –elects representative – CompanyBoard and CompanyBoard –cancels –Decision	Manager-controls-Manager and Manager-chairs-CompanyBoard and CompanyBoard –elects representative – CompanyBoard and CompanyBoard –takes –Decision Manager-controls-Manager and Manager-chairs-CompanyBoard and CompanyBoard –elects representative – CompanyBoard and CompanyBoard –cancels –Decision
# Incorrect Potential Redundant Path Discovery	1	0	0
Incorrect Potential Redundant Path Discovery	Manager-chairs-CompanyBoard		
# Incorrect Equivalent Path Discovery	1	0	0
Incorrect Equivalent	Manager-chairs-		

Path Discovery	CompanyBoard and CompanyBoard - elects representative – CompanyBoard		
# Missed Potential Redundant Path Discovery	0	0	0
Missed Potential Redundant Path Discovery			
# Missed Equivalent Path Discovery	0	0	0
Missed Equivalent Path Discovery			

Experience Results Vs Expected Results (Test Q)	
User Interface Output	
Order	2
# Correct Potential Redundant Paths Discovery	1
Correct Potential Redundant Paths Discovery	Controller – monitors – CompanyBoard
# Correct Equivalent Paths Discovery	1
Correct Equivalent Paths Discovery	Controller – checks – Investment and CompanyBoard – makes – Investment
# Incorrect Potential Redundant Path Discovery	0
Incorrect Potential Redundant Path Discovery	
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test R)	
User Interface Output *Order 2*	
Order	2
# Correct Potential Redundant Paths Discovery	1
Correct Potential Redundant Paths Discovery	Controller – monitors – CompanyBoard
# Correct Equivalent Paths Discovery	2
Correct Equivalent Paths Discovery	Controller – checks – Investment and CompanyBoard – makes – Investment Controller – checks – Investment and CompanyBoard – cancels – Investment
# Incorrect Potential Redundant Path Discovery	1
Incorrect Potential Redundant Path Discovery	Controller – checks – Investment and CompanyBoard – reviews – Investment
# Incorrect Equivalent Path Discovery	0
Incorrect Equivalent Path Discovery	
# Missed Potential Redundant Path Discovery	0
Missed Potential Redundant Path Discovery	
# Missed Equivalent Path Discovery	0
Missed Equivalent Path Discovery	

Experience Results Vs Expected Results (Test S)			
User Interface Output			
Order	2	3	4
# Correct Potential Redundant Paths Discovery	2	2	1
Correct Potential Redundant Paths Discovery	Manager-manages-Project Manager-supervises-ContractEmployee	Manager-manages-Project Manager-supervises-ContractEmployee	Manager-supervises-ContractEmployee
# Correct Equivalent Paths Discovery	4	4	1
Correct Equivalent Paths Discovery	Manager –contracts for –Client and Client –sponsors –Project Manager –controls - Manager and Manager-reviewPerformanceOf-ContractEmployee Manager –contracts for –Client and ContractEmployee –billsTimeTo -Client Manager –manages for –Project and ContractEmployee –works - Project	Manager – controls - Manager and Manager –contracts for –Client and Client –sponsors –Project Manager – controls - Manager and Manager –contracts for –Client and ContractEmployee –billsTimeTo –Client Manager – controls - Manager and Manager –manages for –Project and ContractEmployee –worksOn - Project Manager –contracts for –Client and Client –sponsors –Project and ContractEmployee –worksOn - Project	Manager – controls - Manager and Manager –contracts for –Client and Client –sponsors –Project and ContractEmployee – worksOn - Project
# Incorrect Potential Redundant Path Discovery	0	0	0
Incorrect Potential Redundant			

Path Discovery			
# Incorrect Equivalent Path Discovery	0	0	0
Incorrect Equivalent Path Discovery			
# Missed Potential Redundant Path Discovery	1	0	0
Missed Potential Redundant Path Discovery	ContractEmployee-billsTimeTo-Client		
# Missed Equivalent Path Discovery	1	0	0
Missed Equivalent Path Discovery	Client-sponsors-Project and ContractEmployee-worksOn-Project		

Appendix D – The Quality Discovery Raw Data Transformation

														Potential Redundant Path			Equivalent Composed Path		
Id	Test Name	Order	Relationship Type (S)ingle/(M)ultiple	Recursion (Y)es/(N)o	Cycle (Y)es/(N)o	Total # Relationships	# Correct Potential Redundant Paths Discovery	# Incorrect Potential Redundant Path Discovery	# Missed Potential Redundant Path Discovery	# Correct Equivalent Paths Discovery	# Incorrect Equivalent Path Discovery	# Missed Equivalent Path Discovery		Correctness Metric (%)	Incorrectness Metric (%)	Missed Metric (%)	Correctness Metric (%)	Incorrectnes s Metric (%)	Missed Metric (%)
1	Test A	2	S	Y	N	2	N/A	0	0	N/A	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
2	Test B	2	M	Y	Y	3	0	2	0	0	2	0		0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
3	Test C	2	M	N	Y	2	N/A	0	0	N/A	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
4	Test D	2	M	N	Y	2	N/A	0	0	N/A	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
5	Test E	2	M	Y	Y	3	0	1	0	0	1	0		0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
6	Test F	2	M	Y	Y	2	N/A	0	0	N/A	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
7	Test G	2	M	Y	Y	2	1	0	0	1	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
8	Test H	2	M	Y	Y	2	2	0	0	2	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
9	Test I	2	M	Y	Y	4	2	2	0	2	2	2		50.00%	50.00%	0.00%	33.33%	33.33%	33.33%
10	Test I	3	M	Y	Y	4	0	1	0	0	1	0		0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
11	Test J	2	M	Y	Y	4	0	3	0	0	3	0		0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
12	Test J	3	M	Y	Y	4	0	1	0	0	1	0		0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
13	Test K	2	S	N	Y	2	0	3	0	0	3	0		0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
14	Test L	2	S	N	Y	2	1	0	0	1	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
15	Test M	2	M	Y	Y	4	1	0	0	1	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
16	Test M	3	M	Y	Y	4	1	0	0	1	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
17	Test N	2	S	Y	Y	5	1	1	0	1	1	0		50.00%	50.00%	0.00%	50.00%	50.00%	0.00%
18	Test N	3	S	Y	Y	5	1	0	0	2	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
19	Test N	4	S	Y	Y	5	1	0	0	1	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
20	Test O	2	M	Y	Y	6	1	1	0	2	1	0		50.00%	50.00%	0.00%	66.67%	33.33%	0.00%
21	Test O	3	M	Y	Y	6	1	0	0	4	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
22	Test O	4	M	Y	Y	6	1	0	0	2	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
23	Test P	2										0		0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
24	Test Q	2	S	N	Y	2	1	0	0	1	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
25	Test R	2	S	N	Y	2	1	0	0	2	1	0		100.00%	0.00%	0.00%	66.67%	33.33%	0.00%
26	Test S	2	M	Y	Y	9	2	0	1	4	0	1		66.67%	0.00%	33.33%	80.00%	0.00%	20.00%
27	Test S	3	M	Y	Y	9	2	0	0	4	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
28	Test S	4	M	Y	Y	9	1	0	0	1	0	0		100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
	# means 'number of'																		
	N/A means 'Not Available'																		

Appendix E – The Execution Time Test Cases

The full test detail list can be found on the CD-ROM, at the following location:

Sparse ERDs test cases (no loops)

Files:
TestETA.xml
TestETB.xml
TestETC.xml
TestETD.xml
TestETE.xml

Path:

…\Documents and Settings\All Users\Start Menu\Programs\a_Thesis\Eclipse\Eclipse SDK\eclipse-SDK-3.3M4-win32\eclipse-SDK-3.3M4-win32\eclipse\MyFiles\MiniCases\ExecTime\Sparse

Sparse ERDs test cases (with loops)

Files:
TestETBRecursive.xml
TestETCRecursive.xml

Path:

…\Documents and Settings\All Users\Start Menu\Programs\a_Thesis\Eclipse\Eclipse SDK\eclipse-SDK-3.3M4-win32\eclipse-SDK-3.3M4-win32\eclipse\MyFiles\MiniCases\ExecTime\Sparse

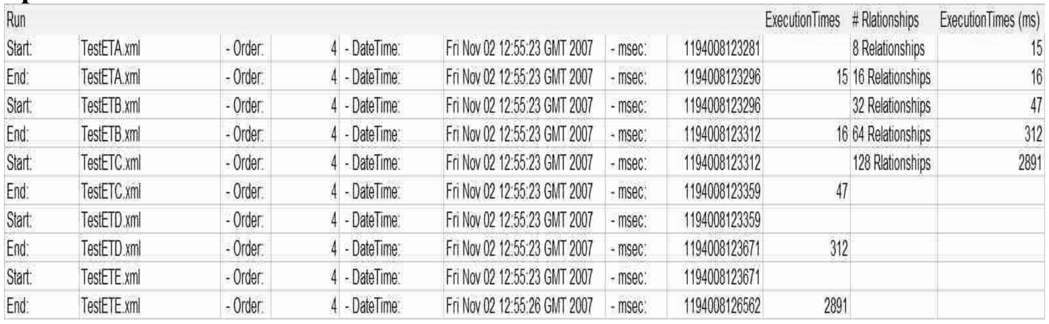
Dense ERDs test cases (no loops)

TestETA.xml
TestETB.xml
TestETC.xml
TestETD.xml
TestETE.xml
TestETF.xml
TestETF100.xml
TestETF120.xml

…\Documents and Settings\All Users\Start Menu\Programs\a_Thesis\Eclipse\Eclipse SDK\eclipse-SDK-3.3M4-win32\eclipse-SDK-3.3M4-win32\eclipse\MyFiles\

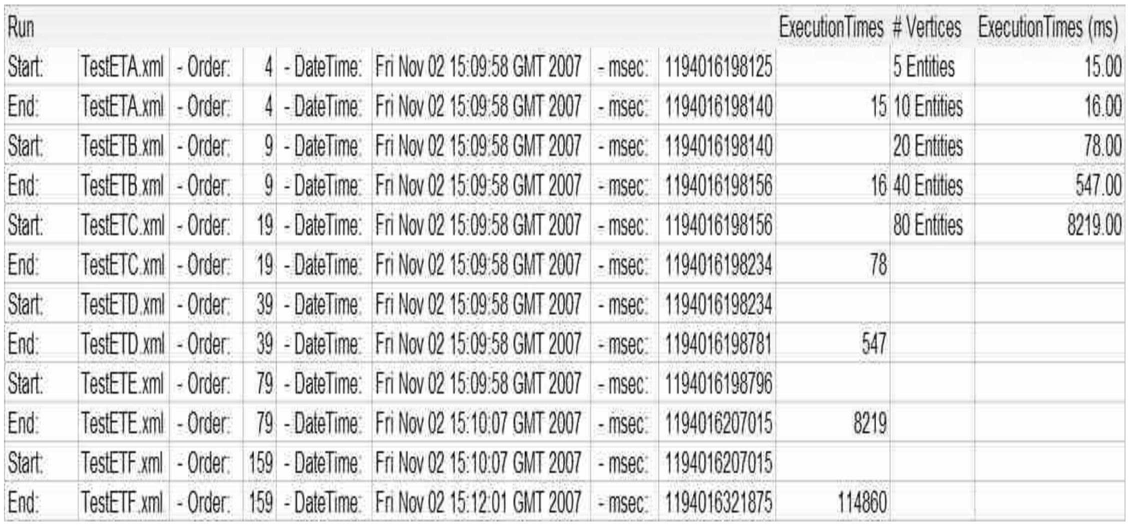
Appendix F – The Execution Time Results

Sparse ERD



Run								ExecutionTimes	# Rlationships	ExecutionTimes (ms)
Start:	TestETA.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123281		8 Relationships	15
End:	TestETA.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123296	15	16 Relationships	16
Start:	TestETB.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123296		32 Relationships	47
End:	TestETB.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123312	16	64 Relationships	312
Start:	TestETC.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123312		128 Rlationships	2891
End:	TestETC.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123359	47		
Start:	TestETD.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123359			
End:	TestETD.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123671	312		
Start:	TestETE.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:23 GMT 2007	- msec:	1194008123671			
End:	TestETE.xml	- Order:	4	- DateTime:	Fri Nov 02 12:55:26 GMT 2007	- msec:	1194008126562	2891		

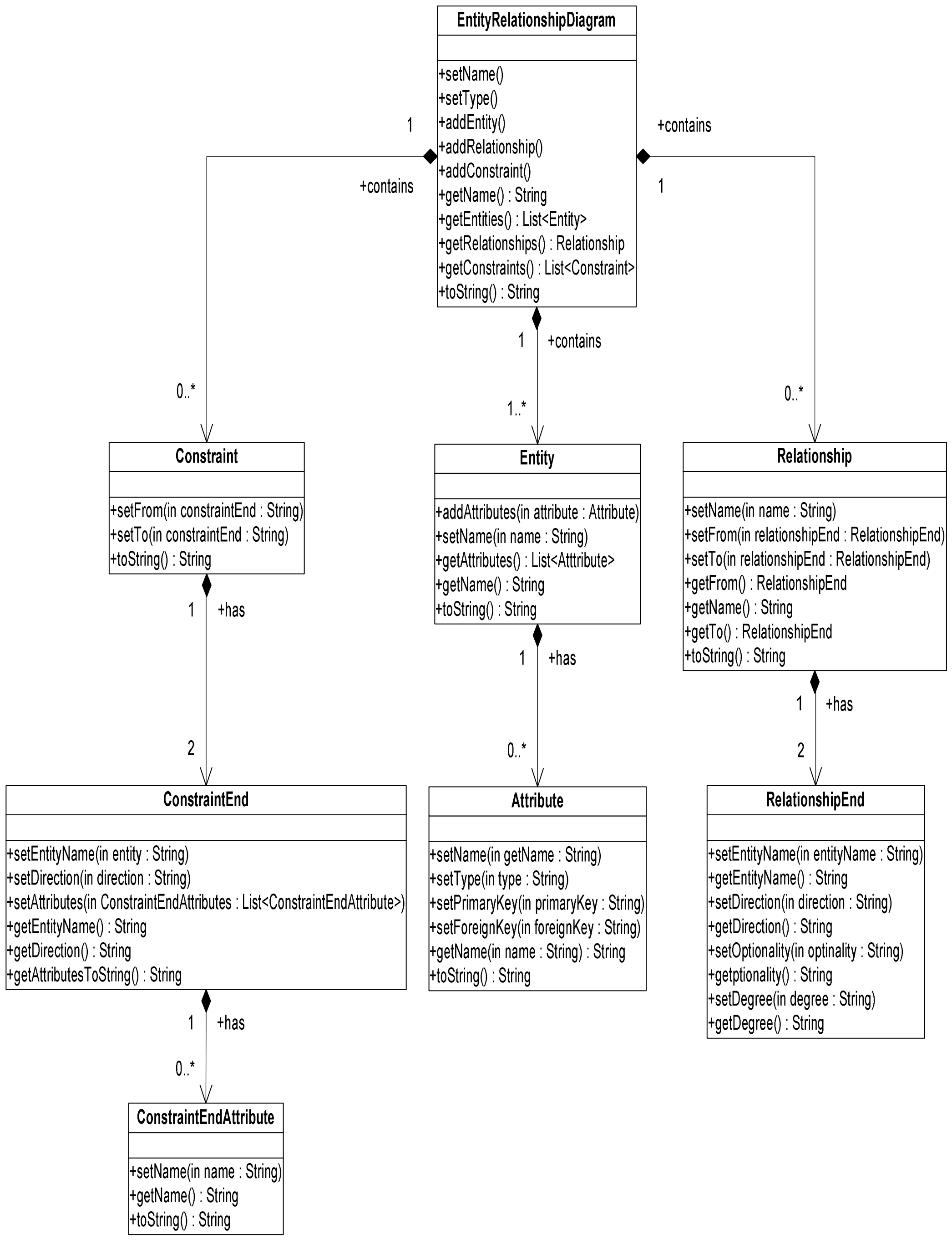
Dense ERD



Run								ExecutionTimes	# Vertices	ExecutionTimes (ms)
Start:	TestETA.xml	- Order:	4	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198125		5 Entities	15.00
End:	TestETA.xml	- Order:	4	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198140	15	10 Entities	16.00
Start:	TestETB.xml	- Order:	9	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198140		20 Entities	78.00
End:	TestETB.xml	- Order:	9	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198156	16	40 Entities	547.00
Start:	TestETC.xml	- Order:	19	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198156		80 Entities	8219.00
End:	TestETC.xml	- Order:	19	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198234	78		
Start:	TestETD.xml	- Order:	39	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198234			
End:	TestETD.xml	- Order:	39	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198781	547		
Start:	TestETE.xml	- Order:	79	- DateTime:	Fri Nov 02 15:09:58 GMT 2007	- msec:	1194016198796			
End:	TestETE.xml	- Order:	79	- DateTime:	Fri Nov 02 15:10:07 GMT 2007	- msec:	1194016207015	8219		
Start:	TestETF.xml	- Order:	159	- DateTime:	Fri Nov 02 15:10:07 GMT 2007	- msec:	1194016207015			
End:	TestETF.xml	- Order:	159	- DateTime:	Fri Nov 02 15:12:01 GMT 2007	- msec:	1194016321875	114860		

Appendix G – The ERD Schema Business Object

This section shows the class diagram of the ERD business object that is the in-memory representation of the XML schema storage. Only the main public methods have been listed. The description of the methods is not provided, as there name should clearly indicate their high level behaviour. Private properties have not been shown in the diagram to improve clarity.


EntityRelationshipDiagram
+setName()
+setType()
+addEntity()
+addRelationship()
+addConstraint()
+getName() : String
+getEntities() : List<Entity>
+getRelationships() : Relationship
+getConstraints() : List<Constraint>
+toString() : String
1
+contains
0..*
+contains
1
0..*
1
+contains
1..*
Constraint
+setFrom(in constraintEnd : String)
+setTo(in constraintEnd : String)
+toString() : String
Entity
+addAttributes(in attribute : Attribute)
+setName(in name : String)
+getAttributes() : List<Atttribute>
+getName() : String
+toString() : String
Relationship
+setName(in name : String)
+setFrom(in relationshipEnd : RelationshipEnd)
+setTo(in relationshipEnd : RelationshipEnd)
+getFrom() : RelationshipEnd
+getName() : String
+getTo() : RelationshipEnd
+toString() : String
1
+has
2
1
+has
0..*
1
+has
2
ConstraintEnd
+setEntityName(in entity : String)
+setDirection(in direction : String)
+setAttributes(in ConstraintEndAttributes : List<ConstraintEndAttribute>)
+getEntityName() : String
+getDirection() : String
+getAttributesToString() : String
Attribute
+setName(in getName : String)
+setType(in type : String)
+setPrimaryKey(in primaryKey : String)
+setForeignKey(in foreignKey : String)
+getName(in name : String) : String
+toString() : String
RelationshipEnd
+setEntityName(in entityName : String)
+getEntityName() : String
+setDirection(in direction : String)
+getDirection() : String
+setOptionality(in optinality : String)
+getptionality() : String
+setDegree(in degree : String)
+getDegree() : String
1
+has
0..*
ConstraintEndAttribute
+setName(in name : String)
+getName() : String
+toString() : String

Appendix H – Plug-in Installation

1. Install Eclipse SDK 3.3M4 for Windows XP (c.f. www.eclipse.org

2. Open the Eclipse application

3. Import the existing project from the CD-ROM: File -> Import… -> Existing Project into Workspace -> Browse to m801.Model_Checking_Tool -> click on Finish

4. Create the following path if it does not already exist - C:\Documents and Settings\All Users\Start Menu\Programs\a_Thesis\Eclipse\Eclipse SDK\eclipse-SDK-3.3M4-win32\eclipse-SDK-3.3M4-win32\eclipse\MyFiles\

5. Store the 'MiniCases' folder and content into the above location. Failing to do so will impede the user to run the 'execution time' test cases (as the path is hard coded)

Appendix I – Training Manual

These steps require the plug-in to be successfully installed (c.f. Appendix H, p. 142):

1. Run the Eclipse project menu -> Run -> Run

2. How to execute the 'Quality discovery' test cases?
    - 2.1 Got to the run-time Eclipse -> Menu -> Sample Menu -> GEV2008 User Interface
    - 2.2 A Java window will appear ->
        - 2.2.1 Select the ENHANCEDBOWERS Algorithm (none of the other listed algorithms have been implemented)
        - 2.2.2 Input the full path to a Test[Name].xml that leaves in the "MiniCases" folder, e.g. C:\Documents and Settings\All Users\Start Menu\Programs\a_Thesis\Eclipse\Eclipse SDK\eclipse-SDK-3.3M4-win32\eclipse-SDK-3.3M4-win32\eclipse\MyFiles\MiniCases\ TestS.xml
        - 2.2.3 Hit the 'Reload' button. This will load the graph of order 2. The potential redundant relationships and equivalent composed paths are listed in the table.
        - 2.2.4 Hit the 'Next' button to get the graph for the next order.

3. How to execute the 'Execution time' test cases?
    - 3.1 Got to the run-time Eclipse -> Menu -> Sample Menu -> Automated Dense Test to get the test result for a dense graph
    - 3.2 Got to the run-time Eclipse -> Menu -> Sample Menu -> Automated Sparse Test to get the test result for a sparse graph
    - 3.3 Got to the run-time Eclipse -> Menu -> Sample Menu -> Automated Sparse (Recursive) Test to get the test result for a recursive sparse graph

The results will be stored in the PerformanceLogger.txt file that resides in the directory where the Eclipse project lives.

Appendix J – Application Code

Please refer to the attached CD-ROM to view the code base of this application.