Translation
of an
Object Role Model Schema
into the
Formal Language Z

Gilbert Ravalli
Bachelor of Science (Honours), Melbourne University
Graduate Diploma in Mathematical Science, Melbourne College of Advanced Education
Diploma in Education, Melbourne University

Dissertation
for the degree of Master of Science (Research)

Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne Australia
December 2002
Abstract

In the development of information systems for business, structured approaches are widely used in practice. Structured approaches provide a prescription and guidelines for how to go about the process of developing an information system, are relatively easy to learn and provide tools which are well suited to their task. However, the products of structured approaches are sometimes seen to be vague and imprecise since requirements are written using natural language or represented in the form of models which do not have a formal foundation. This vagueness or ambiguity can be the source of problems later in development of the information system. A possible solution to this is to represent requirements using formal methods since these are seen as precise and unambiguous. However, formal methods are typically only a mathematical language for representing requirements. They are often regarded as difficult to learn and use. Even though formal methods of one sort or another have been in existence for many years they are not popular and appear unlikely to become popular in the future.

One possible approach to providing the advantages of structured approaches and formal methods is to provide translation procedures from the products of structured approaches to a formal description in a suitable formal language. The work in this thesis follows this theme and is aimed at the creation of a translation procedure from an Object Role Model (ORM) schema to a Z specification. An object role model schema is the end product of a process called the Natural Language Information Analysis Method (NIAM) which is used to produce an information model for an information system. NIAM is a method which has been used successfully in industry since the mid 1970s and continues to be used today.

This thesis provides a translation procedure from ORM to Z which is less arbitrary and more comprehensive than previous conversion procedures in the literature. It establishes a systematic method for
(i) choosing suitable types and variables for a Z specification and
(ii) predicates that express all the standard constraints available in ORM modelling.

The style of representation in Z preserves ORM’s concepts in a way that aids traceability and validation. The natural language basis of ORM, namely the use of elementary facts, is preserved. Furthermore, an ORM schema differentiates between abstract concepts and the means by which these concepts are represented symbolically and this thesis provides a representation in Z that maintains the distinction between conceptual objects and their symbolic representation. Identification schemes of entity types are also translated into the Z specification but it is left as an option in the translation procedure.
Guiding and evaluating the work conducted here are a published set of criteria for the evaluation of a conceptual schema. These have helped in making decisions regarding the translation procedure and for assessing my work and that of others.
Declaration

This thesis does not contain any material which has been previously submitted for a degree or diploma in any university.

To the best of my knowledge and belief, the material herein is entirely original and has not been reported by any other person except where indicated to the contrary.

Gilbert Ravalli
December 2002
Acknowledgements

I would like to thank my supervisor Dr Jim Sykes who has been my coordinating supervisor during the development of this thesis. As a part time candidate my progress has been painfully slow at times and he has been extremely patient and supportive over what has turned out to be a long time. I have greatly enjoyed the discussions over the years we have had about this thesis directly and all the other aspects of information systems both practical and philosophical that we have had. I have learned a great deal from him during that time.

I would also like to thank my other supervisors Associate Professor Paul Swatman who assisted in the earlier stages of my work and Professor Douglas Grant who helped me in the later stages.

I should also thank the Swinburne University for providing funding during the first three years of my candidature which paid my course fees. I also thank the School of Information Technology for providing a 20% work load reduction during 2000 and 2001 in order to work on my thesis.
Translation of an Object Role Model Schema into the Formal Language Z

Table of Contents

CHAPTER 1 INTRODUCTION ................................................................. 1
Thesis Scope ................................................................................................................. 1

Introduction ...................................................................................................................... 1
Structured Methods ......................................................................................................... 1
Formal Methods ............................................................................................................... 2
Systems Development as an engineering discipline ....................................................... 3
Advantages of Formal Methods ...................................................................................... 3
Criticisms of Structured Methods .................................................................................. 4
Formal methods are not widely accepted ........................................................................ 5
Integrating Formal and Structured Methods .................................................................. 8
Object Oriented Approaches ............................................................................................ 9
Data Modelling .............................................................................................................. 9
Fact Modelling ............................................................................................................... 10
Advantages of Fact Modelling ....................................................................................... 11
Previous work in mapping Fact Models to Z ................................................................. 12

Evaluation Criteria ........................................................................................................ 13
A Formal Specification as a Conceptual Schema ........................................................... 13

CHAPTER 2 CONVERSION OF ENTITY RELATIONSHIP MODELS TO Z16

Introduction ...................................................................................................................... 16

Review .............................................................................................................................. 16

An Example Illustrating the SAZ Method ...................................................................... 17
Selection of basic types ................................................................................................... 18
Schema types as Entity Types ......................................................................................... 18
Identification of entities .................................................................................................. 19
Relationships .................................................................................................................. 20

Discussion ......................................................................................................................... 21
Structure of the formal specification ............................................................................... 21
Selection and Use of Basic Types: ................................................................................ 22
Removing Implementation Independence ...................................................................... 23
Premature Locking in of Implementation Decisions ..................................................... 23
Resolution of Higher Order Relationships .................................................................... 24

Other work in converting ER diagrams to Z ................................................................. 25

Conclusion and Summary .............................................................................................. 26

CHAPTER 3 MAPPING OF FACT MODELS TO Z SPECIFICATIONS ......28

Introduction ...................................................................................................................... 28
Introduction................................................................................................................................................67

Total Population of an object type.............................................................................................................68

Internal Uniqueness Constraints across Fact Types .......................................................................................70

External Uniqueness Constraint spanning binary fact types .........................................................................72

Mandatory Role Constraints..........................................................................................................................74

 Subset Constraint........................................................................................................................................78

Role Equivalence Constraint.........................................................................................................................79

Exclusion Constraint.....................................................................................................................................81

Subtyping........................................................................................................................................................82

Student / Lecturer example ............................................................................................................................82

CHAPTER 7 CONVERSION PROCEDURE STEPS AND EXAMPLES .....85

Introduction..................................................................................................................................................85

Conversion Procedure Steps .......................................................................................................................85

Step 0 Standardize the fact model schema....................................................................................................85
Step 1 Declare Basic Types ...........................................................................................................................88
Step 2 Declare Schema Types ......................................................................................................................88
Step 3 Declare fact type variables ..............................................................................................................88
Step 4 Define internal uniqueness constraints ............................................................................................89
Step 5 Define mandatory role constraints ..................................................................................................89
Step 6 Define external uniqueness constraints ........................................................................................89
Step 7 Define subset constraints ...............................................................................................................89
Step 8 Define equality constraints ............................................................................................................89
Step 9 Define exclusion constraints ........................................................................................................89

Example 1 .....................................................................................................................................................90

Step 0 Standardize the fact model schema....................................................................................................92
Step 1 Declare Basic Types ...........................................................................................................................92
Step 2 Declare Schema Types ......................................................................................................................92
Step 3 Declare fact type variables ..............................................................................................................93
Step 4 Define internal uniqueness constraints ............................................................................................94
Step 5 Define mandatory role constraints ..................................................................................................94

Example 2 : A More Complex Example .......................................................................................................97

Step 0 Standardize the fact model schema....................................................................................................97
Step 1 Determine Basic Types ........................................................................................................................98
Step 2 Declare Schema Types ......................................................................................................................98
Step 3 Declare fact type variables ..............................................................................................................99
Step 4 Define internal uniqueness constraints ............................................................................................99
Step 5 Define mandatory role constraints ..................................................................................................100
Step 6 Define external uniqueness constraints ........................................................................................100
Step 7 Define subset constraints ...............................................................................................................101
Step 8 Define equality constraints ............................................................................................................101
Step 9 Define exclusion constraints ........................................................................................................101

Other constraints .........................................................................................................................................101

The Final Schema .......................................................................................................................................103
## Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4</td>
<td>31</td>
</tr>
<tr>
<td>Figure 5</td>
<td>36</td>
</tr>
<tr>
<td>Figure 6</td>
<td>36</td>
</tr>
<tr>
<td>Figure 7</td>
<td>37</td>
</tr>
<tr>
<td>Figure 8</td>
<td>45</td>
</tr>
<tr>
<td>Figure 9</td>
<td>45</td>
</tr>
<tr>
<td>Figure 10</td>
<td>62</td>
</tr>
<tr>
<td>Figure 11</td>
<td>69</td>
</tr>
<tr>
<td>Figure 12</td>
<td>70</td>
</tr>
<tr>
<td>Figure 13</td>
<td>71</td>
</tr>
<tr>
<td>Figure 14</td>
<td>72</td>
</tr>
<tr>
<td>Figure 15</td>
<td>73</td>
</tr>
<tr>
<td>Figure 16</td>
<td>75</td>
</tr>
<tr>
<td>Figure 17</td>
<td>76</td>
</tr>
<tr>
<td>Figure 18</td>
<td>77</td>
</tr>
<tr>
<td>Figure 19</td>
<td>78</td>
</tr>
<tr>
<td>Figure 20</td>
<td>79</td>
</tr>
<tr>
<td>Figure 21</td>
<td>80</td>
</tr>
<tr>
<td>Figure 22</td>
<td>81</td>
</tr>
<tr>
<td>Figure 23</td>
<td>82</td>
</tr>
<tr>
<td>Figure 24</td>
<td>85</td>
</tr>
<tr>
<td>Figure 25</td>
<td>86</td>
</tr>
<tr>
<td>Figure 26</td>
<td>86</td>
</tr>
<tr>
<td>Figure 27</td>
<td>86</td>
</tr>
<tr>
<td>Figure 28</td>
<td>87</td>
</tr>
<tr>
<td>Figure 29</td>
<td>91</td>
</tr>
<tr>
<td>Figure 30</td>
<td>97</td>
</tr>
<tr>
<td>Figure 31</td>
<td>121</td>
</tr>
<tr>
<td>Figure 32</td>
<td>122</td>
</tr>
<tr>
<td>Figure 33</td>
<td>123</td>
</tr>
<tr>
<td>Figure 34</td>
<td>124</td>
</tr>
</tbody>
</table>
Foreword

It is common when writing a piece of work such as a thesis to use sentences that are written in the passive tense. This takes out the work a sense of direct personal involvement. I suppose it suggests a certain distance from the work and perhaps a notion of an impartial and unbiased view of the research being conducted. However, it is also suggested (particularly from my grammar checker) that this makes the work more tedious to read and I agree with this sentiment.

I have written the thesis using a mixture of the singular and plural personal pronouns. I have used “I” and “my” when it was stating something that I myself did or claimed or thought which is directly attributable to me. For example, I would write “I have used an example taken from … to illustrate an idea” or “I believe this is the best way to do this” or “I claim that…” and so on. On the other hand, I have used “we” or “our” typically when working through an example such as “We do this first, then we can do that” or “If we did this, our result would be” or in situations where “we” could be replaced with the anonymous “one” or “someone” or “some persons” etc. such as in “we could look at the situation this way”.

When I use this device the reader should please keep in mind that there is only one person actually writing and this is not a “group” thesis. I have tried to be consistent in my use. I apologise if the reader takes offence and it is a straight forward task to render it to a more conventional style if need be.

Z schemas and other predicates have been type checked using the software package Z Formalizer (from Logica).
Chapter 1

Introduction

Thesis Scope

The primary purpose of this work has been to establish a systematic method for converting a schema founded on the Object Role Model (also called the Fact Model) into a formal specification using the Z specification language.

In the following section I describe the motivation behind this work. The second section describes the evaluation criteria I have used to evaluate both previous work and my own work. The third section describes what has been achieved.

Introduction

If one scans through the history of the software development process, what emerges quite strongly is the increasing structure, sophistication and precision that has been imposed on that process over the years as knowledge about how to go about the process has been learned and shared. In the early years of the computer industry, software development was a task that consisted essentially of writing programs or suites of programs for relatively small applications that would work given the considerable constraints imposed by the hardware of the time. By the late 1960’s, developments in hardware allowed applications of greater and greater complexity to be attempted and with that came the correspondingly increased need to become better organised in terms of the writing and structuring of programs. It became widely recognised that the “craft” approach hitherto accepted was not adequate for building the more complex systems that were required. Techniques for structuring and writing software that made it more orderly, predictable and readable became accepted.

Structured Methods

With the increasing complexity of applications systems being undertaken, it also became recognised that analysis and design activities needed to be undertaken as separate activities within the software development process. In the late 1970’s, approaches as to how to go about analysis and design in a structured (or systematic) way began to be published and gained acceptance. These were the so-called structured analysis and design methods (De Marco
that are widely used today. As with programming, we observe that this aspect of the software development process also has become increasingly sophisticated, orderly and predictable.

Structured analysis methods have now been in use for many years and been accepted by the information systems community as very useful in describing how to go about the process of analysing an information system. They provide techniques such as Entity Relationship Modeling, Data Flow Diagrams and State Transition Diagrams (amongst others) to represent the static and dynamic aspects of information systems. These methods have been well tried and understood by information systems developers.

**Formal Methods**

Within the community of those interested in the development of application software there are those interested in further increasing the sophistication of the development process by the use of appropriate mathematical techniques. Approaches that use mathematics for part or all of the development process are typically called *formal methods*. In the computing literature, which encompasses a very broad range of work, the word “formal” sometimes is used in a more colloquial way meaning associated with being “precise and unambiguous” or following very prescriptive rules. In such situations, it is possible to envisage a diagramming technique or a set of directions in a users’ procedures manual as being formal. However, within a narrower range of literature which I shall describe as computer science related, “formal” is usually directly associated with being soundly based on mathematics and anything else is regarded as either informal or semiformal. According to Hinchey and Bowen (1995, p.1), the description “formal method” originated from formal logic but now refers to a wide variety of mathematically based activities and means different things to different people. Hinchey and Bowen themselves define a formal method as

> “a set of tools and notations (with a formal semantics) used to specify unambiguously the requirements of a computer system that supports the proof of properties of that specification and proofs of correctness of an eventual implementation with respect to that specification.”

and I will use this definition within this thesis as the basis for my view of what constitutes a formal method. Some of the better known formal methods include: CCS, CSP, HOL, LOTOS,
OBJ, VDM, B-method and Z. It is with the last of these mentioned, Z, that this thesis is concerned.

**Systems Development as an engineering discipline**

What we now know as Computer Science grew out of the fields of Engineering and Mathematics and eventually established itself as a field in its own right (Parnas 1990). Some argue that the process of developing software is still essentially an engineering activity and that engineering principles should be applied to the process. Fundamental to engineering is the use of mathematics to specify and analyse systems. Hoare (1983) argued that programming was an engineering discipline that could be likened to civil engineering, and that mathematical techniques should be used in specification and development of software systems just as in other engineering disciplines. This theme of presenting the software building process as an engineering activity and the use of formal methods as an essential aspect of that process still continues. Still others also advocate formal methods as a vital part of software building, but do not necessarily also take on other aspects of the engineering paradigm.

**Advantages of Formal Methods**

Within the context of software development as an engineering discipline, it is perceived that the design of products relies on a theoretical foundation based on well-established laws, precise specification and rigorous testing (as much as possible done at the requirements gathering and elicitation stage) to ensure that they meet those specifications. There are several reasons why formal methods appear to satisfy this motivation:

- While formal methods do not have physical laws as a foundation, requirements statements effectively become the equivalent of “physical laws” governing the system to be created. Mathematically based languages such as set theory, predicate logic and process algebra are used to specify the requirements of the system. For example Denvir (1986) describes a software system in this way.

- It is claimed that a specification in a formal language is precise and unambiguous whereas specifications written in other forms, particularly natural language, may be vague or ambiguous (Potter, Sinclair and Till 1991; Spivey 1992, p.1). The possibility of a completely correct requirements statement is thereby improved by using a formal language.
• A specification written in a formal language can be manipulated and reasoned about in a mathematical way. As mentioned earlier, mathematically based languages such as set theory, predicate logic and process algebra specify the requirements of the system, but they also form the basis of a deduction system which allow us to reason about the behaviour of the system specified. This means that we do not have to build the product in order to test it (Denvir 1986, p.3; Hinchey and Bowen 1995, p.11; Potter, Sinclair and Till 1991, p.7). This is not to suggest that testing when the system or a part of the system is built is eliminated, just that testing can be incorporated as early as the requirements gathering stage in the expectation that the number of errors can be reduced in the later stages of development.

• It is possible to move from a requirement specification to a design specification and prove that the design actually still meets the requirements. It is well recognised that it is not practically possible to test a system, other than the very simplest ones, to the point that we can be sure that it meets the requirements exactly and in every respect. Dijkstra (Dahl, Dijkstra and Hoare 1972) pointed out that testing only demonstrates the presence of errors and not their absence. Potter, Sinclair and Till (1991, p.4), in a simple exercise, suggest that a medium sized system containing 1000 decisions would require $10^{300}$ test cases to test every possible scenario that might conceivably occur. Testing the system by considering all possible errors is therefore not a feasible approach to ensuring the correctness of the system. Formal notations allow the possibility of proving correctness by means of mathematical proof. This is accomplished in a series of specifications, each of which is a refinement of the previous one but heading closer toward some implementation model. Each specification is mathematically proved to be correct relative to its predecessor and eventually back to the original analysis model. (Potter, Sinclair and Till 1991, p.7; Spivey 1992, p.10).

**Criticisms of Structured Methods**

Those advocating the use of formal methods criticise the structured analysis methods stating that some of the techniques of structured analysis lack formality i.e. they are vague or ambiguous. Dunne (1995, p.878), states:

“*We should seek to reinforce our structured methods ... whose meanings have often till now been blurred and subjective.*”

Polack, Whiston and Mander (1994, p.542) write

“*Such precision is not readily available in a structured specification method*”
Semmens, France and Docker (1992, p.600) write

“... some of the [structured analysis] techniques lack formality”

In particular, techniques that involve narrative descriptions and dataflow diagrams are most typically mentioned. Randell (1990) states with regard to dataflow diagrams,

“They are relatively easy to understand, but as they have no formal basis may be understood in several ways.”

The quotes above relate to imprecise and ambiguous requirements statements. We can also add that structured methods do not provide an ability to reason about the behaviour of the proposed system (at least not in the way those advocating formal methods would like) nor to prove that an implementation completely satisfies its requirements other than by some means of exhaustive testing which I have indicated is not feasible.

**Formal methods are not widely accepted**

Based on the points and criticisms mentioned above and in the light of the historical trend towards more sophisticated approaches to software development, it would seem reasonable to expect a strong movement toward formal methods in preference to the structured ones mentioned earlier. However, according to Craigen, Gerhart and Ralston (1996) in a study of the use of formal methods in industry, the take up of formal methods approaches has been relatively small. In fact, they report that, in the main, formal methods have had very little impact in industry. Perceptions about formal methods, where they exist in industry, tend to be negative.

At this point, I suggest a set of factors to explain why formal methods are not generally accepted.

Most formal methods are not methods. A method describes the way a process is to be conducted. In the context of building information systems, a method is defined to consist of

1. an underlying model of development,
2. a language or languages,
Formal methods do not support all these aspects required of a method. Hinchey and Bowen (1995, p.2), Semmens, France and Docker (1992) and Dunne (1995) have argued that formal methods are essentially just powerful notations and sets of techniques and conventions for partitioning and refining specification and designs. Hinchey and Bowen (1995, p.1), suggest that many formal methods should be called *formal systems*. While researching this thesis, for example, I saw many examples describing formal methods where the impression is given that a specification is apparently produced directly from the problem domain with little or no guidance as to a process to be followed or any larger context which might exist. This might be perfectly suitable in the context of an academic exercise to be undertaken by students for example, but is unlikely to be regarded as a commercially acceptable “method”.

On the other hand, current structured methods provide a prescriptive framework that describes how to go about the process of system development. These methods have been developed, tested and refined over a long period of time using experience in the field. The criticism of structured methods is more about their deliverables whose interpretation are seen to be vague, imprecise or ambiguous rather than the process that one goes through in order to produce the deliverables.

The diagrammatic tools of the structured methods, which have been criticised as being vague or ambiguous, have a proven record as a communication tool to non-experts. In a survey on the use of Z, Barden, Stepney and Cooper (1991) reported, “the use of diagrams in structured methods was felt to be helpful and to aid understanding”. In her PhD thesis, Fowler (1996, p70) describes an extensive action research project using formal object oriented modelling (FOOM) where she also found that informal diagrams improved communication with users quite significantly.

There is a perception that formal methods are “highly complex, accessible only to highly trained individuals, and unsuitable for practicing software engineers” (Craigen, Gerhart and Ralston 1996, p.410). Furthermore,

> “the notations used with many formal methods are viewed as being complex and poorly compatible with existing industrial capabilities. In general, the notations are arcane and baroque; consequently they are difficult to read and write. The
notations are driven by mathematical and computing imperatives” (Craigen, Gerhart and Ralston 1996, p. 411).

These quotations from Craigen, Gerhart and Ralston (1996) are from a paper that describes the results of a survey on twelve applications of formal methods in industry as well as industry perceptions about formal methods. It is worth noting that those surveyed included engineers (from fields not directly related to computing) who presumably had strong mathematical backgrounds appropriate to their speciality and so could not be regarded as antagonistic to mathematics per se.

Weber (1996) suggests that formal methods are too ambitious; they “aim at a superior and uncompromising methodological framework and aim at perfectly correct systems” and that their rigid approach does not leave room for coexistence with other methods commonly used in industry. He further suggests that they often presuppose idealised circumstances that can only be realised in an academic environment and not in practice.

It can be claimed that diagrammatic tools are easier to work with (in a creative sense) at the early information analysis stages when the knowledge of the domain of interest is low and the structure of the system tends to be very fluid in the mind of the analyst. Formal notations can require a degree of precision which is inappropriate at an early stage and which many might find inhibiting because they do not have any visual impact.

Structured analysis methods are often used in large-scale projects and have proven to be a good (though certainly not perfect) approach to building systems. Hoare (1996) admits that, in spite of years of predictions of software disasters because of the increasing size of systems, these disasters have not occurred, at least not to the point where developers have felt the need to abandon their current techniques (structured or otherwise). Furthermore, while formal methods have been used in small to medium sized applications, according to Barden, Stepney and Cooper (1991) they still have to be proven on large scale problems which presumably they would claim to be able to address.

In the commercial area, the little evidence that exists about the financial benefits (in terms of cost, quality and time to market) of the use of formal methods suggests that its value is only marginally positive (Craigen, Gerhart and Ralston 1996, p.405). Furthermore, there is a perception that formal methods require a substantial initial investment in training time and money (Craigen, Gerhart and Ralston 1996, p.415) even though they suggest that this is not
substantiated by the facts. Nevertheless, the result is that there is little commercial incentive to abandon an existing method on the basis of potential financial gains.

**Integrating Formal and Structured Methods**

We appear to have some strong reasons to use formal methods on the one hand, but also some factors that inhibit their being used. The only areas where formal methods have had a significant impact are those where safety or security issues are particularly important. Rather than seeing formal and structured methods as mutually exclusive of each other, more evolutionary approaches have been suggested. If thoughtfully applied, the two approaches can be made complementary in such a way as to gain at least some of the advantages of formal methods while avoiding or reducing some of the factors that inhibit their use. This potential complementarity was noted in a survey on the use of Z by Barden, Stepney and Cooper (1991), where they reported that there “was widespread acknowledgement that Z could usefully be combined with structured approaches” whereas “structured methods were recognised for the advice they give on approaching problems”. Indeed, in one example given by Barden, Stepney and Cooper, the best Z specification was written by the team with the least experience in Z because of their knowledge of structured methods. This was because they knew

“*how to start, how to break down a problem and how to spot important aspects of the system*”.

Heisel et al. (1995, pp. 53-60) argue that to have formal methods more readily taken up by practitioners they need to be integrated within the methods commonly used in industry. Fowler’s apparently successful resorting to diagrams to aid users in her FOOM project (Fowler 1996, p73) also supports the suggestion that formal and structured methods may be complementary.

A common approach to integrating formal and structured methods is to use structured methods and techniques for the initial analysis and structuring and then to transform the models produced such as data flow diagrams, flowcharts, entity life histories, entity relationship diagrams and so on into formal specifications using a mapping procedure appropriate for each type of diagram. This will result in precise and unambiguous specifications that can be reasoned about and verified. Later any design produced can be validated against the formal specification. Several papers have been written which describe
mapping procedures from the various models produced using structured analysis methods into
formal specifications in Z that will be cited later as appropriate.

**Object Oriented Approaches**

At this point it is worthwhile considering object-oriented methods and how they might affect
the discussion so far. UML (OMG 2002) is effectively the industry standard for specification
of object oriented systems. UML makes use of both narrative descriptions (as in use cases or
scenarios) to capture at least part of the functional requirements and as well as diagrams such
as class or object diagrams (essentially a form of entity relationship diagram with
modifications to handle methods) and other diagrammatic tools (e.g. use case diagrams,
activity diagrams, state transition diagrams etc) to capture static and dynamic requirements of
systems.

One major difference between the structured and object oriented approaches is how the
software is interpreted i.e. as a system of objects interacting by the sending of messages
instead of processes to be performed which affect the states of entities linked by associations.
However, the types of models used to capture requirements (rather than specifically oriented
to software description) in most OO approaches are similar to those used in structured
methods.

The same criticisms are levelled at UML as have been levelled at the more traditional
structured approaches regarding their lack of precision, ambiguity, ability to reason about the
systems behaviour and so on\(^1\). In this regard, therefore, the motivation for this work is
unaffected.

Several articles have appeared regarding object oriented systems described using Z (Giavanni
and Iachini 1990). As well, extensions to Z have been developed to cater for object oriented
systems eg. Object-Z (Duke and Rose 2000; Rose 1992; Smith 1992; Smith 2000) and ZEST
(Cusack and Rafsanjani 1992; Rafsanjani 1993). However, these are still essentially formal
description tools rather than complete methods.

**Data Modelling**

It is commonly stated that there are three perspectives to a requirements specification, namely
data, behaviour and process. Though obviously interrelated, the data perspective is often
regarded as the foundation upon which the others are built. However, I do not wish to become involved in arguments as to whether this is in fact the case and will be satisfied if the reader merely accepts that the data perspective is an important one and worthy of consideration. The transformation of a data model schema into a formal specification is of interest in this thesis.

To describe the data perspective of an information system using structured approaches, the most well known data modelling technique is the Entity Relationship model. There are several papers which have outlined procedures for mapping Entity Relationship diagrams, normally developed as part of some structured method, into Z specifications (Semmens and Allen 1990a; Redmond-Pyle and Josephs 1990; Polack 1992). In a later chapter, I look further into these particular mapping approaches.

**Fact Modelling**

An alternative data modelling technique to Entity Relationship Modeling is Fact Modeling (Halpin 1995; Nijssen and Halpin 1989; Wintraecken 1990). This is also sometimes referred to as NIAM (Nijssen’s Information Analysis Method or Natural Language Information Analysis Method), or more recently, the name Object Role Modeling (ORM) has been revived.

There are several important influences that were brought together in the fact modelling approach. Abrial (1974) described what he called the Binary Object-Role Model (B-ORM). One of the earliest papers in which one recognises a Fact Modelling like approach was by Falkenberg (1976). He introduced the “role” concept based on the work of Filmore (1968) to relate objects within an association that gave a linguistic foundation and described his approach as the “object role model”. Later, there were developed various versions of what is called the Conceptual Schema Design Procedure (CSDP). The CSDP describes how to go about the process of developing a fact model.

1 There are some less well-known methods that do make use of some degree of formality. FOOM, previously mentioned, is one of these.
2 I will use the terms Object Role Model and Fact Model interchangeably in this thesis.
3 Descriptions of NIAM sometimes include Information Structure Diagrams (a variant of dataflow diagrams) as part of the method and not just the data modelling component
4 Fact Modeling is not well known here in Australia when compared to Entity Relationship Modeling. The fact modelling approach originated in Europe and a large proportion of the literature to be found in this area is published there. Its proponents are scattered, as is the literature, and so it is not easy to identify it amongst all the other literature and methods which proliferate unless one looks for it. Compounding this is the dominance of American textbooks in the computing area in Australia and the ease with which American academic literature is searched. However, Microsoft has now incorporated
Because Fact Modelling is well described in the literature, I do not intend to describe it here.

**Advantages of Fact Modelling**

I believe that fact modelling offers advantages over entity relationship approaches:

1. **The problem of misunderstanding between the users and analysts leading to errors in systems is well known. Christel and Kang (1992, p.11), for example, describe the problem of the ambiguity when natural language is used by users to express requirements and the difficulty users have in verification of formal or semiformal models developed by analysts.**

   Fact Modeling is based on the elicitation of information to be recorded in an information system in the form of simple natural language statements. These are then formalised into a fact model schema that describes the structure of the information to be captured. However, at all times the formal schema is expressible in natural language and further it can be validated by reference to instances of facts. So, while natural language is used (with which users will generally feel comfortable), it avoids the problems of ambiguity and hence misunderstanding which typically characterise interactions between users and analysts. Fact Modeling therefore forms an ideal method of bridging the gap between the users potentially informal statement of requirements and the analyst’s need for a formal model of the UoD.

2. **Fact modelling is not just a notation but also the end product of a systematic method of discovery and checking of facts about the problem domain (the Conceptual Schema Design Procedure mentioned earlier). If one adopts fact modelling then one “inherits” the method that goes with it. This provides an advantage over entity relationship modelling.**

3. **Fact modelling notation provides an ability to specify constraints in a richer way than can be done using ER modelling notation. A fact model describes data and data relationships at a level somewhat lower than a ER model which encourages a more detailed analysis to be made and a more “fine grained” specification to be presented. In accordance with this, the constraints can also be of a more fine-grained nature.**

4. **Following on from the point above I believe that an ER model imposes constraints on possible implementations of the system that are not necessary i.e. it represents a partial design**

---

fact modelling with other data modelling tools in its VISIO product and employs Dr Terry Halpin who actively promotes NIAM and fact modelling.
element which may be taken as a requirement when it is in fact not a requirement. The ISO report (1985) titled “Assessment Guidelines for the Conceptual Schema Language Proposals” elaborated on some aspects of the original ISO report on Conceptual Schema Design principles. The report stated:

“The information analyst must be free to express the conceptual schema in terms appropriate to the specific universe of discourse of concern and to the user’s perception of it. Specifically, no constraints are to be imposed on the entities assumed to exist in the universe of discourse or on the properties they may be asserted to possess.” (ISO 1985, p.11)

I argue that a fact model provides a base structure of elements from which it is possible to “build” entities as required either for conceptual or design purposes whereas an entity relationship model imposes a higher level structure which constrains views of the UoD or implementations of the system.

5. Entity relationship modelling is closely tied to relational implementation. On the other hand, while algorithms for transforming a fact model into a relational implementation are quite simple and straightforward, one can argue that a fact model is more flexible tool not as closely tied to the relational database model.

6. Fact modelling is often been used successfully for data analysis of small to medium scale problems but can also be used for large-scale problems at the corporate level as described by James and Olsen (1994).

7. Fact modelling distinguishes between objects (which are conceptual) and the labels (string values or numbers) and provides methods for determining whether objects can be uniquely identified and, if so, how. Polack, Whiston and Hitchcock (1992, p.370) mention uniqueness identification as a possible problem leading to lack of precision in a formal specification. Fact modelling can solve this problem because it is an issue directly addressed by the fact modelling approach.

**Previous work in mapping Fact Models to Z**

---

5 UoD standards for Universe of Discourse and refers the application area under consideration.
Only two papers, that I am aware of, have outlined procedures for transforming a fact model into a Z specification by Habrias, Dunne and Stoddart (1993) and Nguyen and Duke (1995). The purpose of this thesis has been to establish a systematic method for mapping a fact model schema into specifications in Z while extending the work of Habrias, Dunne and Stoddart (1993) and Nguyen and Duke (1995).

**Evaluation Criteria**

In the work accomplished so far in converting graphic data models into formal specifications into Z, no systematic attempts have been made to evaluate or compare the results according to any evaluation criteria. These criteria might judge the conversion procedure or that one conversion procedure might lead to better results than another. I have chosen to use those suggested by Loucopoulos and Zicari (1992, p15) as the basis for criticising and comparing both my own work and that others. These criteria are

- Implementation independence
- Abstraction
- Formality
- Constructability
- Ease of Analysis
- Traceability
- Executibility

The explanation of these criteria are included in Appendix A and used in the concluding chapter of the thesis.

**A Formal Specification as a Conceptual Schema**

I believe these evaluation criteria are relevant because I am examining formal specifications embedded within the structured approaches to building information systems and so I use evaluation criteria relevant to that area. In this context, a formal specification is judged by criteria which may be outside the scope of mathematical considerations alone.

However, I will make this judgement only on the first formulation of the formal specification. I would not demand this requirement on subsequent refinements of the formal specification since they must adhere to different principles related to design and implementation. The first
formal specification statement is the one that is likely to be closest to the UoD description as given by the users and thereafter with each refinement is likely to be further and further removed from concepts that would be familiar to the end users. In any case, we can be sure that if the refinements are performed correctly the system will behave as required in the original statement.

If the purpose of this thesis is to suggest that something else should form the “front end” to the formal model in Z, then it is reasonable to ask why the formal model should be judged on the same grounds as its “front end”. These front ends to the formal model are essentially just a means to an end, aiding the analysts and users. The formal specification must eventually be the final “arbiter” of the functionality of the proposed system. There may be aspects of the system that need to be considered which are best done using the formal model. For example, the analyst may wish to test the formal model in various ways by proposing hypotheses about the systems behaviour and then confirming or denying these hypotheses. However, these hypotheses will still need to be obtained by direct consideration of the UoD and the consequences need also to be explained in terms directly relevant to the UoD. If the formal system retains qualities such as constructability, traceability, ease of analysis and executability then this assists the analyst in obtaining a correct and complete requirements specification.

Whether the requirements of an information system are stated in the form of text, a graphical model or in a formal mathematical notation such as Z, the requirements statement should still adhere to the principles above.6

In evaluating a translation procedure, I will apply the principles to decide whether the Z specification is satisfactorily produced in the sense that every aspect of the original form of the specification (in this case an ER diagram or a fact model) has been carried into the Z specification. This relates to the completeness and correctness of the first formal statement of specification.

**What has been achieved?**

The primary aim of the thesis i.e. to develop a systematic procedure for conversion of a fact model into a specification into Z has been achieved. I claim that the procedure is straightforward in its approach, produces consistent results and largely satisfies the evaluation criteria mentioned above. It is an advance on previous work in that I have provided a reasoned

---

6 Because we are dealing with data models we have ignored the dynamic aspects of the system to which the principles refer.
justification for the manner in which each concept in the object role model is translated and is more comprehensive by including more of the constraints as part of the translation procedure than previous work of this type that has been described.

One area of interest that has emerged as a result of this work is that the selection of basic types in Z is not an issue that is discussed in the literature in any detail. I will go into this in more detail in chapter 4 so at this point I will merely state that I have addressed this issue and that my translation procedure does indicate a systematic method by which basic types could be selected.

Overview

In chapter 2 I review the work on converting data models to Z.
In chapter 3 I examine the work on translation procedures from ORM to Z.
In chapter 4 I look at various approaches to modeling in ORM and select one of these approaches.
Chapter 5 considers the fundamental concepts of the ORM model and examines the abstractions available in Z in an attempt to find the ones most suited to “carrying” those abstractions.
In chapter 6 ORM constraints patterns are translated into Z specifications in a piecemeal fashion.
In chapter 7 I describe the translation procedure and provide fairly complex examples to illustrate and exercise most aspects of the translation procedure.
In chapter 8 I discuss dynamics i.e. transactions on the database. The purpose of this chapter is to get an idea of the repercussions on transactions of expressing specifications according to my method.
Chapter 9 is devoted to considering identification schemes.
Chapter 10 reviews and assesses my work and suggests possible further work.
Chapter 2
Conversion of Entity Relationship Models to Z

Introduction

In this chapter I review and discuss the work that has been done in converting the entity relationship model into specifications in Z. I summarise the approaches that have been taken and evaluate them from the viewpoint of the criteria for evaluating a conceptual schema.

Review

Some work has already been published in transforming Entity Relationship models into Z. Some earlier papers on the subject include (Redmond-Pyle and Josephs 1990; Semmens 1990; Semmens and Allen 1990a; Semmens and Allen 1990b). The SAZ project investigated the use of Z as a complement to the structured method SSADM (Downs, Clare and Coe 1992) to discover errors and ambiguity in specifications and provide more rigor. SAZ, an acronym for SSADM to Z, provides procedures for converting various graphical models produced in SSADM, namely Data Flow Diagrams, Entity Relationship Diagrams and Entity Life Histories, into formal specifications in Z. The result of this project is described in a number of papers (Polack 1992; Polack, Whiston and Hitchcock 1992; Polack, Whiston and Mander 1993; Polack, Whiston and Mander 1994; Polack and Mander 1994) and builds on the work of the earlier papers mentioned. I am only interested in that part of the method relating to the data modelling aspects which converts a variant of the Entity Relationship model (a logical data model, LDM) into a formal specification into Z.

According to Mander and Polack (1995), the logical state of a system in SSADM is expressed by the data stored within the logical data model which is implicitly based on the relational database model. An example of this implicit base is that the logical data model expresses relationships using foreign keys which are clearly a relational database concept. Mander and Polack also mention that many state invariants are hidden within other documentation. When preparing a specification in Z one must map the information expressed by the data model and then incorporate state invariants found elsewhere. Therefore the Z specification provides a useful point for collection and structuring of information based around the static data...
requirements of the system. The effort of expressing the information in a rigorous way helps in the discovery of errors and omissions which would not be easily discovered until later stages of the systems development lifecycle.

Polack, Whiston and Hitchcock (1992) give the most comprehensive description of the SAZ method for converting the Entity Relationship model into a formal specification into Z. This procedure basically states that entity types are represented as schema types, relationships are shown as functions or relations between two entity types and predicates define the scope and optionality of each relationship. Although the mapping procedure shows how to represent many to many relationships, in the logical data model these relationships have been resolved into one- to-many relationships by creation of what are sometimes described as “intersection” or “weak” entity types. The SAZ method does not describe how to represent ternary or higher order relationships so I presume that these must also have been resolved earlier by creation of these intersection entities.

**An Example Illustrating the SAZ Method**

In order to illustrate the mapping procedure I use an example used by Polack, Whiston and Mander (1993) and shown in the figure below.

![Diagram of Entity Relationship model](image)

**Figure 1**

7 A state invariant is a constraint imposed on the data which must apply in each state of the database.
Mandatory and optional role constraints are shown by solid and dashed lines respectively and subtyping is shown by including subtype boxes within a larger box representing the supertype.

**Selection of basic types**

The method maps each attribute of each entity type into a given (or basic) type. However, I observe that where attributes appear to be semantically similar in nature such as the case with an attribute “date_of_birth” and an attribute “date_when_job_commenced” where date is an obvious underlying basic type or the attributes “stock_quantity_ordered” and “stock_quantity_available” where integer (which is the only basic type provided by Z) might be chosen as the basic type, this rule is overridden and one of these “semantic domains” may be chosen in preference to creating a basic type for each attribute. It is suggested by Polack, Whiston and Mander that basic types should be “meaningful”. There is no clear definition or clarification of what is intended by the term “meaningful”, but Polack, Whiston and Mander do state that implementation types such as CHAR are not normally regarded as suitable basic types at this stage.

Continuing on with the example, the basic types selected are

[TITLE, ISBN, AUTHOR, NAME, TEXT, RATING, DATE]

**Schema types as Entity Types**

Entity types are defined in schemas as a collection of their attributes. For example, the entity type BOOK is defined as

```plaintext
BOOK
  Title: TITLE
  Author: AUTHOR
  Stock_Qty: N
  Print_Status: PRINT_STATUS
```

ORDER and ORDER_LINE are similarly defined.
In the case of subtyped entities, the supertype schema defines the attributes which are common to the supertype and the schema list only the attributes common to each subtype.

The Z schema above establish domains (via the basic or free types) and map entity types to schema types which may be referenced through their attributes.

**Identification of entities**

Entity types developed in the Logical Data model in SSADM include attributes which provide the key of the entity. Attributes from other entity types may be imported from other entity types in order to achieve this. An extreme case of this occurs with some intersection entity types which have no attributes of their own but merely consist of imported attributes from other entity types; their purpose is essentially to link entities together through foreign keys. However, Mander and Polack (1995) state that this pre-empts design decisions and so these imported attributes in entity types are removed. As a result, entity types that consist only of imported attributes will disappear. For those entity types remaining, entity types that had imported attributes will contain entities which are no longer uniquely identifiable. As an example of this inability to identify entities, consider order_lines which have the same attribute values but which belong to different orders; without the imported attribute(s) from
the entity type “order”, the order lines are not distinguishable. As a result of this problem, surrogate keys are created in order to establish a unique identity for each entity.

The SAZ method defines the set of instances for each entity type as a function mapping between the type of the entity to the type of a corresponding unique identifier which is the identifying key of the entity. This gives a set of pairs of the form (IDENTIFIER, ENTITY). It is suggested that surrogate identifiers be used although an attribute which can act as a unique identifier can also be used. Where surrogate identifiers are needed, a new basic type is defined that will provide a surrogate key for each entity and a partial function which links each entity instance known to the system to a unique key. The generic form of this association is shown below

```
[IDENTIFIER]

ENTITY_SET

Key: IDENTIFIER → ENTITY_TYPE
```

It is suggested by Mander and Polack (1995) that, later in the development of the information system, these surrogate identifiers will be replaced by an attribute or set of attributes and a predicate specifying the uniqueness of the attribute(s).

**Relationships**

Relationships are represented as mathematical relations or functions between entity types. In doing so one associates entity instances using the entity identifiers which have been established as described in the previous section. If EntityId_A and EntityId_B are surrogate identifiers for entity types A and B then a many to one relationship named many_to_one would be written as

```
many_to_one: EntityId_A → EntityId_B
```

An example of this would be the relationship between an order and an order line. This would be expressed as

```
Order_Line_is_Part_Of_Order_Relation: ORDER_LINE_ID → ORDER_ID
```

A many to many relationship named many_to_many would be specified as
Discussion

We can make several comments relating to the Z specification created and to the process involved.

Structure of the formal specification

Notice that the structure of the formal specification matches the structure of the Logical Data model. One reason for having as close a correspondence as possible is that this aids in traceability between the Logical Data model and the formal model. Since a specification in Z is not directly executable, any errors introduced in the formal specification can at this stage only be detected by inspection and a close correspondence should aid this inspection process considerably.

In SSADM, to obtain the logical data model the system firstly by considering only the structural aspects and then secondly by decomposing this into entities, attributes and relationships. By mapping this decomposition into the formal specification, the constructability of the formal specification is also improved since entities, attributes and relations would be regarded as suitable abstractions. Furthermore, I would expect that reasoning about the system would be easier if the system matches the analyst’s logical view of the system. This will be of greater use to the analyst rather than to the users who are unlikely to be able to view the formal model critically. I note that a similar approach is adopted by those using object oriented methods and producing formal models i.e. they attempt, where possible, to produce a formal model which matches the analyst’s logical view of the system as set of interacting objects containing both data and behaviour. I assume that this is done for similar reasons.

Ease of analysis is improved by having two models, one in a diagrammatic form and the other a formal one. Different ways of looking at the system may highlight incompleteness, inconsistency and ambiguity. Each style of specification can act as a means of cross checking the other.
Selection and Use of Basic Types:

We can note that Stock_Qty, Ordered_Qty, Amended_Qty, and Credit_Rating are all defined to be of type INTEGER. The first three clearly appear to be of the same type since they are all quantities of stock and therefore can (and will) be meaningfully compared. However, while Credit_Rating might happen to use integers to allocate a rating to a person, this is likely to be acting as a code and therefore the type integer is merely a convenience. Credit ratings cannot be meaningfully compared to stock quantities and, since values for credit ratings are codes, they are unlikely to be manipulated in the same way as a value for a stock quantity i.e. as an integer where we might perform arithmetic functions like add, subtract, multiply and divide.

A further point that I will make about the specification relates to the creation of the basic type AUTHOR. It is suggested by Mander and Polack (1995) that, in the implementation stages of the information system, this basic type might be replaced by something like name and initials. It is unclear whether a different implementation of [AUTHOR] would be introduced without making changes to the specification or whether [NAME] would replace [AUTHOR] or a new basic type such as [AUTHOR_NAME] is to be established. I believe that this ability to “swap and change” is an unsatisfactory and unnecessary state of affairs.

Semantic domains (implemented in Z as basic types) should be meaningful. My view is that one aspect of being meaningful is that correctly chosen basic types should allow comparison of objects which I believe to be essentially of the same type and prevent comparisons of objects which are not. I do not believe that objects should need to change their type as we proceed through the development lifecycle except possibly for very pragmatic reasons connected to limitations of the technology. Even in the relational model, the basic concept of using semantic domains is regarded as a useful one (Date 2000, p.112). In the case of AUTHOR and NAME we note that at the analysis stage objects were regarded as semantically different and therefore not comparable but later in the development process there is the possibility that they may not be different after all (through having chosen equivalent representations) and that a legitimate comparison is actually possible.

I have only looked at one example here, but this is typical of the SAZ method in general. The conclusion that I draw from examining the selection and use of basic types in Z specifications is that the process can be improved so that basic types are more appropriately selected and allocated.
Removing Implementation Independence

In the SAZ method, some reverse engineering is employed in that

- entity surrogates replace the use of entity sets and that
- foreign keys are replaced by functions and relations to represent relationships with predicates defining the optionality of the relationship.

The non-inclusion of foreign keys in entity sets and the use of surrogate keys is seen by Mander and Polack (1995) as advantageous from two aspects. Firstly, it does not pre-empt implementation decisions as to how the entity sets might be represented. The entity surrogate can be replaced by an attribute or group of attributes. Secondly, it is seen as a potential quality assurance problem to ensure that the specification maintains data integrity. A simple example they describe suggests that using the entity surrogate [AUTHOR] as a basic type allows system builders to later refine this to a set of names with title, initials etc. or any other set which might be suitable.

In reviewing how SSADM and Z support each other, SAZ attempts to “repair” problems introduced by SSADM such as implementation dependence. In so far as the data modeling aspects are concerned, one may conclude that SAZ provides a useful complement to SSADM. This is of course valuable, but it would seem better to try to prevent the problems from the start. The sequence of events involved in specifying a logical data model in SSADM, reverse engineering it to specify it in Z and then proceeding on to another implementation model while working in Z at a later stage seems, at best, inefficient and at worst likely to introduce errors.

Premature Locking in of Implementation Decisions

The exercise of reverse engineering from the logical data model may not discover the “right” or most appropriate entities. The logical data model is essentially a representation of the relational model. Starting with an implementation model is not a systematic way of discovering entities and relationships and making informed decisions as to the full range of possible implementation options that might exist. As an example of this, the logical data model for an order processing system provided by Mander and Polack (1995) includes CUSTOMER, ORDER, ORDER LINE and BOOK. The Z specification suggested for this creates entity surrogates for each of the above and links them to entity sets listing their attributes. However, there is no particular reason why ORDER LINE is essential as a concept
to the specification and there are ways to avoid it or replace it with other concepts. In a Fact Model analysis of the order situation, for instance, the fact types

Customer ... requested the quantity ... of book ... on date …
Customer ... amended the quantity .. of book ... ordered on date …
The annotation … is attached to the book … ordered by customer … on date …. 

capture the same information as is contained in the entity type ORDER and ORDER LINE and its relationships but the concepts of an order and order line have disappeared as explicit entities. However, the reverse engineered Z specification has “locked in” ORDER LINE and ORDER as the implementation objects of choice even though it represents only one of several possible implementation choices. There are many possible criteria which may be taken into account in making the best choice for implementation so an important issue is whether the specification allows the designer the freedom to consider all the available options.

It should be emphasized here that the idea I am trying to convey is that the process of selecting the most appropriate way to represent the UoD should be performed as part of the original business analysis where the business users should be part of a consultation process. At this time ideas can be explored, appraised and the best option chosen. After this the best method of implementing the requirement can be chosen.

**Resolution of Higher Order Relationships**

Mander and Polack (1995), comment that there was a quality assurance problem with the existence of entities which contained only foreign keys. Removing these resulted in replacing two many-to-one relationships between three entities with one many-to-many relationship between two entities. Higher order relationships (where three or more entity types, not necessarily all different, are involved in the one relationship) do not appear to be mentioned in any of the literature related to SAZ. I presume therefore that they must be resolved to many to one binary relationships. In SSADM the stated reason for performing the resolution process is that “it is a powerful analysis technique … which provides more information about the two master entities and the relationship between them”. However, while there may be some truth in this, it might be suggested that this process may be related more to consideration of relational model implementation rather than analysis concerns. Extending the quality assurance argument given by Mander and Polack (1995), I believe that the same quality assurance problem exists if higher order relationships are resolved. Later in this work, I will
criticize the approach of resolution in general because it leads to a proliferation of basic types, the selection of which is relatively arbitrary.

**Other work in converting ER diagrams to Z**

The work of Semmens and Allen (1991) in converting Yourdon diagrams to Z and Redmond-Pyle and Josephs (1990) in the integration of LBMS SE and Z have the similar aim of converting the products of structured techniques into formal representations. They have very similar approaches to that already described so I will concentrate more on the differences.

The Semmens and Allen (1991) procedure for generation of formal specifications has the following features:

- **Z basic types** are defined for each distinct attribute type. In other words, each attribute becomes a basic type
- A **Z schema** is defined for each entity type
- **Datastore schemas** (equivalent to the Polack, Whiston and Mander entity set) represent the instances of each entity and include an injection from the entity type to the attribute(s) which form a unique identifier
- Relationships are declared as functions or relations with predicates which define cardinality and mandatory role constraints

One can make the criticism with this approach that whereas Polack recognised that sometimes different attributes could be of the same type (such as two or more attributes based on dates) and hence created a common underlying basic type, the Semmens and Allen procedure does not recognise this and therefore we could find that the specification would not allow comparison of attributes which could be of the same type semantically.

In the approach taken by Redmond-Pyle and Josephs (1990), the transformation into Z is done before any decisions are made about which attributes will form the primary key so it is performed slightly earlier in the development process. An abstract identifier is used for each entity type. An injection from the abstract identifier to the entity type is included in a “table schema” (equivalent to Polack’s (1992) entity set). This deferment of selecting a primary key (amongst several candidate keys) may provide an advantage in later system development but, on the other hand it introduces variables whose significance when trying to relate the specification back to the application concepts is not clear.
Conclusion and Summary

In reviewing the literature on mapping procedures for ER models into specifications in Z the following has been found:

Producing an entity relationship model provides a way of decomposing the system in a manageable way. This decomposition is mapped into the specification in Z thereby enhancing the constructability, ease of analysis and traceability of the formal specification.

1. The various mapping approaches from Entity Relationship models to Z are very similar.

2. Attributes of entity types are more or less directly converted to basic types with consequent inability to compare attributes which appear to us to be semantically of the same type.

3. Entity types from the ER model are converted into schema types in the Z schema.

4. Identification of instances of entity types may use either sets of attributes or surrogate identifiers or both.

5. Ternary and higher order relationships are resolved with the subsequent creation of new (intersection) entity types which may not be part of the original domain of entity types.

6. Many to many relationships may also be resolved to produce new (intersection) entity types which also may not be part of the original domain of entity types.

I am generally positive about the usefulness of using ER modelling as a front end to a Z schema because it provides a systematic means of structuring the formal model and provides a logical view of the system which is useful to the analyst. I have made some criticisms of the SAZ method, some of which relate to the fact that it is used as a reverse engineering tool which is a criticism about when it is being used rather than the method itself. More important criticisms relate to

- the manner in which basic types are selected in the formal specification
The first point I see as a problem which cannot be addressed easily by the entity relationship approach since I do not see any mechanism which would deal directly with it. The second point could be addressed by not performing the resolution process but this impacts on the mapping procedures which would have to be modified significantly.

I believe that the n-ary approach to fact modelling can address the deficiencies just mentioned by not requiring the introduction of resolved entities and by providing a means of selecting basic types through a consistent and judicious mapping of object types from the fact model.

The next chapter examines the current literature for mappings from fact models into Z.
Chapter 3
Mapping of Fact Models to Z Specifications

Introduction

There appear to be only three papers outlining the mapping of fact models into Z. These are by Habrias, Dunne and Stoddart (1993), Nguyen (1994) and Nguyen and Duke (1995).

In this chapter, I describe their mapping methods and comment on them. I first consider the Habrias, Dunne and Stoddart method and then Nguyen’s method.

Habrias, Dunne and Stoddart Method

Habrias, Dunne and Stoddart (1993), in a working paper, describe two approaches to translating a fact model into Z. Their work is exploratory in nature and the intention was to provide ideas about how such a translation might be performed. They provide an example of each approach but do not provide generalised algorithms for the mapping procedures. As a result, in describing their approaches, while trying to be as faithful as possible to their work, I may not have always correctly inferred the intention or I may have attempted to extend the work beyond the thoughts at the time. They do not provide an detailed evaluation of the approaches that suggests a preference for one approach over the other.

The examples of fact models from Habrias, Dunne and Stoddart (1993), include only binary facts to illustrate the mapping procedure and it is unclear as to whether this stems from a binary only fact modelling approach (rather than an n-ary approach) or whether this is simply incidental. If a binary only approach is taken then this is achieved by resolving any ternary or higher order fact types into binary ones by the creation (or “discovery” as it is sometimes described) of intermediate object types. For the purposes of this section we need not consider this aspect.

Both translation procedures begin by grouping facts together to produce a set of relational tables which are restricted in that the relational tables do not allow null values for any attribute. The grouping procedure is not outlined by Habrias, Dunne and Stoddart (1993), but detailed descriptions are provided by Nijssen and Halpin (1989) and Halpin (1995). Constraints on the fact model are translated into equivalent constraints in the relational tables. One needs to refer to both the fact model and the set of relational tables in order to generate
the Z schema. This step is intuitive in nature and left to the judgement of the analyst as to the most appropriate representation in Z.

I illustrate the translation procedures using a sample fact model given by Habrias, Dunne and Stoddart (1993).

![Figure 2](image)

The set of relational tables generated from this is shown below (Figure 3) together with the primary key constraints (solid underline) and foreign key constraints (dotted lines between tables) which apply.

![Figure 3](image)
Method 1

Habrias, Dunne and Stoddart create a schema type for the entity type Student and interpret the roles “born on”, “entered in IUT on” and “with” (name) as well as the student number as the attributes of Student. Each of these roles has a total (mandatory) role constraint on them which means that every student of interest in the UoD will have a value for each of these attributes. This gives the following (incomplete) schema for Student:

```plaintext
STUDENT
  StudentNumber
  StudentName
  EntryDate
  BirthDate
```

With regard to other object types, Habrias, Dunne and Stoddart create a basic type for each entity type which has a simple identifying label type, the corresponding label type is omitted and a basic type is created for each remaining label type. This results in the basic types:

```
[COUNTRY, NAME, DATE]
```

The schema can now be completed:

```plaintext
STUDENT
  StudentNumber: N
  StudentName: NAME
  EntryDate: DATE
  BirthDate: DATE

IUT_OF_NANTES
  country: P COUNTRY
  students: P STUDENT
  is_placed: STUDENT → COUNTRY
  speaks_language: STUDENT ↔ COUNTRY

∀ s1, s2: students | s1 ≠ s2 •
  s1.studentNumber ≠ s2.studentNumber
  dom is_placed = dom is_placed
  is_placed ∈ speaks_language_of
```

The basic type N appears in the schema because in Z it is an assumed available basic type (actually the only one that one does not have to be declared). Although the example provided is limited, some relevant comments can be made.
1. This version of the Habrias, Dunne and Stoddart method does not preserve the distinction between label types and entity types. This is usually regarded as an important distinction (ISO 1982) and the mapping procedure loses this information.

2. The grouping procedure suggested is based on the relational model hence it is targeted toward a particular implementation. If a specification in Z is intended to be a statement of what is required and not how it might be implemented then the grouping procedure which results in relational tables as a preliminary step violates this principle. It further assumes particular implementation goals related to reducing the total number of tables and reducing (though not eliminating completely) the number of null values. If different goals are set or a different type of database environment is assumed then the grouping algorithm is no longer appropriate and presumably some other algorithm would take its place. The result in either case is a different formal specification which is implementation dependent.

3. It is well known that under certain conditions, the grouping procedure may yield alternative but equally valid sets of relational tables from the same fact model schema. Hence, the grouping procedure may yield alternative but equally valid Z schema from the same fact model. For example, where there is a uniqueness constraint spanning each role in a binary fact type such as in figure 4 below. It is possible to regard the car allocated as an attribute of person or the person to which a car is allocated as an attribute of car. In this example, both car and person will still be represented as schema types. However, if car had no other facts attached to it (other than its identification scheme) and it is incorporated as an attribute of Person, the object type Car or alternatively its identification scheme of reg# will become a basic type. This simple example reflects the many possible formal schema representations in Z can occur.

4. Consider the results obtained using Habrias, Dunne and Stoddart method. The fact model began with three entity types and four label types. The final Z schema resulted in

- one entity type becoming a schema type (Student)
- two entity types each becoming basic types (COUNTRY and DATE)
- two label types each becoming basic types (NAME and StudentNumber which became N)
- one label type disappearing altogether (country_name)

Each entity type appears once as either a schema type or basic type whereas label types may or may not appear at all. It is not clear what is to become a basic type and what is to become a schema type or variable. What is it that should be regarded as a domain and what is that should be regarded as a variable which draws its values from a domain?

**Method 2**

The second method by Habrias, Dunne and Stoddart differs from the method above in that the distinction between label and entity types is preserved. This is accomplished by declaring an object type as a schema and its label representation as the component e.g.

```
COUNTRY
    countryName: COUNTRY_NAME
```

similarly for Date

```
DATE
    date: DATE_STRING
```

The preliminary grouping procedure is performed as before⁹.

```
(NAME, DATE, COUNTRY_NAME, DATE_STRING)
```

```
COUNTRY
    countryName: COUNTRY_NAME
```

```
DATE
    date: DATE_STRING
```

```
STUDENT
    StudentNumber: N
    StudentName: NAME
    EntryDate: Date
    BirthDate: Date
```

⁸ See, for instance, the relevant sections in Halpin (1995) and Nijssen and Halpin (1989)
⁹ The liberty of making two minor corrections for the schema for IUT_of_Nantes was made. This however does not make any difference to the essential ideas being presented.
Two comments can be made:

1. This method distinguishes between entity and label types and makes each label type into a basic type and each entity type into a schema type.

2. Because of the grouping procedure the method is still implementation based and this again means that there are potentially many equally valid Z schemas which are possible.

A comment which applies to both the first and second method is that all entity types in the examples have an identification scheme which is a simple 1:1 mandatory bridge type between each entity type and label type e.g. a student is identified by their unique student identity number. This allows a blurring of the distinction between the entity and its label so that one can see the two as, more or less, interchangeable. However, sometimes there may be more than one way to uniquely identify elements of an entity type (e.g. a person might be identifiable by their employee number, their tax file number and their drivers licence number). As well, identification schemes may be complex such as where two or more bridge types may be required in the identification of an entity type or fact types may be involved. These issues related to alternative or complex identification schemes from the ORM schema to Z do not arise in the examples and are not addressed in the paper.

In conclusion, the work of Habrias, Dunne and Stoddart (1993) is a useful exploration of approaches to a translation procedure from a fact model to Z. However, they have not provided systematic translation procedures and their examples have dealt with simple situations only.
Nguyen’s Method

Introduction
The first paper by Nguyen on this topic (Nguyen 1994) concerns itself with the mapping of fragments of the fact model into formal representation in Z. This work is superseded by a second paper on this topic by Nguyen and Duke (1995) where the earlier work is modified and placed in the context of a larger elicitation and analysis method, an overall structure for coping with larger UoDs using state schema in Z is proposed and some dynamic aspects are considered. Hence the second paper will be the basis of the discussion here.

Nguyen and Duke (1995) describe their approach as a “formal analysis method” the aim of which is “to enable the modeller to carry out the analysis leading to a complete and unambiguous model in a constructively effective fashion”. This method covers both static and dynamic aspects of a specification. The method incorporates the NIAM method of information analysis for eliciting and modelling static aspects of the UoD and then describes how to map the schema obtained into a formal specification in Z. It also suggests how the specification could be structured in order to make it more manageable and useful. The manner in which dynamic aspects of the UoD might be elicited and modelled (graphically) are not part of the method. Based on the definition I gave earlier as to what constitutes a method, Nguyen’s approach is not truly complete. However, to put the approach in context, it should be noted that Nguyen’s method is intended for systems which are data intensive. The inference appears to be that the dynamic aspects are of lesser importance in the specification and can be handled in an ad hoc way.

Steps
The steps involved in the approach by Nguyen and Duke (1995, p 96) are:

1. a) Perform the usual NIAM analysis and develop a NIAM data model
   b) Separate the data schema into one essential data schema and zero or more detailed data schemas
2. Introduce the Z state variables and the UoD’s state schema
3. Analyse and specify at least all the non-trivial operations
4. Modify the essential schema, if necessary, to reflect the essential character of the system (based on the newly acquired knowledge).

Nguyen’s mapping procedure assumes the n-ary version of ORM in which fact types can be of any arity. Because this also is the version of ORM used in this thesis (as will be seen later) this work has particular relevance.
**Overall Structure of the Formal Specification**

Specification of UoDs of any significant size are difficult to comprehend if they are presented as one monolithic block. In order to make the specification more understandable Nguyen and Duke make use of state schema to break up the specification into “essential schema” which describe the “big picture” (or what I would describe as key features) and associated with each of these state schema are zero or more “detailed schema” which describe aspects which “fill in” the details. This addresses the issues of *constructability* (i.e. handling complexity and large sets of facts) and *ease of analysis*.

Deciding which aspects to put into essential state schemas is largely arbitrary. The analyst (presumably in conjunction with user experts) must decide which aspects of the specification are key or essential ones. Nguyen and Duke suggest two possible approaches to making this decision, the first requires judgement about the key operations or events for the UoD. For example, in an academic institution enrolling a student might be a key operation but finding a student’s address might not. The fact types associated with these key operations make up what they call the “essential data schema”. The second approach looks at preconditions of operations. Preconditions decide whether an operation will proceed or not. The set of fact types involved in determining whether preconditions are met or not can provide an alternative way to determining an essential data schema.

**Mapping from a Fact Schema to Z**

The steps in Nguyen and Duke’s mapping procedure to Z are as follows

1. Identify the basic types
   
   Each label type is converted into a basic type. Entity types are included by stating their equivalence to these declared basic types. An example given describes orders and order lines. The fact schema is shown below.
The basic types declared are [ORDER#, LINE#]

Entity types are introduced as equivalences

\[
\text{ORDER} \cong \text{ORDER#}
\]
\[
\text{ORDER\_LINE} \cong \text{ORDER} \times \text{LINE#}
\]

Note that ORDER\_LINE is identified by the combination of line# and through its association with an order. An order line therefore also requires the identifier for order (order#). Even so, Nguyen and Duke point out that an order line with order# 2 and line# 4 does not automatically belong to order with order# 2 unless this is built in as constraint in the schema.

2. Binary fact types are introduced as partial functions (if a uniqueness constraint spans only one role) or as Cartesian product (if the uniqueness constraint spans across both roles).

3. For fact types of arity greater than two there are several possible translations. This can be illustrated using the diagram below.

\[
\text{TIME} \leftrightarrow \text{(RESOURCE} \leftrightarrow \text{USER)}
\]
\[
\text{RESOURCE} \leftrightarrow \text{(TIME} \leftrightarrow \text{USER)}
\]
\[
\text{USER} \leftrightarrow \text{(TIME} \leftrightarrow \text{RESOURCE)}
\]
\[
\text{(TIME} \times \text{RESOURCE)} \times \text{USER}
\]
An Example

In order to comment on the method, a sample NIAM schema used in Nguyen’s paper is duplicated together with the final result:

![Diagram of NIAM schema](image)

Figure 7

All label types are converted into basic types:

\[ \text{[STUDENTID, UNITCODE, ADDRESSDESC, GRADECODE, NAME, TITLE]} \]

We note then those label types which are used as references (for identification) of entity types. In this example, the following equivalences are then made:

\[
\begin{align*}
\text{STUDENT} & \triangleq \text{STUDENTID} \\
\text{UNIT} & \triangleq \text{UNITCODE} \\
\text{ADDRESS} & \triangleq \text{ADDRESSDESC} \\
\text{GRADE} & \triangleq \text{GRADECODE}
\end{align*}
\]

These equivalences indicate the manner in which these entity types are identified.

The state schema which define the static structure of the specification are as follows:
Comments

1. I note first that Nguyen and Duke’s method does consistently handle label and entity types in translating from the fact model schema to Z.

2. Although the example provided only includes simple reference schemes with a one to one mandatory fact type between the entity and label types, the translation procedure for more complex schemes (e.g. where another fact type is required in order to identify an entity type) is mentioned in the paper’s appendix.

3. Nguyen and Duke describe a formal analysis method which makes use of NIAM and the resultant fact model schema to handle the data analysis component of the method. It does provide guidance in how to translate the fact model schema into Z.

4. A fact based representation in Z is chosen rather than adopting a record based or object oriented based representation. It is justified on the basis that record and object oriented based representations are implementation biased and a requirements statement should avoid implementation decisions.
However, I have criticisms or questions regarding the method.

1. Nguyen and Duke’s guidance in translating the fact model schema is limited and considers only the simpler situations.
2. The method converts all label types in the fact model schema into basic types in Z. What is the basis for this? Is this really suitable in all circumstances?
3. In the Z schema the initial basic types are derived from the fact model label types. Then basic types representing fact model entity types are introduced which are made equivalent to the basic types representing label types. However, my impression of the way that fact modelling is performed in practice (based on experience of observing fact model schemas created by students which include industry practitioners in data modelling) is that there is more care in the choice of entity types (since they represent the key concepts) than in label types. If the procedure was followed slavishly or carelessly, there would be some potential problems.
   (i) A schema could include several different instances of a label type NAME that are attached to different entity types. In this circumstance, Nguyen and Duke’s translation procedure would result in only one basic type. The entity types to which they are attached when then be made equivalent they would result in them being the same basic type.
   (ii) If there are various different types of names (student name, employer name and so on) should the label types be translated into different basic types and therefore not be comparable? How are the fact model entity types translated to which they are attached?
4. Following on from the previous point, I can point out that the translation procedure does not preserve the distinction between entity and label types which are incorporated in the fact model. There is therefore a loss of information.

**Summary**

In this chapter, I reviewed the limited literature which exists regarding translation procedures from the fact model into Z. The paper by Habrias, Dunne and Stoddart (1993) is essentially only an exploratory paper but does indicate some possibilities while Nguyen and Duke’s (1995) the translation procedure is limited in scope and considers only the simpler situations.

None of the approaches provide much justification for their translation procedure from the fact model to Z. All are largely presented as a fait accompli. In order to have some confidence
in a translation procedure there has to be a rationale behind why things are done as they are. It is suggested that the procedure has to have a conceptual basis on which the translation is made. In order to judge the procedure is suggested that the principles for the creation of conceptual schema as outlined by Loucopoulos and Zicari (1992) provide some guidance here. Furthermore, a translation procedure should be complete so it should work in complex as well as simple situations and be comprehensive enough to cover fact types of any arity and all standard constraints used in the fact model. The purpose of this thesis is then to provide a procedure based on some conceptual basis for the translation and which translates all the information contained in the fact model. I will then evaluate it to demonstrate that it satisfies the principles for the creation of a “good” conceptual schema.
Chapter 4

Selecting a Fact Model Approach

Introduction

There are essentially two forms of representation of fact types in fact modelling. These are the n-ary and binary approaches. In the binary model approach there are two further subdivisions into the “pure” binary and pseudobinary models. In this chapter I describe the various forms and explain the reasoning behind why I have chosen the n-ary form of representation over the binary forms.

The N-ary Model

The elementary n-ary approach to representing fact types allows for the expression of fact types in what is described as their “natural” form and are demonstrated by Nijssen and Halpin (1989), Halpin (1995) and in an earlier form by Falkenberg (1976). Fact types are expressed as sentences which provide information about one or more objects of interest. For instance, consider the sentence patterns which follow

Person … jogs
Person … plays Sport …
Person … plays Sport … on Weekday …

Each ellipsis (indicated by …) provides a “hole” into which an appropriate object (actually its label) can be placed in order to provide a fact relevant to the universe of discourse. The word preceding each ellipsis indicates the type of the object that follows. The number of ellipses indicates the arity of each fact type. In the examples above we have unary, binary and ternary elementary fact (or sentence) types respectively. The n-ary form allows facts to be expressed in what is asserted to be a more natural form of expression and, when a fact model schema is drawn, makes the diagrams easier to read.

The Binary model

The binary approach requires all fact types to be in expressed binary form. This approach is demonstrated by Wintraecken (1990), Verheijen and van Bekkum (1982), ISO reports (ISO
1982; ISO 1985) and elsewhere. To illustrate the binary approach I will consider the three sentences with holes given previously and the appropriate binary form.

The unary fact

Person … jogs

could be replaced by

Person … has a Jogging_status …

where Jogging_status could have the boolean values such as “yes” or “no”.

The binary fact

Person … plays Sport …

is already a binary fact type so we leave it as it is.

Ternary or higher arities must be converted into binary form. The process is straightforward and involves treating fact type or part of the fact type as an object type (this is called *objectifying*) and relating it back to the original components via one or more new fact types. This can be handled in different ways using either the pure binary model approach or the pseudobinary model approach which are described in the following.

In the pure binary model, components of a fact type are “paired” together and treated as an object to be connected to remaining components. If we take a ternary fact type and represent the three roles as A, B and C which we write as ABC, there are three possible groupings:

\[(AB)C \quad A(BC) \quad (AC)B\]

One of these would need to be chosen in order to represent the original fact type ABC.

Using the example given earlier we might arbitrarily objectify “Person plays Sport” and call it “PersonalActivity” and then state

PersonalActivity … is played on Weekday ….

In the ternary case the number of possible groupings is relatively small, however examination of the possible groupings of a quaternary fact type indicates how quickly the number of possibilities can proliferate as the arity of the fact type increases. If, as before, we imagine a quaternary fact type with the roles A, B, C and D, the possible groupings such that a set of binaries is the result is shown below:
The question that might be asked is whether there is a “correct” or best representation. It may be that there is no suitable grouping i.e. no one actually makes use of or perceives any particular grouping or it may be that some individuals have a preference for one grouping and others for a different group. Some of the objectified fact type components therefore might exist in the schema simply to satisfy the requirements of the binary model and be without any basis within the UoD.

Those who advocate binary modelling, such as Wintraecken (1990), argue that the grouping process is an important “discovery” procedure because, by considering groupings, one can discover something more about the fact type. While we can see some truth in this argument, sometimes the argument might be trying to make a virtue out of necessity. If there is a software package into which the schema is going to be drawn that only handles binary fact types then one must conform to the requirements of the package. If there is some formal mathematical model underlying the representation of a fact model which assumes binary relationships (maybe because it is a particularly elegant or useful approach) then again one may feel compelled to conform the model to the formal representation. However it would seem preferable to have the decision based on some guiding principle to do with requirements specification or analysis rather than other needs.

**The Pseudobinary model**

An alternative to the pure binary model is the pseudobinary model (Kent 1978; Wintraecken 1990). In this approach, the entire n-ary fact type (where n is greater than two) is objectified and then n binary relationships are created between the objectified relationship and the entity types participating in the original n-ary fact type. For example, if there is a ternary fact type with roles connected to entity types P, Q and R (which may be the same). An objectified relationship X representing the ternary fact type is created as are three binary fact types connecting X to P, X to Q and X to R. For a quaternary fact type connected to entity types P, Q, R and S we would create an objectified relationship X and create binary fact types XP, XQ, XR and XS.

To demonstrate the pseudobinary model we can use the previous example again. The fact type

“Person … plays Sport … on Weekday …”
is objectified to, say, “Booking”. This is related to the three entity types in the original fact type as follows

   Booking … is for Person …
   Booking … relates to Sport …
   Booking … is on Weekday …

As was the case with the pure binary model there is no guarantee that the new objectified fact type has particular significance in the UoD other than being a fact type.

If an n-ary model has been created but a binary model is required, then the pseudobinary model provides a consistent approach to translation into the binary form. However, the final model created contains a mixture of entity types some of which are atomic (e.g. Person, Sport and Weekday) and some which are objectified fact types (e.g. Booking). To anyone looking at the final model, it would not be obvious which entity types were atomic and which were not.

Discussion

When looking for some straightforward, unambiguous translation procedure then we want to avoid the problem of having to decide which object types were the truly atomic ones and which were the (artificially created) composites.

I may have presented a picture of the n-ary and binary models which might suggest that things are rather “black and white”. However, the process of developing a fact model is an individual one and there will be argument as to what is perceived and what should modelled. In some cases, what one person regards as a fundamental (atomic) concept some one else may regard as a composite of other concepts. This is to be expected. I give a simple example to illustrate this.

Example

Consider a situation in which we are recording events about lightning strikes. We want to record when it occurred, where it occurred, its severity (measured by an integer) and the damage it caused (measured in dollar terms). Should a lightning strike be modelled as an entity type?
One model of this could be as two ternary sentences as shown below.

![Diagram](image)

**Figure 8**

In this model, there is no entity type for the lightning strike but it does exist within the sentences which go with the ternary fact types.

On the other hand, we may want to have a lightning strike as an entity type as modelled below.

![Diagram](image)

**Figure 9**

How would we identify each lightning strike? We could (artificially) generate a number or code for each one e.g.

“Strike 33 caused damage of amount $1000”
However, this would really only be legitimate if the users in the UoD agreed to this. Alternatively, we can “borrow” the identification scheme of Location and Time so that a lightning strike is identified by the combination of a location and time.

In this example we see that in one model a concept is specifically modelled as an entity type while in another it exists only within a sentence description (assuming that the analyst makes the effort to write sentences in a reasonably meaningful way). There is no judgement here about correctness. These all correctly represent the UoD.

In representing a particular UoD, the role of the analyst is to represent it using the concepts of the users. The example above illustrates that in representing the UoD, there is scope for a variety of satisfactory representations using the n-ary model approach. If concepts that the users are familiar with are used then the users have a chance of being able to understand and to validate the model. This is why it is preferable to use the n-ary model to represent the UoD. It produces a model of the UoD that is more in keeping with the users view. This does not preclude the analyst negotiating with the users about a satisfactory view of the UoD especially if inconsistencies or ambiguities or other problems arise. However, if concepts are introduced artificially (which is what can happen with a binary modelling approach) then the model becomes more difficult to interpret and hence validate. To an independent observer, it will not be obvious which entity types have a genuine interpretation in the UoD and which are artificial.

While we could produce an n-ary based fact model as an intermediate step and then turn it into a binary one for the purposes of translation to a Z schema, the argument that I would put is that it still creates artificial concepts in the Z model. This is still more difficult to validate than one based on the n-ary based model. Furthermore, if refinement of the Z schema takes place or some other manipulation then the task of relating the schema back to the UoD is going to be more difficult.

The translation procedure which is described in later chapters can handle either n-ary or binary forms of representation because it is designed around the n-ary form. However, the means by which we represent fact types of any arity using the one consistent approach using schema types wouldn’t necessarily be the best way to represent fact types if only binary fact types were to be considered.

**Summary**

I have described the binary and n-ary forms for representing a fact model. I have argued that the binary form introduces concepts (represented as entity types) that are not familiar to the
users and hence these artificial concepts have no place in the fact model schema. If the n-ary form represents the UoD in a way which is a closer to the way it is viewed by the users it should be easier to validate. When translating the fact model into Z the intention is to preserve this representation as closely as possible for the same reasons.

The translation procedure which is described later can handle the binary form of the fact model but may not be the most suitable translation and would need to be reconsidered.
Chapter 5

Mapping the Fact Model into Z

Introduction

This chapter establishes a mapping between the concepts which form the basis of the fact model and the Z language. I describe the fundamental concepts of the fact model and then examine Z to determine to which Z concepts they might most appropriately map. I provide arguments for representing fact model entity types as Z basic types and representing fact model fact types as Z schema types. In the next chapter, I provide a translation procedure for the various standard constraints used in the fact model that apply both within and across fact types.

The aims I have in mind with the translation procedure are to develop an approach which

- Is straightforward so that it is easy to apply
- Preserves the semantic integrity of the fact model
- Covers all the standard elements (both fact types and constraints) that we are going to encounter in a fact model schema
- Is prescriptive enough so that when applied it is consistent i.e. not arbitrary
- Results in a Z schema which can be related back to the UoD thereby making it readily able to be validated

The first three aims will ensure that the translation procedure could be automated in a straightforward manner should the need arise.

I justify these aims by making reference to the criteria for judging a conceptual schema (appendix A). In particular, these are

- Constructability – We want to preserve easy communication between analysts and users and this is accomplished if we preserve the concepts of fact types, natural language sentences and meaningful names.
- Completeness – we want to do this in as far as all standard elements of the fact model schema should be included in the translation procedure into the Z schema
- Traceability – this is aided by having a translation process which is consistent and not subject to arbitrary choices or decisions
• **Executability** – this refers to the ability to simulate the specification against facts relevant to the UoD. This is aided if the UoD concepts are preserved in the translation.

• **Ease of analysis** – We require completeness and correctness as far as the Z schema should be able to be cross checked against its corresponding fact model schema. Furthermore this is aided by traceability above.

**Background**

I am aware of two broad views about Z. One view is that Z is a language and in that view the task is to establish an interpretation of the language which allows us to suitably represent fact model schemas. A different view is that Z is essentially just set theory and predicate calculus and in that view the task is to map the concepts relating to the fact model into the language of sets and logic. I have chosen to view the process as the latter for two reasons. The first reason is that the literature relating to this area of transforming (semi formal) models to Z specifications appears to have assumed this view and so this work would be in keeping with that literature. The second reason is that the concepts involved in set theory and predicate calculus fit rather naturally with the concepts of the fact model and so we don’t have to establish the mapping ideas from first principles so to speak\(^{10}\).

Although this work is not primarily aimed at providing a formal definition of the Fact Model it is worth mentioning earlier work in formalisation of the model. Attempts at formal definitions for the Fact Model (NIAM) were made by Creasy (1983), De Troyer (1986) and Sernadis et al. (1989) but they were either not complete or provided only a semi formal definition. Complete and formal definitions were provided first by Halpin (1989) and later by van Bommel, ter Hofstede and van der Weide (1991) and De Troyer (1994). De Troyer (1994) provides a formalism to the binary model but as the reader will recall I have rejected the binary model for reasons given earlier. The work of van Bommel, ter Hofstede and van der Weide (1991) is based more on describing fact models in terms of relational database theory which didn’t appear particularly relevant to this work. Halpin (1989) provides a formalisation of n-ary fact modelling. Halpin’s (1989) work is based on first order predicate calculus but is tailored for knowledge work. It appears to be the first formalisation providing a rigorous model theory and proof theory for fact modelling.

\(^{10}\) Martin (1999) states that the second view of Z (as a combination of set theory and predicate calculus) can also be viewed as one interpretation of the Z language anyway so there does not appear to be a contradiction between the views. However, it does affect the language used in discussions and
If I trace back to determine from where these ideas for the translation procedure to be described here emerged then I would say that the works which were most influential to the approach were Halpin (1989) and the various works of Polack et al., Habrias et al. and Nguyen et al. that suggested translations from data models into Z to which I have referred in earlier chapters.

**Criteria for a Mapping Process**

In trying to transform from a fact model schema to Z based schema I saw the first task should be to establish a framework such that the fact model concepts are assigned to the mathematical concepts in a consistent manner which maintains the semantic integrity of the fact model. I distinguish **mapping** from the **translation** process. Translation is the mechanics of taking symbols that represent “things” of interest in a fact model schema and changing them into corresponding symbolic representations in a Z schema. Mapping is about determining what should be the correspondences. Mapping is about relating one or more concepts in one system to one or more concepts in another system. This distinction was made by Loucopoulos and Zicari (1992, p6) and earlier by Loucopoulos et al. (1987). I admit that this is a task that may be somewhat subjective in nature. However, to reduce subjectivity where I can, I have drawn from the literature (both Fact Modelling and Z) to justify the correspondences that I have made so that it does not appear to be entirely my own subjectivity.

We saw in the work by Habrias, Dunne and Stoddart (1993) and Nguyen and Duke (1995) that the same types of Fact Model concepts (for example an object type or a fact type) were implemented in different ways in Z. I would expect this in different mapping approaches. Sometimes, however, within one approach there might be an inconsistent treatment of the same type of concept. For example, fact types of different arity might be represented in different ways or object types were translated (or not) in what appeared to be an arbitrary manner. These are not consistent with the aims for a translation procedure mentioned above.

**Fundamental Concepts of the Fact Model**

Both Z and fact modelling (and NIAM) are well described in the literature and I do not intend to describe either of these in this thesis. However, with regard to the fact model I do need to introduce the concepts to the reader and establish the terminology since there is no universally so it is worth establishing with the reader my viewpoint so as to avoid potential confusion or
accepted standard language. Accordingly, the following passage presents some of the fundamental concepts of the fact model with the relevant terms as used in this thesis in italics. This is not intended to be a definition of the fact model although I have tried to be as precise and unambiguous as possible.

Within the UoD we observe that there are *objects* of interest that we wish to represent in the schema. These objects can be arbitrarily classified into *object types* as deemed appropriate by the observer.

Object types are sub-classified as either *labels* or *entities*. Entities are conceptual and exist within the mind of the observer. Labels are symbols (typically names, numbers etc) which are used to refer to the entities from the UoD.

Each and every entity is classified as one and only one particular *entity type*. The set of all entity types is capable of classifying all objects of interest in the UoD. All labels are classified into at least one label type and hence may classified under several label types.

A *fact* describes a property of an object or a relationship between two or more objects. Facts about the UoD are stated in the form of elementary sentences. An elementary sentence is one that cannot be decomposed within the scope of the application domain without loss of information occurring. Facts which have the the same pattern are classified as belonging to the same *fact type*. A fact type is a pattern in the form of a sentence with “holes” into which the objects which make the sentence true can be inserted.

An *idea* describes a property of an entity or a relationship between two or more entities. Ideas which have the the same pattern are classified as belonging to the same *idea type*. Each “hole” in an idea type accommodates only one object and each hole accomodates objects of the one object type only. Idea types can be populated only by use of labels which represent the entities.

A *bridge* represents a relationship between an entity and a label. Bridges which have the the same pattern are classified as belonging to the same *bridge type*. 

---
Mapping to \textit{Z}

In this section I establish two mappings. I argue that the entity types in the fact model maps closely to the basic types in \textit{Z} and that the idea type of the fact model best maps to schema types in \textit{Z}.

Objects and Object Types

Objects represent things observed in the application domain which we wish to model in the conceptual schema. The NIAM methodology does not begin with finding objects and classifying them into object types but rather with the statement of elementary facts about the application domain. For example we might state

\begin{quote}
The person with employee number 12345 is employed by the company with company name XYZ
\end{quote}

The parts in italics are the described objects. Note that each italicized part consists of three components: the object type (eg. person, company) which classifies an object, the reference mode which indicates how we are referring to the object (eg. employee number, company name) and the label (eg. 12345, XYZ) which refers to the object. Facts about the application domain which have a common structure are grouped together into classes of facts called “fact types”. The common structure we might see in this example is

\begin{quote}
Person … is employed by Company …
\end{quote}

This is described by Nijssen and Halpin (1989) as a “sentence with holes”. The holes provide a point into which relevant objects can be inserted. The process of eliciting facts and generalising them into fact types eventually leads to a structure into which any relevant fact from the application domain (both now and into the foreseeable future) can be accommodated.

In the process of eliciting elementary facts, unique identification of described objects is not required. However, it should be clear in the mind of the specifier that two or more objects described (whether concrete or abstract) are distinct and different within the context of the application domain even though with the current reference mode they may have the same referent. Temporarily the specifier might do this by tagging the observed objects. This ‘tag’ is not a value which exists in the application domain (such as an employee identity number which identifies different employees) but is purely a convenience. It is quite possible prior to
completion of a fact model that two different objects have exactly the same properties and could not be distinguished. However, because NIAM uses a value based identification scheme the NIAM method requires and provides techniques for resolution of this issue.

The mathematical formalism for the fact model schema proposed by Halpin (1989) is based around predicates and does not specifically include object types. Predicates may range over the entire domain of objects. However, in developing a fact model schema, it is natural to subdivide the domain into conceptual types of objects. Each role is restricted to objects of a certain type. In both the description of the NIAM procedure by Nijssen and Halpin (1989) and Halpin (1995) and implied by the graphical notation the concept of an object type is clearly assumed. There is a clear and precise description of an object type. We have for instance the following quotation from Nijssen and Halpin (1989, p37):

“"A type is a set of all possible instances. Each entity is an instance of a particular entity type (e.g. person, subject). For a given UoD, the entity type Person is the set of all people we might possibly want to talk about during the lifetime of the information system."

Further from Nijssen and Halpin (1989, p58) we have the rule

“.. in selecting entity types we should ensure they are mutually exclusive, that is, they have no instances in common."11

It is suggested within the NIAM procedure that where it is discovered that there are common instances across two object types then the two types should be collapsed into one. Further, where the object types have similarities, such as the same unit of measurement (length, money etc) so that they can be meaningfully compared or it is noticed that they participate in the same fact types such that they appear to have very similar (static) properties again we might consider collapsing the types.

The conclusion is therefore that a strong typing rule relates to objects. By not including object types in Halpin’s (1989) formalism, I believe that he omits one very important constraint that applies in NIAM which should be included. I have assumed strong typing of entities as a requirement of the fact model.

The mathematical concepts and structure of Z is well known and documented for example Spivey (1989). The manner in which one writes a specification depends on what the specifier

11 Note that subtypes in NIAM may contain only instances of their root type and the statement is made in that light
chooses to “observe” in the UoD (what are the objects and relationships) and the manner in which the specifier chooses to represent it. There is a great deal of scope in terms of the manner of representation. The specifier may interpret that a basic type in Z shall represent a collection of objects or he may interpret a set variable to represent the objects or possibly a schema type. In a UoD with objects such as stock, customers and shipments it is conceivable that shipments could be represented as basic type or a set (based at least partly on customer and products) or as a schema type (also based at least partly on customer and products).

Very little literature related to Z is explicitly devoted to the topic of how one might go about best representing an application in Z. There are no rules as such other than the expectation that the specification correctly represent the UoD and be internally consistent.

The set of basic types (also called given sets or basic sets) will determine not only the mathematical statements but also the ways in which the mathematical structure of the model can be developed. Selecting appropriate basic types would seem to be an important part of creating a Z specification. Hence, I begin by considering how basic types (also called given sets or basic sets) could be interpreted in Z.

Consider how basic sets are different to other sets which may be defined in a Z specification. Spivey (1989; 1992) is often cited in the literature as the de facto standard on Z, states:

"Every Z specification begins with certain objects which play a part in the specification, but have no internal structure of interest. These atomic objects are the members of the basic or given sets of the specification."

later

"From the atomic objects, composite objects can be put together in various ways .... There are three kinds of composite types: set types, Cartesian product types and schema types."

Spivey, while providing the mathematical framework (Spivey 1988, Spivey 1989, Spivey 1992), does not provide any advice or methodology as to how basic and other types should be decided or constructed other than what might be implied in the examples given within the text. This is not an uncommon finding in the literature describing Z.

Wordsworth (1992, p28) states that

"it is difficult to give general rules about how to recognize the (basic) types from informal requirements ... we shall teach type selection mostly by example".

Potter, Sinclair and Till (1991) state
"These basic types ... are introduced by enclosing their names in square brackets ... We are not really interested in the internal structure of these sets : we care only that introducing them in a specification makes them available as valid types and that variables of these types can be declared."

The quotes given above represent what is typical in the literature. For some authors the process of selecting basic types of a specification is not an issue because the focus of their work is to demonstrate the mathematics. However, others who have considered the selection of basic types seem to have relied to an extent on a "magic leap" which suggests that the reader, on investigation of a particular situation, will intuitively know what the basic types should be.

A summary, gained from the literature, of what this author feels are the underlying ideas about the objects making up the basic types in Z specifications are the following: Basic types are sets which should consist of objects which are

1. atomic - The objects that make up a basic set are regarded as being so fundamental that nothing is to be gained by looking further into their nature i.e. they have no internal structure of interest (Spivey 1992).

2. homogeneous - The objects should all have a common property (Potter, Sinclair and Till 1991, p. 9) or stating it more loosely, they should be "all of the same kind" or "homogeneous" (Wordsworth 1992, p. 28)

3. mutually exclusive - The "objects...may be categorized into different kinds and ... there is no overlap between distinct kinds" (Potter, Sinclair and Till 1991, p. 9); "every object has or belongs to one type" and "no object can belong to more than one" (Edmond 1992, p. 316).

4. a maximal set - the basic type should be capable of incorporating all candidate members, for example "a type is a maximal set" (Edmond 1992) and it should form a "maximal set" (Woodcock 1989, p. 85).

We could also add the further requirement about the entire set of basic types chosen is that it must be sufficient to adequately\(^\text{12}^{12}\) describe the UoD.

\(^{12}\) adequately here we suggest means permitting the specifier to describe all the objects of interest and aspects of relevance
When we compare the description for object types with the descriptions for basic types in Z, very simply, they appear strongly similar to the earlier descriptions for entity types in the Fact Model. Based on this, the mapping that I will make based on this is that entity types discovered through fact modelling can be mapped directly into Z basic types\(^\text{13}\).

Having made this first mapping of concepts, I can make the claim that since NIAM can be used as a step by step procedure for the discovery and identification of a “good” set of object types we have now a way of determining basic types for a Z specification through the mapping. We can therefore replace the “magic leap” mentioned earlier with a systematic procedure. This could be a useful aid for those attempting to produce a Z specification.

**Facts and Fact Types**

Schema types have been used to represent entity types as based on the concepts of the Entity Relationship Model (ERM) or the Relational Database (Polack 1992; Polack, Whiston and Mander 1994; Polack, Whiston and Hitchcock 1992; Polack and Mander 1994; Edmond 1992). A schema type used in this way has the usual form (represented using vertical form below) of

```
  Schema Name
  declared variables
  predicate
```

For example, there may be an entity type such as “employee” about which we wish to know their id#, name, phone extension, and department. These would be grouped together in a schema type as shown below.

```
  Employee
  id#
  name
  phone
  department
```

A Z schema type can be used to declare a collection of elements which are then constrained in some way in the predicate section. The elements comprised in the schema type are named but

\(^\text{13}\) This will not be one to one exactly as the reader will see later.
not ordered and form what is called a schema binding. Hence, the order of declaration of elements is not important. Two schema types are deemed to be the same if they declare the same elements i.e. it is the names and types that distinguish one schema type from another. The predicate part of the schema type is not relevant.

In the Conceptual Schema Design Procedure of NIAM (Halpin 1995; Halpin 2001; Nijssen and Halpin 1989; Wintraecken 1990) the final product is a set of fact types which together represent all the data of interest about the UoD broken down into its smallest possible components without loss of any information in the context of that UoD. Procedures to group or combine fact types into larger constructions such as ERM entity types or RDB records are not part of NIAM. Furthermore, where supplementary procedures to develop the larger constructions are provided, the groupings are to an extent arbitrary and in some cases targeted toward a particular implementation (e.g. the relational database). A fact model schema can be regarded as more “stable” or fundamental. Hence, we preserve the concept of a fact type in conversion to Z.

I have also chosen to use schema types but in this case to model fact types of degree two or higher. Seeing that facts represent relationships between objects and that Cartesian product types are typically used to represent such relationships in Z, some justification for this choice seems necessary.

What characterises a fact type? It is not just the words joining the objects together in, say for example, the form of infix notation such as

... owns ...

In a schema there might be another different fact type using the same word owns such as

The person with name Bill Gates owns the company Microsoft
The person with name Fred owns the car with registration ABC123

What makes the two fact types different is the combination of object types allowed in conjunction with the infix notation. In this case,

Person … owns Company …
Person … owns Car …

are two different fact types because the former describes facts about a person owning a company whereas the latter describes facts about a person owning a car.
Another characteristic of fact types is that when a fact type is used to express a relationship among objects, there are different but equivalent ways of expressing it (Halpin 1995, p48-51) (Halpin 2001, p65). For example, the fact type

The person with name ... owns the car with registration number ...

could be expressed as

The car with registration number ... is owned by the person with name ...

From a linguistic viewpoint, these sentences are are said to display different surface structures but the same deep structure.

An alternative approach is to pair the object with its role in a relationship so that the order of representation becomes irrelevant. One could also have expressed the earlier fact type as

CarOwnership{
  (the person with name ..., owns car),
  (the car with registration number ..., is owned by person)}

An object type can appear more than once in a relationship and is said to play different roles in that relationship as in

Person … is parent of Person ….

While the example above deals only with binary facts the approach can be extended to higher order facts.

What are the properties of a fact type then? The underlying philosophy is that each fact type represents essentially just one type of relationship. This relationship can be expressed in a number of ways such that the roles played by the objects can potentially be expressed in any order but each way is regarded as being equivalent to any of the other ways.\(^\text{14}\) I wish to preserve the idea of the different roles of object types when the same object type appears more than once in a relationship and that any particular order is not truly important.

I believe that in representing a fact type as a schema type we are able to maintain the main features mentioned above. We do this by declaring a variable within the schema type corresponding to each of the roles within the fact type. Each variable is based on a Z basic type which corresponds to the Fact Model type of the object. Naturally, no two variables of the same type in the one schema type can have the same name.

\(^{14}\) This is not completely true. For example, while for many sentences connecting two objects there is both an active and passive tense which reverses the order of the objects and which are equivalent, this is not universally true.
Summarizing then

- A schema type will represent a fact type.
- Just as the combination of roles and not their order makes a fact type unique, similarly the schema type can be created with elements representing the roles and in a schema type the order is not relevant.
- Just as no two roles in a fact type are the same (although the object types they are based on can be the same) no two variables in a schema type can have the same name (although they too can be based on the same basic type).

The schema type, therefore, has intrinsic properties which match reasonably well with the properties of a fact type.

**Example**

For the fact type

Supplier ... supplied quantity ... of item ..

the schema type which would be created to represent this would be

```
+Supplier_supplied_Quantity_of_Item
  Supplier : SUPPLIER
  Quantity : QUANTITY
  Item : ITEM
```

In order to preserve some element of natural language I have chosen to name the schema type using a “natural language” sentence indicating the relationship between the variables. Admittedly this results in some rather long schema type names and in practice this may be clumsy. However, this does assist in traceability if the sentences used in the fact model schema match the schema names in Z schema.

When we wish to create a variable which can be populated, we would declare a new variable based on this schema type:

```
supplier_supplied_Quantity_of_Item : \P Supplier_supplied_Quantity_of_Item
```

Here I have used the convention of declaring schema type names beginning with an upper case letter and the corresponding variable with the same name but beginning with a lower case letter.
Reference to any values of any component of the fact type is done by using the dot selector notation ‘._._’ indicating the variable representing the fact type and the component in the form FACT_TYPE.Component

eg.
supplier_supplied_Quantity_of_Item.Item

refers the item component of the fact type supplier_supplied_Quantity_of_Item.

To specify the set of objects populating a particular role in a fact type we use:

{x : FACT_TYPE: • x.component}

eg.
{x : supplier_supplied_Quantity_of_Item • x.item}

refers to all the items populating the fact type supplier_supplied_Quantity_of_Item.

**Transformation Approaches**

I have so far considered the mapping of concepts in the fact model to Z. This has been at a high level. Now that we come to considering the detail of the mapping process, I have decided on two possible approaches. The two approaches arise because fact modelling separates entities and labels. When we connect an entity type to more than one fact or bridge type, we must create in our mind a surrogate entity. We must create the surrogate because all the labels (from the application domain) have been placed somewhere else i.e. in the label types. This surrogate is involved in multiple relationships and acts as an anchor connecting this same particular entity to just these particular relationships involving labels and other entities. For example, if we were to remove that particular entity then we would sever all the relationships in which this entity was involved. On the other hand, the fact modelling method (or NIAM or whatever other name is being given to variations of this approach) is pragmatic and recognises that all entities must be suitably identified and assumes a value based identification scheme. It tells us which bridge(s) and/or fact types (and ultimately through them other label types) can suitably identify an entity type and these can then represent the surrogate entity mentioned earlier.

The first approach considered is based on maintaining the idea of a surrogate for each entity to provide a connection point among the various labels and roles attached to that entity. In this approach each and every fact type (whether idea or bridge type) is translated into a schema type. When attempting to populate the Z schema, entities may need to allocated some
internal identification which does not necessarily have any direct relevance to the application domain. Labels which refer to the entities would still be inputs into the schema and the rules related to making sure that every entity could be uniquely identified would still apply provided that they applied in the fact model schema. Naturally, any variables corresponding to outputs from the Z schema would have to be in terms of the label types and not the surrogate values created to represent the entities.

The second approach assumes that we have considered the identification of entities and that a value based identification scheme applies. These values must be related to the application domain. Label types and idea types used to identify entities are “removed” from the fact model schema. Usually, this removes most bridge types and possibly some idea types. In this approach only the remaining fact types are translated on a one for one basis into a Z schema. When populating the Z schema, entities are referred to by the label type or combination of label types which have been identified as capable of referring to an entity without any ambiguity by the methods provided by fact modelling. Hence, we can be confident that all entities will be uniquely identifiable but the identification scheme does not play a part directly in the Z specification.

### A Simple Example - Approach 1

The conceptual schema below is adapted from (Nijssen 1989, p.267) to illustrate the basic ideas of the mapping to Z. Constraints have been deliberately omitted at this stage as they are not relevant to illustrating the approach. Constraints are dealt with in later chapters of this thesis.

\[\text{This is similar to an early step of the Rmap procedure (Halpin 1995) to remove all bridge and fact types used to identify an entity in the fact model schema as a preparation to producing a set of normalised relational tables.}\]
We first extract the basic types by writing down all the entity types.

[SUPPLIER], [ITEM], [COUNTRY], [QUANTITY], [DEPARTMENT].

We will also require the basic types [STRING] and [N] (which have their usual meaning) for the label types.

We now create schema types for each fact type (including all bridge types) in the schema.

Idea Types

[SUPPLIER, COUNTRY, QUANTITY, ITEM, DEPARTMENT, STRING, N]

```plaintext
Supplier_is_in_Country
Supplier : SUPPLIER
Country : COUNTRY

Supplier_supplied_Quantity_of_item
Supplier : SUPPLIER
Quantity : QUANTITY
Item : ITEM

Item_is_made_in_Country
Item : ITEM
Country : COUNTRY
```

The reader will recall that where a mandatory one to one relationship exists between an entity type and label type and this label type is to be used as the primary means of identification, we can use the diagrammatic shorthand of enclosing the label type with the corresponding entity type thus making the diagram less cluttered.
We can populate the Z schema with facts by creating sets based on the schema types:

```plaintext
Department_uses_Item
Department : DEPARTMENT
Item : ITEM

Bridge Types

Item_is_identified_by_Itemnr
Item : ITEM
Itemnr : STRING

Item_is_identified_by_ItemName
Item : ITEM
ItemName : STRING

Department_is_identified_by_Code
Department : DEPARTMENT
Code : STRING

Quantity_is_identified_by_Nr
Quantity : QUANTITY
Nr : N

Supplier_is_identified_by_SupNr
Supplier : SUPPLIER
SupNr : STRING

Supplier_has_name_SupplierName
Supplier : SUPPLIER
SupplierName : STRING

country_is_identified_by_Name
Country : COUNTRY
Name : STRING
```
A Simple Example - Approach 2

In this approach the identification scheme for each entity type is ignored. In this case, this is straightforward because each entity type has a simple identification scheme. The basic types will be [SUPPLIER], [ITEM], [COUNTRY], [QUANTITY], [DEPARTMENT], [STRING]. The basic type [N] is not required.

The schema types for each remaining fact type will be:

Idea Types

-[SUPPLIER,COUNTRY,QUANTITY,ITEM,DEPARTMENT,STRING,N]-

-Supplier_is_in_Country-
Supplier : SUPPLIER
Country : COUNTRY

-Supplier_supplied_Quantity_of_item-
Supplier : SUPPLIER
Quantity : QUANTITY
Item : ITEM

-Item_is_made_in_Country-
Item : ITEM
Country : COUNTRY

-Department_uses_Item-
Department : DEPARTMENT
Item : ITEM

Bridge Types

-Item_is_identified_by_ItemName-
Item : ITEM
ItemName : STRING

-Supplier_has_name_SupplierName-
Supplier : SUPPLIER
SupplierName : STRING

We can populate the schema by creating sets based on the schema types:
The definitions above create sets into which facts can now be inserted.

**Discussion**

What we notice immediately when comparing the two approaches is that the second approach (Approach 2) leads to a much smaller Z schema. Fewer schema types are required and consequently fewer variables based on those types. Even though I have not discussed constraints, there is a corresponding reduction in the number of constraints that have to be specified. This is already an advantage provided we have not lost anything of significance in the reduced schema.

I would argue that the missing parts in approach 2 (including constraints) are implied. They are implied because they relate to the identification of entities. In approach 1 all the extra schema types, the corresponding sets and the associated constraints (not covered here) all relate to aspects of the identification scheme of entity types. The label or combination of labels is considered to be the entity (or its symbolic representation) and is then ignored from further consideration. This allows one to concentrate on relationships between entities and operations on the entities within the application. The identification scheme can be reintroduced later when required.

How does this compare with what happens in practice with other Z schema? I have observed two ways of going about this. In the first way, when a basic type is introduced the label type identifying an entity is what is stated e.g. such as declaring [EMPLOYEE NUMBER] or [NAME]. That is, the label either “is” or represents the entity. This appears to be exactly the same as is done in approach 2. However, this equivalence is made with the approach in this thesis with the confidence that a suitable means of identification has been considered as part of the fact modelling process. In the second way, the matter of identification is not considered as an important issue. For example we could declare a basic type [CUSTOMER] and assume that later some suitable means of identification will be chosen to identify a customer. This is rather like dealing with surrogates to represent entities. It is possible to foresee that this could potentially lead to situations where a label type could be omitted.

---

17 Assuming due diligence is taken of course.
inadvertently (assuming that it would be used to identify an entity type and therefore implied but later some other identification means is used) or a label type could be redundant such as when we declare [CUSTOMER] and [NAME] and then decide later that a name can identify a customer. Furthermore, Z schema predicates might also need alteration in this case since they then might not precisely reflect the requirements.

It was decided, of the two approaches, to choose approach 2. There are two reasons for this:

1. The decision regarding the means of identification of an object is an analysis decision and not a design decision and so should be decided prior to drawing up a formal specification. The fact model schema allows one to see the various options available, discuss it with the appropriate stakeholders and come to a decision. In a Z schema, the identification scheme is often assumed and may not always be appropriate. This may result in possible problems with the specification. If it is not considered at all, this may lead to errors.

2. The Z specification produced is simpler using approach 2. The simplicity comes initially from having fewer basic constructs required. The specification will not have to include surrogate identifiers required to connect entities and labels to each other. For each entity type there will be a label (or set of labels) which uniquely identifies an entity and represents the entity.

I believe that approach two provides a specification for the UoD concentrating on the “real” entities and relationships. While many label types and some idea types used for identification schemes are not represented in the specification, they can be introduced when required. In Chapter 9 after describing my translation procedure, I provide a final step which introduces the identification scheme for each entity type into the Z schema without any substantial change to the specification which has already been created.
Chapter 6

Constraints

Introduction

In this chapter, I provide a translation procedure whereby standard constraints from the fact model schema are translated into Z. The procedure aims to be consistent in its style so that the constraint pattern for fact types any of arity have the same basic form. I would also describe the approach as intuitive in the sense that each constraint is considered separately and is written in a manner which is closely related to the way it is described in fact modelling terms. This has the advantage of making it easier to validate and trace the specification but the disadvantage that it may not be written mathematically in the most elegant or most concise way.

The translation of a constraint is described in the form of a pattern to be followed in which the user replaces the generic variables provided with the corresponding object types, fact types or roles from the fact model schema which is to be translated.

The full translation procedure and examples of substantial size and complexity illustrating the translation procedure are provided in the next chapters.

Naming Conventions

Here I define some naming conventions for variables that have been used in this chapter.

- O indicates an object type
- o indicates an object or set of objects
- F indicates a fact type
- f indicates a fact or set of facts
- r indicates a role played by an object or set of objects of a particular type

Integers are often used in the following as subscripts to differentiate objects, roles and facts. One exception to this is my use the subscript c in O_c which denotes an object type which is common (hence the subscript c) to a number of roles within a number of fact types.

Where double subscripts are used with facts, such as in f_{ab}, the interpretation is that f_a indicates the fact type while the second subscript is used to differentiates between different
facts within that same fact type. For example \( f_{a1} \) and \( f_{a2} \) are two facts (which may or may not be different) within the fact type indicated by \( f_a \).

Where double subscripts are used with roles, such as in \( r_{a,b} \), the first subscript typically indicates which fact type is being referred to and the second subscript the role within that fact type. For example, \( r_{a1} \) and \( r_{a2} \) are two particular roles within the fact type indicated by \( r_a \).

With regard to roles, each variable representing a role in a schema type is distinct and different from any other role. However, two or more roles of the same fact type may be of the same type of objects. In terms of naming roles, for the purposes of establishing conversions within this thesis, use of the same role name is allowed but not within the same fact type. Reference to a role name in a particular variable which is based on a schema type distinguishes that role from one with the same name in another. This is a similar approach to that used in relational databases where column names may be duplicated across tables but not within the same table.

**Total Population of an object type**

In the fact model the total population of an object type in a schema is defined as the union all the objects of that type that currently participate in some fact type (which may be unary, binary, ternary etc.). In practice this must always be a finite set. This is not the same as the concept of object type. An object type is used to classify objects of interest in the UoD. In the case of label types, it is possible to enumerate the objects that fall into that classification scheme since we are talking about labels which by definition we must be able to express in some way (customer number, integer, name etc). For entity types a definition provides the classification scheme since we are talking about concepts (customer, department, mass, invoice).

Consider all the facts in which objects of one type participate such as is illustrated in Figure 11. \( O_c \) indicates an object type common to the fact types \( F_1 \), \( F_2 \), ... \( F_n \).
\( r_1, r_2, r_3, \ldots, r_n \) refer to all the different roles within fact types played by objects which are all of the type indicated by \( O_c \).

Each fact type is first declared

\[
F_1 \quad r_1: O_c \\
\text{other object types participating in this fact type}
\]

\[
F_2 \quad r_2: O_c \\
\text{other object types participating in this fact type}
\]

\[
\ldots
\]

\[
F_n \quad r_n: O_c \\
\text{other object types participating in this fact type}
\]

In the following schema, the total population is defined

\[
\text{Total Population Schema for Object Type}
\]

\[
f_1: P \cdot F_1 \\
f_2: P \cdot F_2 \\
\ldots \\
f_n: P \cdot F_n
\]

\[
\text{objectTypePopulation: } P \cdot O_c
\]

\[
\text{objectTypePopulation} = \{ x:f_1 \cdot x.r_1 \} \cup \{ x:f_2 \cdot x.r_2 \} \\
\cup \ldots \{ x:f_n \cdot x.r_n \}
\]

In specifying the total population, it may be possible that two different roles, say \( r_x \) and \( r_y \), may belong to the same fact type. An example of this situation is a fact type such as

**Person ... was married to Person ... on Date ...**

In this case the corresponding variables \( f_x \) and \( f_y \) will be the same as will the corresponding schema types \( F_x \) and \( F_y \). The same idea extends to three or more roles belonging to the same fact type. The predicate part is not affected.
**Internal Uniqueness Constraints across Fact Types**

In fact modelling, for any fact type of arity\(^{18}\) \(n\) the uniqueness constraint must span at least \(n - 1\) roles or the fact type is not elementary and can be split into two or more fact types\(^{19}\). In order to discuss uniqueness constraints covering fact types of any arity \(n\), we consider a general case of a fact type \(F\) of arity \(n\). This is depicted in Figure 12. Each role is of one and only one object type although two or more roles may be of the same object type.

![Figure 12](image)

We represent the fact type \(F\) as a schema type and the roles as variables as shown as follows.

\[
F \\
\begin{array}{c}
r_1: O_1 \\
r_2: O_2 \\
\vdots \\
r_n: O_n \\
\end{array}
\]

where some or all \(O_1, O_2 \ldots O_n\) may be the same.

Assume that we have a variable \(f\) declared as power set the schema type \(F\)

\[
f : \mathcal{P} F
\]

For the case where the uniqueness constraint spans all \(n\) elements of \(F\), the uniqueness constraint is captured by the predicate

\[
\forall f_1, f_2 : f : f \bullet f_1 \neq f_2
\]

This constraint will always be true for correctly formed fact types.

---

\(^{18}\) the arity of a fact type is the number of roles participating in that fact type

\(^{19}\) A fact type with a uniqueness constraint spanning \(n\) or \(n - 1\) roles may still be splittable into two or more elementary fact types, i.e. this constraint is a necessary but not sufficient condition for a fact type to be elementary. However this is not relevant in the statement of constraint. The important issue is that elementary fact types will not have uniqueness constraint spanning less than \(n - 1\) roles and hence we need not consider these cases.
For the case where the uniqueness constraint spans \( n - 1 \) roles, let \( r_n \) represent the role in \( F \) in the schema above not covered by the uniqueness constraint being considered. While \( r_n \) happens to occur as the last element of \( F \) in the diagram above it could have occurred anywhere in the fact type.

In the binary (\( n=2 \)) case

we have

\[
\forall f_1, f_2: f \bullet \\
\quad f_1 \neq f_2 \Rightarrow f_1 \cdot r_1 \neq f_2 \cdot r_1
\]

In the ternary case (\( n=3 \)) we have

\[
\forall f_1, f_2: f \bullet \\
\quad f_1 \neq f_2 \Rightarrow f_1 \cdot r_1 \neq f_2 \cdot r_1 \lor f_1 \cdot r_2 \neq f_2 \cdot r_2
\]

In the general case

\[
\forall f_1, f_2: f \bullet \\
\quad f_1 \neq f_2 \Rightarrow f_1 \cdot r_1 \neq f_2 \cdot r_1 \lor f_1 \cdot r_2 \neq f_2 \cdot r_2 \ldots \lor f_1 \cdot r_{n-1} \neq f_2 \cdot r_{n-1}
\]

In order to make the specification easier to write and more intuitive, the shorthand \( U_i(f, r_n) \) is suggested. \( f \) is a variable based on a schema type \( F \) which has the variables \( r_1, r_2, \ldots r_n \) where \( n > 1 \). \( U_i(f, r_n) \) is equivalent to the predicate immediately above. In the context of this translation procedure, this shorthand would be used to describe a uniqueness constraint which covers \( n - 1 \) roles of a fact type represented by \( f \) and \( r_n \) represents the role not covered by a uniqueness constraint in the fact type.
**External Uniqueness Constraint spanning binary fact types**

In this section, we consider the relatively common situation of two or more binary fact types connected by a common object type where a uniqueness constraint operates across the roles based on a common object type. These constraints are indicated by a line joining the roles participating in the constraint and are denoted on the diagram by a circled \( u \) notation. Figure 14 illustrates this situation for two binary fact types.

![Diagram of External Uniqueness Constraint spanning binary fact types]

An interpretation of this constraint is that if a natural join on the binary fact types is performed by matching values for the roles attached to common object type, then the uniqueness constraints covers the sets of values indicated by the uniqueness constraint on the diagram.

For the case of two binary fact types

\[
\begin{align*}
F_1 & \quad r_c : O_c \\
r_1 & : O_1 \\
F_2 & \quad r_c : O_c \\
r_2 & : O_2
\end{align*}
\]
Schema for external uniqueness constraints across two binary fact types

\begin{align*}
f_1 &: \prod F_1 \\
f_2 &: \prod F_2 \\
\forall f_{1a}, f_{1b} &: f_1 \cdot \\
\forall f_{2a}, f_{2b} &: f_2 \cdot \\
(f_{1a}.r_1, f_{2a}.r_2) &= (f_{1b}.r_1, f_{2b}.r_2) \land \\
f_{1a}.r_c &= f_{2a}.r_c \land \\
f_{1b}.r_c &= f_{2b}.r_c \\
\Rightarrow f_{1a}.r_c &= f_{1b}.r_c
\end{align*}

For the general case where we have $n$ binary facts spanned by a uniqueness constraint:

![Diagram](image)

**Figure 15**

<table>
<thead>
<tr>
<th>$F_1$</th>
<th>$O_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_c : O_c$</td>
<td>$r_1 : O_1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$F_2$</th>
<th>$O_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_c : O_c$</td>
<td>$r_2 : O_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$F_n$</th>
<th>$O_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_c : O_c$</td>
<td>$r_n : O_n$</td>
</tr>
</tbody>
</table>
External uniqueness constraints occur frequently. It would be useful to avoid the burden of writing in full what is a long and tedious formal description of this constraint and also to make the specification easier to read. It is suggested that some shorthand notation be adopted which would achieve brevity and clarity. The notation suggested is

\[ U_EB(f_1.r_1, f_2.r_2, \ldots, f_n.r_n) \]

where \( f_1, f_2, \ldots, f_n \) represent binary fact types and \( r_1, r_2, \ldots, r_n \) represent roles within their corresponding fact types across which the external uniqueness constraint applies. We require that \( f_1.r_1, f_2.r_2, \ldots, f_n.r_n \) be the roles on the “other side” of an object type common to \( f_1, f_2, \ldots, f_n \). The subscript EB in \( U_EB \) is simply a mnemonic for “external” and “binary”. This would be equivalent to the predicate part shown in the schema above.

**Mandatory Role Constraints**

A mandatory role constraint requires that every instance of an object type must participate in the fact type indicated as mandatory. Figure 6.6 illustrates the binary case of a number \( n \) of binary fact types with one role from each fact type connected to a common object type i.e. they are all of the same type.
Assume that there is a mandatory role constraint on \( F_i \) on the \( O_c \) side

\[
\begin{align*}
F_1 & : \ O_c \\
r_1 \ : \ O_1
\end{align*}
\]

\[
\begin{align*}
F_2 & : \ O_c \\
r_2 \ : \ O_2
\end{align*}
\]

\ldots

\[
\begin{align*}
F_1 & : \ O_c \\
r_1 \ : \ O_1
\end{align*}
\]

\ldots

\[
\begin{align*}
F_n & : \ O_c \\
r_n \ : \ O_n
\end{align*}
\]

In order to describe the mandatory role constraint, we must first determine the total population for the common object type (\( O_c \)).

\[
\text{Total Population Schema for Object Type}
\]

\[
\begin{align*}
f_1 & : \ P \ F_1 \\
f_2 & : \ P \ F_2 \\
\ldots \\
f_1 & : \ P \ F_1 \\
\ldots \\
f_n & : \ P \ F_n \\
o_c & : \ P \ O_c
\end{align*}
\]

\[
o_c = \{f: f_1 \cdot f.r_c \} \cup \{f: f_2 \cdot f.r_c \} \cup \ldots \{f: f_n \cdot f.r_c \}
\]

The mandatory role constraint can be expressed informally in the following way:
“Every currently existing object in the total population of that object type also exists as an object in the population of the role associated with that object type having a mandatory role constraint.” In the schema below, $r_c$ is the role with the mandatory constraint

\[
\exists \text{ Total\_Population\_Schema\_for\_Object\_Type}
\]

\[
\forall x: o_c \cdot x \in \{f: f_1 \cdot f.r_c\}
\]

i.e. every object which is currently an element of the object type population is also an element of the role population.

In the general case, an object participates in an n-ary fact type $F$ with a mandatory role constraint on any role. A typical situation is shown in Figure 6.7.

The schema below describes the mandatory role constraint relating to the role $r_i$, associated with the object type $O_i$. It is similar to the binary case described above.

Assume that the fact type concerned is $F$ and that there is a set $f_t$ of type $F$ defined which records all current facts of that type. The roles in $F$ are $r_1, r_2, \ldots, r_i, \ldots, r_n$. Assume that $r_i$ is of type $O_i$ and that the set $o_i$ records all the objects currently participating in any roles of the type $O_i$. The mandatory role constraint on the role $r_i$ belonging to the fact type $F$ is defined by the predicate

\[
\forall x: o_i \cdot x \in \{f: f_i \cdot f.r_i\}
\]

The constraint above provides an explicit condition in $Z$ which enforces the mandatory role constraint.

In the literature on $Z$ this constraint is normally not applied in a direct way but implied by (as consequence of) other constraints in the schema. The typical approach is to declare a set variable which maintains the current values of that type. Before a fact is added, deleted or modified, it checked against this set variable. Therefore, the schema which describe the
dynamic behaviour of the system leading to addition, deletion and update of facts belonging to a particular fact type maintain the equivalent of the mandatory role constraint provided that the variables are appropriately initialised in the initial state schema.

A *disjunction of roles* can specified as mandatory on a fact model diagram by joining all participating facts to one point on the object type and placing a mandatory role dot there.

![Figure 18](image)

The most general case of a disjunctive mandatory constraint is that of fact types of any arity or combination of arities each with one or more roles participating in the disjunctive mandatory role constraint.

Suppose that \( r_1, r_2 \ldots r_n \) are distinct and different roles participating in a disjunctive mandatory role constraint. Although drawn on the extreme left of each fact type, assume that they can actually occur anywhere within fact types \( F_1, F_2 \ldots F_n \). In fact, \( F_1, F_2 \ldots F_n \) may not be distinct fact types. This allows the possibility that two roles participating in the disjunctive mandatory role constraint may occur within the same fact type. For example, if, in the diagram above, \( r_2 \) and \( r_4 \) are two different roles in the same fact type then \( F_2 \) and \( F_4 \) must be the same fact type.

\( f_1, f_2, f_3 \ldots f_n \) are power sets of type \( F_1, F_2, F_3 \ldots F_n \) respectively.

The disjunctive mandatory role constraint can be expressed as

\[
\forall x : o_c \quad x \in \{ f : f_1 \bullet f.r_1 \} \lor \\
\quad x \in \{ f : f_2 \bullet f.r_2 \} \lor \\
\quad \ldots \\
\quad x \in \{ f : f_n \bullet f.r_n \}
\]
Subset Constraint

In this section, we first consider the simplest case of two fact types where a subset constraint operates from a single role of one fact type to another role of the other fact type. The two roles must obviously belong to the same object type. Diagramatically, these constraints are indicated by a dashed arrow joining the roles participating in the constraint with the head of the arrow pointing to the superset and tail pointing to the subset. The figure below illustrates this situation for two fact types (of any arity).

Assume that \( r_{11} \) and \( r_{21} \) are of the same object type as indicated in the figure above. The variables \( f_1 \) and \( f_2 \) are sets of facts of power set type \( F_1 \) and \( F_2 \) respectively. The subset constraint is expressed as
Where the population of a pair of roles in one fact type must be a subset of the population of a pair of roles of another fact type as shown in figure 6.10 we can specify the subset constraint as follows:

\[ \{ f : f_2 \cdot (f.r_{21}, f.r_{22}) \} \subseteq \{ f : f_1 \cdot (f.r_{11}, f.r_{12}) \} \]

Note that the pairs of roles of either fact type need not necessarily be adjacent to each other although they drawn in this way in the diagram.

The general case is where there are \( n \) roles in one fact type which are subset of \( m \) roles of another fact type. Assume that \( r_{11} \) is of the same type as \( r_{21} \), \( r_{12} \) is of the same type as \( r_{22} \), ... and \( r_{1m} \) is of the same type as \( r_{2m} \) (i.e. \( r_{1x} \) is of the same type as \( r_{2x} \) where \( x \) is a value in the range \( 1 \ldots m \)).

\[ \{ f : f_2 \cdot (f.r_{21}, f.r_{22}, \ldots f.r_{2m}) \} \subseteq \{ f : f_1 \cdot (f.r_{11}, f.r_{12}, \ldots f.r_{1m}) \} \]

**Role Equivalence Constraint**

An equivalence constraint across corresponding sets of roles declares that the populations of those sets of roles must be the same. Corresponding here means that the number of roles must be the same as well as their types.

In this section, we first consider the case of two or more binary fact types where an equivalence constraint operates across single roles. Diagramatically, these constraints are
indicated by a dashed double headed arrow joining the roles participating in the constraint. The figure below illustrates this situation for two binary fact types and for an arbitrary number (n) of binary fact types.

\[
\begin{align*}
F_1 & \quad \text{F}_2 \\
O_e & \quad \text{F}_1 \\
F_2 & \quad \text{F}_n \\
\end{align*}
\]

Figure 21

As before, although in figure 21 the roles participating in the equivalence constraint are shown on the extreme left of each fact type, they can occur anywhere in the fact types provided the roles are of the same object type.

For the case of two binary fact types with role equivalence across \( r_{11} \) and \( r_{21} \):

\[
\{ f : f_1 \cdot f.r_{11} \} = \{ f : f_2 \cdot f.r_{21} \}
\]

Across several fact types, say \( m \) facts, the form can be generalised to

\[
\begin{align*}
\{ f : f_1 \cdot f.r_{11} \} & = \{ f : f_2 \cdot f.r_{21} \} \\
& = \{ f : f_m \cdot f.r_{n1} \}
\end{align*}
\]

Where the population of a pair of roles in one fact type must be identical to the population of a pair of roles of another fact type we have:

\[
\begin{align*}
\{ f : f_1 \cdot (f.r_{11}, f.r_{12}) \} & = \{ f : f_2 \cdot (f.r_{21}, f.r_{22}) \}
\end{align*}
\]

For the general case assume that there a population of \( n \) roles in one fact type which must be identical to the population of \( n \) roles of \( m \) other fact types. Assume that \( r_{11}, r_{21} \ldots r_{n1} \) are all of the same type; \( r_{12}, r_{22} \ldots r_{n2} \) are all of the same type ... and that \( r_{a1}, r_{a2} \ldots r_{am} \) are all of the same type.
The general form of the role equivalence constraint can be written as:

\[
\{ f : f_1 \bullet (f.r_{11}, f.r_{12}, \ldots, f.r_{1m}) \} \\
= \{ f : f_2 \bullet (f.r_{21}, f.r_{22}, \ldots, f.r_{2m}) \} \\
\ldots \\
= \{ f : f_n \bullet (f.r_{n1}, f.r_{n2}, \ldots, f.r_{nm}) \}
\]

**Exclusion Constraint**

An exclusion constraint across corresponding sets of roles declares that the populations of those sets of roles must be mutually exclusive. Corresponding here means that the number of roles must be the same as well as their types. The figures below graphically describe the exclusion constraint (circled X) across two single roles (left) and across a number (n) of single roles. Any object can only belong to one of these roles.

![Exclusion Constraint Diagram](image)

**Figure 22**

I will use the same conventions for variable names and descriptions as in the previous section.

For the case of a group of n binaries each with one role participating in the exclusion constraint, this may be expressed as:

\[
\emptyset = \cap\{ \{ f : f_1 \bullet f.r_{11} \}, \{ f : f_2 \bullet f.r_{21} \}, \ldots, \{ f : f_n \bullet f.r_{n1} \} \}
\]

In the case of pairs of roles with the common object types O₁ and O₂, assume that the pairs occur in the first two positions (although they may of course occur in any two positions in reality) in each of n participating fact types. The constraint can be expressed as

\[
\emptyset = \cap\{ \{ f : f_1 \bullet (f.r_{11}, f.r_{12}) \}, \{ f : f_2 \bullet (f.r_{21}, f.r_{22}) \}, \ldots \}
\]
In the case of m roles with the common object types O₁, O₂ ... Oₘ (some of which may be the same type), assume that the m roles occur in the first m positions (although they may of course occur in any m positions in reality) in each of n participating fact types. The constraint can be expressed as

$$\emptyset = \bigcap \{{f: f_1 \bullet (f.r_{11}, f.r_{12}, \ldots, f.r_{1m})},
\{f: f_2 \bullet (f.r_{21}, f.r_{22}, \ldots, f.r_{2m})\}, \ldots,
\{f: f_n \bullet (f.r_{n1}, f.r_{n2}, \ldots, f.r_{nm})\}\}$$

### Subtyping

There is no direct support for subtypes in Z, however, the concept of subtyping can be conveyed by creating a schema for each subtype. A subtype variable must be declared which is of the same basic type as its root supertype. I have adopted the approach that each object of that subtype population must satisfy a condition or set of conditions related to its participation in roles higher in the subtype hierarchy. These conditions determine which objects are allowed to participate in the roles related to a particular subtype.²⁰

An example illustrates the basic ideas.

#### Student / Lecturer example

![Diagram of Student / Lecturer example](image)

Figure 23

Firstly, we define the basic types for the schema:

```
[PERSON, STRING, SALARY, COURSE]
```

²⁰ here we have adopted the approach taken by Halpin [Halp95 pp.188-191] for instance.
Secondly we define a schema for each fact type and bridge type not used for the purpose of identifying an entity type.

<table>
<thead>
<tr>
<th>Person_known_by_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person : PERSON</td>
</tr>
<tr>
<td>Name : STRING</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person_has_status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person : PERSON</td>
</tr>
<tr>
<td>Status : STATUS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student_is_enrolled_in_course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student : PERSON</td>
</tr>
<tr>
<td>Course : COURSE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lecturer_earns_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecturer : PERSON</td>
</tr>
<tr>
<td>Salary : SALARY</td>
</tr>
</tbody>
</table>

Thirdly, we define a schema which contains populatable fact type variables and constraints on those variables.

<table>
<thead>
<tr>
<th>StudentLecturer_database</th>
</tr>
</thead>
<tbody>
<tr>
<td>person_has_status : P Person_has_status</td>
</tr>
<tr>
<td>person_known_by_name : P Person_known_by_name</td>
</tr>
<tr>
<td>student_is_enrolled_in_course : P Student_is_enrolled_in_course</td>
</tr>
<tr>
<td>lecturer_earns_salary : P Lecturer_earns_salary</td>
</tr>
<tr>
<td>person : P PERSON</td>
</tr>
<tr>
<td>student : P PERSON</td>
</tr>
<tr>
<td>lecturer : P PERSON</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{person} &= \{ f: \text{person\_has\_status} \bullet f.\text{Person} \\
&\quad \cup \{ f: \text{person\_known\_by\_name} \bullet f.\text{Person} \} \\
\text{student} &= \{ f: \text{student\_is\_enrolled\_in\_course} \bullet f.\text{Student} \} \\
\text{student} &\subseteq \{ f: \text{person\_has\_status} \mid f.\text{Status} = S \bullet f.\text{Person} \} \\
\text{lecturer} &= \{ f: \text{lecturer\_earns\_salary} \bullet f.\text{Lecturer} \} \\
\text{lecturer} &\subseteq \{ f: \text{person\_has\_status} \mid f.\text{Status} = L \bullet f.\text{Person} \}
\end{align*}
\]

According to the NIAM approach, the subset of students must have a definition by which they can be identified from facts related to the object type Person. In this case it is that they must have a status of ‘S’ from the fact that relates persons to status i.e. `person_has_status`. 

\[
\text{STATUS} ::= S \mid L
\]

Note that status can only take on the values S or L
In this example, there is only one fact type associated with Student, that is, Student is enrolled in Course, so the population of students is simply those objects participating in that single role.

While the set student must be a subset of the set of person, that is, student ⊆ person, this constraint is weaker than the one requiring that objects in student must have a status of ‘S’ and so this predicate would be redundant if included.
Chapter 7

Conversion Procedure Steps And Examples

Introduction

In prior chapters, all the components required for my translation procedure were provided. In this chapter I put them altogether into a procedure in which I describe how to prepare a schema for translation, describe the steps that must be performed and where necessary in each step any special points that need to be considered.

Conversion Procedure Steps

The general conversion procedure consists of the following steps:

Step 0 Standardize the fact model schema

This requires that the schema be brought back to a “standard” form.

Nested Fact Types

First, nested fact types are transformed back to the flattened form. Examine, for example the nested diagram below

![Nested Diagram](image)

We can transform the nested form back to two fact types,

Student … for the Unit … obtained Grade …
Student … completed Unit … on Date …
This may be visualised now as

\[\text{Figure 25}\]

**Lazy Entity Types**

In the case of “lazy” entity types, an idea proposed by Halpin (2001) but not widely used by others as far as I know, they should handled in the following way. If they appear as part of a nested fact type, create a fact type for that lazy entity type. For example in the diagram above, enrolment is a lazy entity type. Therefore a fact type relating A and B is also necessary.

Student … is enrolled in Unit …

\[\text{Figure 26}\]

Where an atomic entity type is declared to be lazy, an extra unary fact type should be created to hold the “surplus” entities not used in facts of other fact types.

For example, if there is a pool of available units, not all of which were necessarily being used currently but we wish to keep track of all these available units, this might be a reason to create a lazy entity type Unit. However, we could also represent this as a unary fact that a “unit … is available”. The diagram below indicates the transformation.

\[\text{Figure 27}\]
Identification schemes

Any bridge types or fact types used for identification should be “hidden”. These will be used for the purposes of populating the variables but take no further part in the conversion steps. For example, if we consider the diagram below, a Track can uniquely be identified by the combination of TrackNo and the identifier of CD, that is, CDtitle. When we refer to a track therefore we should use the combination <CDtitle, TrackNo>. The corresponding bridge and fact type are then “hidden” and ignored.

![Diagram](image)

**Figure 28**

The rules are as follows:

1. Where there is a mandatory one to one mapping across a bridge\(^{21}\) type (an injection) which is to be used to identify the entity type then ignore this bridge type\(^{22}\).
2. Where a group of label types attached to an entity type is mandatory and provides a one to one mapping between the entity type and set of label types and this is to be used as the identification scheme, then ignore these label types.
3. If a fact type needs to be used as part of the identification scheme for an object type then ignore this fact type.

This now standardises the model so it can be used for the conversion procedure.

The steps in the conversion procedure are described below.

\(^{21}\) A bridge type we define as a fact type which connects an entity type and a label type. The term bridge is used to highlight the idea that it connects conceptual objects (things of the mind) with labels (say strings or numbers) used to refer to them.

\(^{22}\) In the case of more than one bridge type being able to identify an entity type, then only one is chosen as THE identification scheme.
Step 1 Declare Basic Types

After step 0, all remaining entity types in the schema become basic types in the specification. Any other label types can be declared as strings if they refer to labels such as a name or title and so on or numeric if they relate to measurable entities such as length, mass, temperature or other dimensions. This simple subdivision into strings and numbers is very simple of course and there may be times when it is useful to declare other or further basic types.

Step 2 Declare Schema Types

Each remaining fact type in the diagram is declared as a schema type (except for unary fact types which appear in step 3). The components of the schema type take their names from the entity or label types which comprise the fact type except in the case where an object type is used two or more times in the same fact type in which case different names will be necessary to distinguish between the different roles played by that object type.

It is suggested that the Z schema type names created should be the same as in the fact model schema with the entity type names included. This will make every Z schema type name unique and aid in checking that the fact model schema is consistent with the Z schema.

In some cases the Z schema types with different names may have the same signature i.e. each has declared exactly the same variable names with the same type. This may occur when fact types of the same arity are connected to the same object types in the same numbers. In this case the schema types are equivalent and therefore redundant but I don’t see that this causes any problems.

Other role constraints that apply within a fact should be declared here as a predicate within the schema type eg. a fact type such as

“Employee … increases salary from Amount … to new Amount …”

may require that old_Salary < new_Salary and this condition can be included in the predicate of this fact type’s Z schema type definition. Constraints on a fact type that require comparison of one fact with another of the same type such as with internal uniqueness constraints could be applied after step 4.

Step 3 Declare fact type variables

Each fact type from the fact model schema will be declared as a variable of the corresponding schema type. All the fact types may be declared within the one schema or we may choose to subdivide the fact types into smaller logically connected components.
At this point, unary facts should be declared and these will be the same type as the entity type to which they are attached. Names for unary fact types should also include the entity type name to which they are attached eg. for the unary fact below,

![Diagram](unit (unitcode) is available)

The suggested name for the corresponding variable is unit_is_available

**Step 4 Define internal uniqueness constraints**

Assume that each fact type has n roles where n is the number of roles. If the strongest uniqueness constraint spans all n roles then it is not necessary to specify the uniqueness constraint since duplicate values in sets do not exist. Remaining fact types must have at least one uniqueness constraint spanning n – 1 roles. One predicate for each of these uniqueness constraints is required.

**Step 5 Define mandatory role constraints**

For each entity type with an explicit mandatory role constraint attached, define a variable based on that entity type and establish the total population of all the role populations attached to that type.

Define a mandatory role constraint for each role that is mandatory.

It is not necessary to define a mandatory role constraint for an object type that participates in only one fact type since the mandatory role constraint is implied in such circumstances.

**Step 6 Define external uniqueness constraints**

**Step 7 Define subset constraints**

**Step 8 Define equality constraints**

**Step 9 Define exclusion constraints**

The only standard constraint which has not been included and which may seem to be the simplest is the Value constraint. I do, in fact, consider this constraint and also the introduction of the reference scheme for each entity type at the end of this chapter.
The order of steps 4 to 9 is not critical and can simply be regarded as a checklist to ensure that all the common constraint patterns have been considered. The constraints explicitly mentioned in steps 4 to 9 I regard as standard ones since they have been in existence for quite some time and appear across the literature. Authors have created other constraint patterns which can be represented graphically but I have not considered to be recognised as standard constraints and I regard them as beyond the scope of this thesis. Nevertheless I did include one such non-standard constraint in the second example which follows in this chapter and it specific to that example (i.e. I chose not to provide a generalised pattern).

**Example 1**

As a first illustration of the use of a fact model and the conversion procedure into Z, I have chosen a simple example from the text “Object Orientation in Z” by Stepney, Barden and Cooper (1992) which deals with a video library. This example has the advantage that it was used to illustrate the use of Entity Relationship diagrams as an intermediate step towards producing a requirements statement in Z. It will allow for some pertinent comments to be made. Operations are also included in the example but discussion of this is not relevant until considered later in Chapter 8.

A short summary of the situation is as follows:

*The video library is a club which loans out videos to users who may be club members or staff. For any particular video, there may be many copies. Sometimes a copy may not be in stock in which case it is not available for hire. For charging purposes, videos are grouped into bands and each band has charge associated with it. Videos have a title, a subject category and a censorship classification which restricts who may borrow a video.*

From the Entity relationship diagram provided by Stepney, Barden and Cooper it is possible to derive an ORM schema in Figure 29. However, because ORM allows for a more detailed and precise data model, assumptions were made in the translation.
It is possible to make a few preliminary comments regarding the original entity relationship diagram provided from a fact modelling viewpoint.

First, some of the entity types are not referable i.e. no scheme has been provided by which the users can uniquely refer to an entity of that type. Video is not referable unless we assume that title for instance, is going to be unique. The identification scheme for a copy has been deferred. We may choose to allocate each copy a unique number or to combine a title with a copy number within title. As yet we do not know how to identify a club member or a staff member.

Secondly, it is not clear the mandatory role or uniqueness constraints to apply in some cases. Is it possible that some fields described in the ER diagram could be null? Could a video have no subject category for instance or could a user have no date of birth recorded? Because the fact model requires us to make these explicit, I have made what seem to be reasonable assumptions in each case. These are:

1. Users are identified by some suitable means using a 1:1 mandatory fact type not specified in the schema, for example, each user is allocated a unique identification number.
2. Copies of videos are similarly given a unique identification number (unspecified in the schema).
3. It is assumed that each band has a unique code (unspecified in the schema).
4. Where null values in the original ER description were possible, the assumptions made are as specified in the schema.

5. It will be assumed that titles are unique because there is insufficient information to uniquely identify two different videos that have the same title, such as a remake of a film. If date of production or release of a film were also included as a fact the combination of film title and this date could act as a unique combination. It will be assumed that the constraint between video and title is therefore 1:1 and mandatory on the video side.

We now go through the translation procedure.

**Step 0 Standardize the fact model schema**

The fact model schema in Figure 29 contains no nested fact types or lazy entity types. Bridge types or fact types which used for the purpose of identifying an entity type are effectively “hidden” in the schema. This leaves only entity types and some label types which are not used directly for the purpose of identifying an entity (although they may be used as a form of “secondary key” such as when a customer forgets their unique customer number and data is accessed through their name).

**Step 1 Declare Basic Types**

Each entity type is converted into a basic type. These are

[COPY, VIDEO, BAND, MONEY, USER, ADDRESS, DATE, SUBJECT, STRING]

We can explicitly state the values associated with CERTIFICATION as

CERTIFICATION ::= exempt | uCert | pg | twelve | fifteen | eighteen

We can explicitly state the values associated with MEMBER_TYPE as

MEMBER_TYPE ::= staff | club | member

The type [STRING] will be used for label type Title which will appear in the Z schema types which follow in step 2.

**Step 2 Declare Schema Types**

For each remaining fact type and bridge type, a schema type is created. Note that unary facts are not included here but are found in the schema that describes the entire database.
Step 3 Declare fact type variables

Variables for each fact type and bridge type are created as well as a variable for each entity type which participates in more than one fact type. The total population related to an entity
type variable is the sum of the populations of each role associated with that entity type. I do not include label types. Mandatory role constraints are specified.

Note that the unary fact Copy_is_in_stock is also included as a variable of type COPY.

```plaintext
Video_library_database
user_known_by_name : P User_known_by_name
user_resides_at_address : P User_resides_at_address
user_born_on_date : P User_born_on_date
user_classed_as_member_type : P User_classed_as_member_type
copy_is_hired_to_user : P Copy_is_hired_to_user
copy_of_video : P Copy_of_video
video_is_charged_according_to_band : P Video_is_charged_according_to_band
band_charges_amount_of_money : P Band_charges_amount_of_money
video_classified_as_certification : P Video_classified_as_certification
video_is_about_subject : P Video_is_about_subject
copy_is_in_stock : P COPY
```

**Step 4 Define internal uniqueness constraints**

Each uniqueness constraint on the ORM schema (which remains after reference schemes are hidden and ignored) is translated separately to a predicate in the Z schema. For example the uniqueness constraint across the fact type

user … is known by name …

is

\[ \forall f1, f2: \text{user_known_by_name} \cdot f1 \neq f2 \Rightarrow f1.\text{user} \neq f2.\text{user} \]

**Step 5 Define mandatory role constraints**

This consists of two sub-steps. First define establish the total membership of each entity type by combining all the memberships of each role in which user participates. For example for the entity type user we have

```plaintext
user
  = \{ x: \text{user_known_by_name} \cdot x.\text{user} \}
  \cup \{ x: \text{user_resides_at_address} \cdot x.\text{user} \}
  \cup \{ x: \text{user_born_on_date} \cdot x.\text{user} \}
  \cup \{ x: \text{user_classed_as_member_type} \cdot x.\text{user} \}
  \cup \{ x: \text{copy_is_hired_to_user} \cdot x.\text{user} \}
```

Then for each role of user where the role is mandatory we write a predicate establishing the mandatory role constraint. For the entity type user there are two mandatory role constraints

\[ \forall x: \text{user} \cdot \exists x \in \{ f: \text{user_classed_as_member_type} \cdot f.\text{user} \} \]
\[ \forall x: \text{user} \cdot \exists x \in \{ f: \text{user_known_by_name} \cdot f.\text{user} \} \]
The complete schema for the video library database that results from procedure described in this thesis is then as follows.
Video_library_database

user_known_by_name : $ User_known_by_name
user_resides_at_address : $ User_resides_at_address
user_born_on_date : $ User_born_on_date
user_classed_as_member_type : $ User_classed_as_member_type
copy_is_hired_to_user : $ Copy_is_hired_to_user
copy_of_video : $ Copy_of_video
video_is_charged_according_to_band : $ Video_is_charged_according_to_band
band_charges_amount_of_money : $ Band_charges_amount_of_money
video_classified_as_certification : $ Video_classified_as_certification
video_is_about_subject : $ Video_is_about_subject
copy_is_in_stock : $ COPY
user : $ USER
copy : $ COPY
video : $ VIDEO
band : $ BAND

∀ f1, f2: user_known_by_name • f1 ≠ f2 ⇒ f1.user ≠ f2.user
∀ f1, f2: user_resides_at_address • f1 ≠ f2 ⇒ f1.user ≠ f2.user
∀ f1, f2: user_born_on_date • f1 ≠ f2 ⇒ f1.user ≠ f2.user
∀ f1, f2: user_classed_as_member_type • f1 ≠ f2 ⇒ f1.user ≠ f2.user
∀ f1, f2: copy_is_hired_to_user • f1 ≠ f2 ⇒ f1.user ≠ f2.user
∀ f1, f2: copy_of_video • f1 ≠ f2 ⇒ f1.copy ≠ f2.copy
∀ f1, f2: video_is_charged_according_to_band
  • f1 ≠ f2 ⇒ f1.video ≠ f2.video
∀ f1, f2: band_charges_amount_of_money • f1 ≠ f2 ⇒ f1.band ≠ f2.band
∀ f1, f2: video_classified_as_certification
  • f1 ≠ f2 ⇒ f1.video ≠ f2.video
∀ f1, f2: video_is_about_subject • f1 ≠ f2 ⇒ f1.video ≠ f2.video

user
  = { x: user_known_by_name • x.user }
  ∪ { x: user_resides_at_address • x.user }
  ∪ { x: user_born_on_date • x.user }
  ∪ { x: user_classed_as_member_type • x.user }
  ∪ { x: copy_is_hired_to_user • x.user }
  ∪ { x: copy_is_in_stock • x.copy }

video
  = { x: video_is_charged_according_to_band • x.video }
  ∪ { x: video_classified_as_certification • x.video }
  ∪ { x: video_is_about_subject • x.video }
  ∪ { x: copy_is_hired_to_user • x.copy }
  ∪ { x: copy_is_in_stock • x.copy }

band
  = { x: video_is_charged_according_to_band • x.band }
  ∪ { x: band_charges_amount_of_money • x.band }
∀ x: user • x ∈ { f: user_classed_as_member_type • f.user }
∀ x: user • x ∈ { f: user_known_by_name • f.user }
∀ x: copy • x ∈ { f: copy_of_video • f.copy }
∀ x: video • x ∈ { f: video_classified_as_certification • f.video }
∀ x: video • x ∈ { f: video_is_about_subject • f.video }
∀ x: video • x ∈ { f: video_is_charged_according_to_band • f.video }
∀ x: band • x ∈ { f: band_charges_amount_of_money • f.band }
**Example 2: A More Complex Example**

This example (shown below) is taken from Halpin (1995, p.373) to illustrate more comprehensively the various steps of the conversion procedure. It demonstrates most aspects involved in a translation to Z.

![Diagram](image)

**Figure 30**

**Step 0 Standardize the fact model schema**

There are no nested fact types here and no lazy entity types.

Each entity type has an identification scheme enclosed so no other fact types are required to identify entity types and so no fact types need to be hidden for the purpose of conversion.
Step 1 Determine Basic Types

These are

\[
\text{[ DATE, PROJECT, SITE, MONEYAMT, OPERATING\_SYSTEM, PROGRAMMER, LANGUAGE, STRING]} \]

\[
\begin{align*}
\text{STAGE} & : = \text{'1'} | \text{'2'} \\
\text{SKILL} & : = \text{'3GL'} | \text{'4GL'} \\
\text{CLASS} & : = \text{'Junior'} | \text{'Senior'}
\end{align*}
\]

I have included the basic type STRING to cover the inclusion of PROGRAMMER NAME as part of the signature of a schema type.

Step 2 Declare Schema Types

<table>
<thead>
<tr>
<th>Stage of Project at Site completed on Date</th>
<th>Stage: STAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project: PROJECT</td>
<td>Project: PROJECT</td>
</tr>
<tr>
<td>Site: SITE</td>
<td>Site: SITE</td>
</tr>
<tr>
<td>Date: DATE</td>
<td>Date: DATE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project at Site has budget of MoneyAmt</th>
<th>Stage: STAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project: PROJECT</td>
<td>Project: PROJECT</td>
</tr>
<tr>
<td>MoneyAmt: MONEYAMT</td>
<td>MoneyAmt: MONEYAMT</td>
</tr>
</tbody>
</table>

| Project at Site uses OperatingSystem     | Project: PROJECT |
|------------------------------------------| Site: SITE |
| OperatingSystem: OPERATING\_SYSTEM      | OperatingSystem: OPERATING\_SYSTEM |

| Project requires Skill                   | Project: PROJECT |
|------------------------------------------| Skill: SKILL |

| Project involves Programmer             | Project: PROJECT |
|------------------------------------------| Programmer: PROGRAMMER |

| Programmer has ProgrammerName           | Programmer: PROGRAMMER |
|------------------------------------------| ProgrammerName: STRING |

| Programmer is novice at Language        | Programmer: PROGRAMMER |
|------------------------------------------| Language: LANGUAGE |
Step 3 Declare fact type variables

Project Programmer Schema

\[
\begin{align*}
\text{stage_of_Project_at_Site_completed_on_Date} : & \quad \mathcal{P} \text{Stage_of_Project_at_Site_completed_on_Date} \\
\text{project_at_Site_has_budget_of_MoneyAmt} : & \quad \mathcal{P} \text{Project_at_Site_has_budget_of_MoneyAmt} \\
\text{project_at_Site_uses_OperatingSystem} : & \quad \mathcal{P} \text{Project_at_Site_uses_OperatingSystem} \\
\text{project_requires_Skill} : & \quad \mathcal{P} \text{Project_requires_Skill} \\
\text{project_involves_Programmer} : & \quad \mathcal{P} \text{Project_involves_Programmer} \\
\text{programmer_has_ProgrammerName} : & \quad \mathcal{P} \text{Programmer_has_ProgrammerName} \\
\text{programmer_is_novice_at_Language} : & \quad \mathcal{P} \text{Programmer_is_novice_at_Language} \\
\text{programmer_is_expert_at_Language} : & \quad \mathcal{P} \text{Programmer_is_expert_at_Language} \\
\text{programmer_has_Class} : & \quad \mathcal{P} \text{Programmer_has_Class}
\end{align*}
\]

Step 4 Define internal uniqueness constraints

In the predicate part of the Z schema above we will add the following constraints, using the shorthand suggested earlier:

\[
\begin{align*}
\mathcal{U}_1 (\text{stage_of_Project_at_Site_completed_on_Date}, \text{Date}) \\
\mathcal{U}_1 (\text{project_at_Site_has_budget_of_MoneyAmt}, \text{MoneyAmt}) \\
\mathcal{U}_1 (\text{programmer_has_ProgrammerName}, \text{ProgrammerName})
\end{align*}
\]
Step 5 Define mandatory role constraints

We need to declare the following variables

project: PROJECT
site: SITE
programmer: PROGRAMMER

The population of project is defined as

\[
\text{project} = \{ x : \text{stage_of_Project_at_Site_completed_on_Datetime} \cdot x.\text{Project} \} \cup \\
\{ x : \text{project_at_Site_has_budget_of_MoneyAmt} \cdot x.\text{Project} \} \cup \\
\{ x : \text{project_requires_Skill} \cdot x.\text{Project} \} \cup \\
\{ x : \text{project_at_Site_uses_OperatingSystem} \cdot x.\text{Project} \} \cup \\
\{ x : \text{project_involves_Programmer} \cdot x.\text{Project} \}
\]

The population of site is defined as

\[
\text{site} = \{ x : \text{stage_of_Project_at_Site_completed_on_Datetime} \cdot x.\text{Site} \} \cup \\
\{ x : \text{project_at_Site_has_budget_of_MoneyAmt} \cdot x.\text{Site} \} \cup \\
\{ x : \text{project_at_Site_uses_OperatingSystem} \cdot x.\text{Site} \}
\]

The population of programmer is defined as

\[
\text{programmer} = \{ x : \text{project_involves_Programmer} \cdot x.\text{Programmer} \} \cup \\
\{ x : \text{programmer_has_ProgrammerName} \cdot x.\text{Programmer} \} \cup \\
\{ x : \text{programmer_has_Class} \cdot x.\text{Programmer} \} \cup \\
\{ x : \text{programmer_is_novice_at_Language} \cdot x.\text{Programmer} \} \cup \\
\{ x : \text{programmer_is_expert_at_Language} \cdot x.\text{Programmer} \}
\]

Mandatory role constraints for project will be defined as follows:

\[
\forall x : \text{project} \cdot \\
\exists f : \{ f : \text{project_at_Site_has_budget_of_MoneyAmt} \cdot f.\text{Site} \}
\]

\[
\forall x : \text{project} \cdot \\
\exists f : \{ f : \text{project_at_Site_uses_OperatingSystem} \cdot f.\text{Site} \}
\]
∀x : programmer • x ∈ {f : programmer_has_ProgrammerName • f.Programmer}

∀x : programmer • x ∈ {f : programmer_is_expert_at_Language • f.Programmer}

∀x : programmer • x ∈ {f : programmer_has_Class • f.Programmer}

**Step 6 Define external uniqueness constraints**

For this step we can use the earlier proposed shorthand notation \( U_{EB} \)

\( U_{EB} \) (programmer_has_Class.Class, project_involves_Programmer.Project)

**Step 7 Define subset constraints**

\( \{f: project_involves_Programmer • f.Project\} \subseteq \{f: project_requires_Skill • f.Project\} \)

\( \{f: stage_of_Project_at_Site_completed_on_Date • (f.Project, f.Site)\} \subseteq \{f: project_at_Site_has_budget_of_MoneyAmt • (f.Project, f.Site)\} \)

**Step 8 Define equality constraints**

\( \{f: project_at_Site_has_budget_of_MoneyAmt • (f.Project, f.Site)\} = \{f : project_at_Site_uses_OperatingSystem • (f.Project, f.Site)\} \)

**Step 9 Define exclusion constraints**

\( \emptyset = \cap \{\)

\( \{f: programmer_is_expert_at_Language • (f.Programmer, f.Language)\}, \)

\( \{f: programmer_is_novice_at_Language • (f.Programmer, f.Language)\}\}

**Other constraints**

Here we translate any other constraints that may be mentioned. In this case we have only occurrence frequency constraints\(^{23}\) mentioned in the fact model schema. These define how many times an object or set of objects may occur within a fact type.

\(^{23}\) Occurrence frequency constraints are described by Halpin (2001) but not mentioned elsewhere to my knowledge so I did not include them as a standard constraint in my procedure.
For the fact type “project_at_Site_uses_OperatingSystem”, we need only set a maximum of 2 for the components Project and Site. The predicate below inside the round brackets states that there exist three facts with that have the same values for project and site. By negating that predicate we ensure that there will not be three (and hence four, five etc) facts with the same project and site combination.

\[\neg(\exists f_1, f_2, f_3: \text{project_at_Site_uses_OperatingSystem} \quad \bullet f_1 \neq f_2 \land f_2 \neq f_3 \land f_1 \neq f_3 \land
\]
\[((f_1.\text{project}, f_1.\text{site}) = (f_2.\text{project}, f_2.\text{site}) \land (f_2.\text{project}, f_2.\text{site}) = (f_3.\text{project}, f_3.\text{site})))\]

Setting a requirement of exactly 2 for the project component of the fact type “project_involves_Programmer” involves setting an upper limit and a lower limit of 2 occurrences of project. This involves a predicate like the one above to limit the number of occurrences of project to less than three and another to ensure that it is at least two.

\[\neg(\exists f_1, f_2, f_3: \text{project_involves_Programmer} \bullet
f_1 \neq f_2 \land f_2 \neq f_3 \land f_1 \neq f_3 \land
\]
\[(f_1.\text{project} = f_2.\text{project} \land f_2.\text{project} = f_3.\text{project})) \land
(\exists f_1, f_2,: \text{project_involves_Programmer} \bullet
f_1 \neq f_2 \land (f_1.\text{project} = f_2.\text{project}))\]
The Final Schema

These following pages contain the final Z schema for this more complex example. I have used the two suggested shorthand notations for internal uniqueness and external binary constraints here\(^\text{24}\).

<table>
<thead>
<tr>
<th>Project_Programmer_Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>stage_of_Project_at_Site_completed_on_Date:</td>
</tr>
<tr>
<td>( \mathbb{P} \text{ Stage_of_Project_at_Site_completed_on_Date} )</td>
</tr>
<tr>
<td>project_at_Site_has_budget_of_MoneyAmt:</td>
</tr>
<tr>
<td>( \mathbb{P} \text{ Project_at_Site_has_budget_of_MoneyAmt} )</td>
</tr>
<tr>
<td>project_at_Site_uses_OperatingSystem:</td>
</tr>
<tr>
<td>( \mathbb{P} \text{ Project_at_Site_uses_OperatingSystem} )</td>
</tr>
<tr>
<td>project_requires_Skill : ( \mathbb{P} \text{ ProjectRequires_Skill} )</td>
</tr>
<tr>
<td>project_involves_Programmer :</td>
</tr>
<tr>
<td>( \mathbb{P} \text{ Project_involves_Programmer} )</td>
</tr>
<tr>
<td>programmer_has_ProgrammerName :</td>
</tr>
<tr>
<td>( \mathbb{P} \text{ Programmer_has_ProgrammerName} )</td>
</tr>
<tr>
<td>programmer_is_novice_at_Language :</td>
</tr>
<tr>
<td>( \mathbb{P} \text{ Programmer_is_novice_at_Language} )</td>
</tr>
<tr>
<td>programmer_is_expert_at_Language :</td>
</tr>
<tr>
<td>( \mathbb{P} \text{ Programmer_is_expert_at_Language} )</td>
</tr>
<tr>
<td>programmer_has_Class : ( \mathbb{P} \text{ Programmer_has_Class} )</td>
</tr>
<tr>
<td>project : PROJECT</td>
</tr>
<tr>
<td>site : SITE</td>
</tr>
<tr>
<td>programmer : PROGRAMMER</td>
</tr>
</tbody>
</table>

* uniqueness constraints *

\( \text{U}\_1 \) (stage_of_Project_at_Site_completed_on_Date, Date)
\( \text{U}\_1 \) (project_at_Site_has_budget_of_MoneyAmt, MoneyAmt)
\( \text{U}\_1 \) (programmer_has_ProgrammerName, ProgrammerName)
\( \text{U}\_1 \) (programmer_has_Class, Class)

\(^{24}\) In order to make the Z schema easier to follow for the reader comments have been added later into the schema after type checking. The predicates using the abbreviations \( \text{U}\_1 \) and \( \text{U}\_\text{EB} \) were also inserted later as well.
* mandatory role constraints

project =
{x: stage_of_Project_at_Site_completed_on_Date • x.Project }
∪ {x: project_at_Site_has_budget_of_MoneyAmt • x.Project }
∪ {x: project_requires_Skill • x.Project }
∪ {x: project_at_SiteUses_OperatingSystem • x.Project }
∪ {x: project_involves_Programmer • x.Project }

site =
{x: stage_of_Project_at_Site_completed_on_Date • x.Site }
∪ {x: project_at_Site_has_budget_of_MoneyAmt • x.Site }
∪ {x: project_at_Site_uses_OperatingSystem • x.Site }

programmer =
{x: project_involves_Programmer • x.Programmer }
∪ {x: programmer_has_ProgrammerName • x.Programmer }
∪ {x: programmer_has_Class • x.Programmer }
∪ {x: programmer_is_novice_at_Language • x. Programmer }
∪ {x: programmer_is_expert_at_Language • x. Programmer }

∀ x:project • x ∈
{f: Project_at_Site_has_budget_of_MoneyAmt • f.Project}
∀ x: project • x ∈
{f: project_at_Site_uses_OperatingSystem • f.Project}
∀ x: site • x ∈
{f: project_at_Site_has_budget_of_MoneyAmt • f.Site}
∀ x: site • x ∈
{f: project_at_Site_uses_OperatingSystem • f.Site}
∀ x: programmer • x ∈
{f: programmer_has_ProgrammerName • f.Programmer}
∀ x: programmer • x ∈
{f: programmer_is_expert_at_Language • f.Programmer}
∀ x: programmer • x ∈
{f: programmer_has_Class • f.Programmer}

* External binary constraint *

∪_{\Sigma} (programmer_has_Class.Class, 
project_involves_Programmer.Project)

* subset constraints *

{f: project_involves_Programmer • f.Project} ⊆ {f: project_requires_Skill • f.Project}
{f: stage_of_Project_at_Site_completed_on_Date • (f.Project, f.Site)} ⊆ {f: project_at_Site_has_budget_of_MoneyAmt • (f.Project, f.Site)}

* equality constraints *

{f: project_at_Site_has_budget_of_MoneyAmt • (f.Project, f.Site)} = {f: project_at_Site_uses_OperatingSystem • (f.Project, f.Site)}
* exclusion constraints *
\[
\emptyset = \bigcap \{(f : \text{programmer\_is\_expert\_at\_Language} \land (f.\text{Programmer}, f.\text{Language})), \{f : \text{programmer\_is\_novice\_at\_Language} \land (f.\text{Programmer}, f.\text{Language})\}\}
\]

* occurrence frequency constraints *
\[
\lnot (\exists f_1, f_2, f_3 : \text{project\_at\_Site\_uses\_OperatingSystem} \\
\quad \land f_1 \neq f_2 \land f_2 \neq f_3 \land f_1 \neq f_3 \\
\quad \land ((f_1.\text{project}, f_1.\text{site}) = (f_2.\text{project}, f_2.\text{site}) \land \\
\quad (f_2.\text{project}, f_2.\text{site}) = (f_3.\text{project}, f_3.\text{site}))}
\]
\[
\lnot (\exists f_1, f_2, f_3 : \text{project\_involves\_Programmer} \\
\quad \land f_1 \neq f_2 \land f_2 \neq f_3 \land f_1 \neq f_3 \\
\quad \land (f_1.\text{project} = f_2.\text{project} \land f_2.\text{project} = f_3.\text{project}))
\]
\[
\land (\exists f_1, f_2 : \text{project\_involves\_Programmer} \\
\quad \land f_1 \neq f_2 \land (f_1.\text{project} = f_2.\text{project}))
\]
Chapter 8

Dynamics and Operation Schemas

Introduction

I have provided a translation procedure for schemas based on the fact model into a specification in Z that provides a description of the static aspects of a system. In doing so I have used a style of representation in Z that I believe best preserves the concepts involved in the fact model. I believe that doing so makes the validation and verification process easier. However, the result of the procedure is what appears to be a large state schema containing a long list of variables defined which are rather verbose by most comparisons with typical Z schemas.

In this chapter I develop a schema to represent the dynamic aspects of a system. This is to examine the effect that my representation of a fact model schema in Z has on the specification of the system dynamics. The intention is merely to consider the more obvious and immediate consequences.

How are the dynamic aspects of a system expressed?

The dynamic aspects of a system are described by ISO (1982 p39) in the following way: “Dynamic rules are concerned with permissible transitions from a collection of sentences to a next one and thus specify the possible sequences of information base states. They establish dependencies between several parts of the system through several instants in time. Dynamic rules discuss the laws of change.” Given some initial starting point the dynamic constraints restrict the potential populations possible which might have been allowed otherwise (Proper, 1994, p10-11)

Although they are called static rules, static rules can be considered to be a subset of dynamic rules in that they are rules for which the prior state of the information base is irrelevant. Another way of stating this is that all information base states are possible as prior information base states (ISO 1982, p35). The validity of the information base is decided solely by examination of the collection of facts at any particular instant in time and the static rules applying at that time.
When considering the possible dynamic changes that occur from state to state it is assumed that the current (most recent) state of the information system, the operation and any required input data contain all the information we needed to make a decision about the “allowedness” of next state i.e. the history of the information system prior to the current state does not need to be known. This means that the information base needs to be designed at the logical level with this in mind (ISO 1982, p36).

Sometimes a static constraint may impose a rule which can also be interpreted as a dynamic one. For example a subset constraint may be used to impose a particular order to events. As a concrete example, consider the following fact types

“Student … is enrolled in subject …” and

“Student … for the subject … obtained the score …”

A reasonable requirement would be that you must be enrolled in a subject in order to receive a score. This can be enforced using static constraints by having the two fact types in the information base and having a subset constraint such that the populations of student and subject in the second fact type are a subset of those populations in the first. This same constraint can also be enforced dynamically within an operation by checking that students are enrolled in a subject before allowing the addition of a score. The difference in the two approaches is that in the static case the constraint is true for every state of the database no matter what operations take place. When implemented dynamically however, the constraint is only enforced for data added via that operation. In the latter case, the initial state of the database could contain data that does not conform to the constraint and it is possible that other operations (either deliberately or by mistake) may not enforce that constraint.

Two different but mutually convertible kinds of description are possible in the specification of dynamic rules and constraints: state oriented and action oriented. Both see the information system as going through a sequence of states given by the momentary states of the conceptual schema and the information base. According to (ISO 1982, p36):

“In state oriented descriptions rules describing possible changes to the information base make reference to sentences before and after the change.”

Z provides such a state oriented specification of the information system. This is evidenced by the use of before and after states of sets which are identified by the use of a single quote as decoration for the new state of a variable and its omission for the old state (eg. employee’ and employee respectively).

Spivey (1992) describes the static and dynamic aspects of a system in the following way.
The static aspects include:
- the states it can occupy
- the invariant relationships that are maintained as the system moves from state to state

The dynamic aspects include:
- the operations that are possible
- the relationships between the inputs and outputs
- the changes of state that happen

In Z, a schema is used to declare variables and predicates which constrain the possible values of those variables. In applying Z to the specification of information systems we interpret the variables and predicates to represent aspects of interest in the information system. A common approach is then to arbitrarily write some schema called state schema to describe the static aspects of an information system and others called operation schema to describe the dynamic aspects.

An operation schema commonly is written so that it represents an operation that from the external perspective of the application users must be completed in total or not at all. This is sometimes described as an atomic operation. Other schema might be created to handle situations of failure for that operation (e.g. when preconditions are not met). Using schema in this fashion breaks up the specification of the dynamic aspects of an application into a set of manageable components. Within operation schema, one must appropriately interpret the variables:
- Variables decorated with and without a \( ' \) represent the value of the variable before and after the operation respectively (e.g. these represent datastores in the database).
- Variables decorated with a \( ? \) are normally associated with the before state of the information system and interpreted as providing an input value (e.g. a input from the external environment such as user keying in a value).
- Variables decorated with a \( ! \) are normally associated with the after state of the information system and interpreted as providing an output value (e.g. an output to the external environment such as message to a user).

The appropriate interpretation of schema and variables therefore simulates the effects of events on the information system.
**Consequences of the NIAM to Z Mapping Procedure**

Our mapping procedure groups together all static constraints into the one schema. I note that this resultant schema may be regarded as large and unwieldy.

Our single schema could be subdivided into smaller schema. However, subdivision of a database needs to be done judiciously. The fact model schema shows that facts are typically closely interlinked through common entity types and one needs appropriate criteria in order perform the subdivision. Further, significant relationships may not naturally fit into any of the subdivisions. For example, a database may be subdivided according to functional lines such as accounting, marketing, sales and so on. Using these subdivisions a relationship describing authority structures existing among managers of different functional areas does not fit naturally into any particular subdivision. Furthermore, constraints may span across different relationships and in some cases across subdivisions. For example, an exclusion constraint may span several fact types and subdividing a schema may separate the fact types and inadvertently lead to omitting constraints that span across the relationships.

One advantage of using a fact model schema is that a detailed graphical representation of the database allows us to see the database as a whole rather than in a piecemeal fashion and hence reduces the chance of errors such as I have described.

Expressing constraints within a state schema rather than in operation schema avoids duplication of these constraints across several different operation schema. Moreover, because a constraint can be often be expressed in a variety of ways, it may not be readily apparent that a constraint in one operation schema is identical to another constraint in another operation schema. This reduces effort and the chance of error.

When writing operation schema in Z, having a fact model schema to look at more easily allows us to identify the blocks of actions which must be performed and conditions that must be met. For example we may wish to write an operation to add a fact and we may observe from the fact model schema that subset, exclusion or other constraints may relate to that fact type. We may notice from the schema that mandatory role constraints apply to roles in other fact types and as a consequence other facts also may need to be added within that operation. These extra facts may in their turn require further pre and post conditions to be included. All this is clearly visible from a graphical model such as the fact model.

Being bundled into one or more state schema, constraints will not be openly visible to someone viewing the operation schema. On the other hand, expressing constraints in operation schema for greater clarity may lead to problems as well. When there is duplication
of constraints across operation schema there may errors made in duplication. If a constraint happens to be expressed differently (possibly because different analysts developed the different schema) we need to be sure that they are truly equivalent and not subtly different. Contradictory constraints are not as easily detected.

**A sample specification in Z**

In the following I provide some sample operations specified in Z to demonstrate my particular style. I have used the video shop specification provided earlier and the sample operations are based upon the operations in (Barden, Stepney and Cooper 1994, Chapter 11).

Firstly, we consider how to express operation schema is such a way such that only the variables which may change are altered and all others are not. This can be accomplished by creating a state schema which only allows particular variables to be altered as described in (Barden, Stepney and Cooper 1994, p.107) as shown below.

\[
\Delta \text{StateOperation} = \Xi \text{State} \setminus (\text{variables, variables'}) \land \Delta \text{State}
\]

The effect of this is to create a state schema (StateOperation) in which all variables are unchanged (\(\Xi \text{State}\)) except those specified in the hiding list (variables, variables'). However, the variables in the hiding list are now unavailable. The \(\Delta \text{State}\) reintroduces the variables in the hiding list but without the predicates restricting the before and after states of variables to be equal.

\[
\Delta \text{VideoHire} = \Xi \text{Video_library_database} \setminus (\text{copy_is_hired_to_user, copy_is_hired_to_user'}) \land \\
\Delta \text{Video_library_database}
\]
The operation schema above does the following

- It checks that the person wanting to hire a video is a valid user (i.e. is in our database)
- It adds a new fact to the database regarding the copy borrowed and the user borrowing it
- It finds the corresponding hire price for the copy by determining which video has been borrowed, price band to be charged and hence the price. The mu expression relies on the fact that there are functional dependencies across each of the fact types in such a way that only one amount of money is possible

This operation schema assumes that all preconditions are met. For a more robust specification we may also specify what is to occur when preconditions are not met so we include some error conditions in the schema below. First, however, we create a given type called REPORT which provides some values corresponding to error conditions which can be used for reporting purposes.

REPORT := notUser | alreadyHired | tooYoung | OK

The schema below handles the condition where the borrower is not a valid member of the video library.

NotUser

The schema below handles the condition where the copy is currently hired out to another borrower.
The last error condition to consider is the one where a user is too young to hire a particular video. In order to accomplish this, we can introduce two components as is shown by Barden, Stepney and Cooper (1994), the variable today which records today’s date and mayHire which determines if a particular user is able to hire a particular depending on certain rules. I have not provided a definition of the function ageToday but merely state that it calculates the difference between two dates and provides the number of years (truncated) as an integer.

The set defined by mayHire records all the videos that a user is able to hire. The last part provides a constraint that

\[
\text{Video\_library\_database}\_
\text{refined}
\]

\[
\exists \text{Video\_library\_database} \\
\forall c?: \text{Copy\_is\_hired\_to\_user} \\
\exists \text{report!}: \text{REPORT} \\
c?.\text{copy} \in \{c: \text{copy\_is\_hired\_to\_user} \land c.\text{copy}\} \\
\text{Report!} = \text{alreadyHired}
\]

The schema below handles the condition where the borrower is too young to hire out the chosen video.

\[
\begin{align*}
\exists \text{Video\_library\_database}\_
\text{refined} \\
\forall v: \text{video} \\
\forall \text{today}: \text{DATE} \\
\exists \text{mayHire: USER} \leftrightarrow \text{VIDEO} \\
(_\text{mayHire}_) = \{u: \text{user\_born\_on\_date}; \\
v: \text{video\_classified\_as\_Certification}; \\
a: \text{N} | \\
a = \text{ageToday} (\text{today}, u.\text{date}) \land \\
\{a \geq 18 \\
v.\text{certification} \in \{\text{exempt, uCert, pg}\} \\
v.\text{certification} = \text{twelve} \\
\} \\
\{a \geq 15 \land v.\text{certification} = \text{fifteen}\}) \\
\land \\
\text{u.user} \leftrightarrow v.\text{video} \\
\forall c: \text{copy\_is\_hired\_to\_User} \\
(\exists v: \text{copy\_of\_Video} \land c.\text{copy} = v.\text{copy}) \\
\land \\
(c.\text{user} \land \text{mayHire} (v.\text{video}))
\end{align*}
\]

The schema below handles the condition where the borrower is too young to hire out the chosen video.

\[
\begin{align*}
\exists \text{Video\_library\_database}\_
\text{refined} \\
\forall c?: \text{Copy\_is\_hired\_to\_user} \\
v: \text{video} \\
\exists \text{report!}: \text{REPORT} \\
v = (\mu c: \text{copy\_of\_video} \land c.\text{copy} = c.\text{copy} \land c.\text{video}) \\
\land \text{mayHire} (c.\text{user} \land v) \\
\text{Report!} = \text{tooYoung}
\end{align*}
\]
The definition of the complete hire operation is then

\[
\text{Hire} = \text{HireOK} \land [\text{report!} : \text{REPORT} | \text{report!} = \text{OK}] \\
\lor \text{NotUser} \\
\lor \text{AlreadyHired} \\
\lor \text{TooYoung}
\]

As a further example of an operation I consider the addition of a new copy of an existing video. We again require a new state schema which only allows the appropriate variables to be altered.

\[
\Delta \text{VideoCopyAdd} = \exists \text{Video_library_database} \setminus (\text{copy_of_video}, \text{copy_of_video}', \text{copy}, \text{copy}') \\
\land \Delta \text{Video_library_database}
\]

<table>
<thead>
<tr>
<th>VideoCopyAdd</th>
</tr>
</thead>
</table>
| \[\Delta \text{VideoCopyAdd} \]
| \[c? : \text{Copy_of_video} \]
| \[c? \notin \text{copy_of_video} \]
| \[c?.\text{video} \in \text{video} \]
| \[\text{copy_of_video}' = \text{copy_of_video} \cup \{c?\} \]
| \[\text{copy}' = \text{copy} \cup \{c?.\text{copy}\} \]

- It checks that the fact about the copy of the video is not already in the database (although adding a duplicate fact would not actually alter the state of the database)
- it ensures that the video is an existing one (the set of videos should be altered since this is not supposed to be a new video i.e. not a new title)
- it adds the fact about the new copy to the database
- it adds the new copy to the set of copies

**Discussion**

Our state schema description contains quite number of variables which are expressed rather verbosely (in comparison to examples from the Z literature) since they are written as natural language sentences. Perhaps somewhat surprisingly, a comparison of my operation schemas produced here with those from by Barden, Stepney and Cooper (1994) shows that there is not a major difference in terms of the quantity or complexity of expression between the operation schemas. Because the variables are facts, in my approach the number of new variables required is reduced and the adding or deleting of facts to the state of the system is quite straightforward. Furthermore, in situations where facts of ternary or higher arity are involved
I do not foresee that the current quantity or complexity of expression observed is going to be significantly increased.

In terms of readability, I believe that seeing variables written in the form of facts and having operations within a schema that relate to facts makes the intentions in operation schema somewhat clearer. If that is the case, then it is easier to verify the correctness of the schema (in as much as a visual examination can do such a thing) and validate that what it does against what was intended.

One disadvantage of my approach to specification of a system, however, is that commonly used operators relating to functions and binary relations such as composition, inverse, identity, overriding, domain and range restriction and subtraction, relational image and so on are often not going to be applicable.

**Summary**

In this chapter, I described how the dynamics of a system are typically expressed in Z using schema to represent operations. Seeing that my representation of the state of a system requires the definition of relatively many variables which are expressed rather verbosely as natural language sentences, I believed it useful to examine how operations would be expressed using my approach. Continuing the video library specification from Barden, Stepney and Cooper (1994) already developed, I have taken some of the operations from Barden, Stepney and Cooper (1994) and recast them using my state schema. For these examples, I demonstrated that the specifications produced are no more complex than those of Barden, Stepney and Cooper (1994) and possibly easier to read.
Chapter 9

Introducing Identification Schemes into the Z schema

Introduction

In a business information system, correct identification of the entities of interest is a very important task. Poorly selected means of identification could lead to mistaken identification, incorrect processing and so on. For any business that relies on their information systems then the repercussions of not getting this right can be costly. In accord with this reliance, one component of the NIAM procedure is to determine if there is appropriate identification of entities. This involves deciding which labels (objects which can be physically expressed and understood) will be used to represent the entities (objects which are mental constructs) about which we wish to speak.

In an ORM schema, bridge types indicate ways in which an entity type can be referred. The names given to label types will indicate the “reference mode” e.g. the label type “customer id” attached to the entity type customer suggests that customers are referred to by something called their customer id. The entity type “car” attached to the label type “registration number” suggest that something called a registration number can be used to refer to a car. In the case of quantities that are numerically specified e.g. mass, length amounts of money and so on the label type typically indicates the unit of measurement e.g. kilograms, meters, Australian dollars respectively but the implication is that a number (and a unit of measurement) is used to refer to the entity concerned.

According to (Halpin, 2001, p94) there are only really only two types of label types, these are (character) strings and numbers (integers, real etc). All label types are subtypes (in the ORM sense) of those primitive types. However, to avoid clutter “the primitive value types and their subtype graphs are not displayed” (Halpin, 2001, p187). While I believe that restricting the label types to be subsets of strings and numbers to be a unnecessary I will accept this restriction since it does significantly affect the main thrust of this thesis. However, I do suggest that this could be an area for further investigation.

In an ORM schema, choosing an appropriate means of identifying the entities which make up an entity type is accomplished by examination of the bridge and idea types attached to the entity type and the associated constraints. This assists in the appropriate choice of label
type(s). I will call this the entity type’s identification scheme. I have used the term identification scheme rather than the term reference scheme since reference scheme could possibly be interpreted to mean “a way to refer to an entity” which may or may not be unique or mandatory or may be restricted to mean only labels directly attached via bridge types. In the normal or typical course of events, any candidate identification scheme requires that each and every entity of an entity type has a one to one (mandatory) correspondence with a set of labels. Furthermore, where there is more than one candidate identification scheme then only one such scheme is designated to be the primary identification scheme.

In terms of requirements specification and my translation procedure, whether we translate and implement the identification scheme within the Z specification or not depends on the simplicity or otherwise of the identification scheme, the purposes of the requirements specification and the particular requirements themselves.

In many cases, the identification scheme can be “understood” to exist but is omitted. A situation when this might be the case is when the identification scheme involves a simple one to one relationship between the entity type and label type such as when a student is identified by their student number or each bank account is identified by a bank account number. Even if the identification scheme is more complex it may be the case that this is not relevant; it is the properties of the entities and their relationships that is to be specified and the identification scheme is not necessary.

However, for the purpose of the thesis I assume that we do want to indicate the identification scheme. Two reasons could be cited for doing so:

1. Completeness: We might regard that an explicit statement of the identification scheme should be part of the system specification (the ORM perspective) if such a scheme has been chosen at all.
2. Access to properties of the different types making up the identification scheme: The properties that apply to the label types are not the same as those that apply to the entity type.

We consider two examples.

- Suppose that we wish to include in the specification allowed values of an identification scheme. These are defined by the constraints on label types (e.g. strings or numbers) and not on the entity type itself. By including the identification scheme
explicitly, we can indicate the restrictions on the scheme which could not otherwise be specified.

- Suppose that we have an entity type MASS (the physical quantity measured in kilograms) declared as a basic type in a Z schema. If, in our specification, we want to add or subtract masses or to perform other numerical calculations then we are unable to do so since MASS will not have the numerical properties we require.

In this chapter I suggest some possible approaches to identification schemes in a Z specification. I consider first the one (entity type) to one (label type) identification schemes then one to many and finally reference schemes where idea types are involved.

Identification schemes requiring only one label type

We consider first identification schemes where a single label type is sufficient to identify an entity type.

Implied identification scheme

The simplest situation is one in which the number of labels is quite small and can be enumerated. The allowed labels of the label type represent the entities to which they refer and the values enumerated correspond to the labels. Recall for instance from the Video Library example from Chapter 7

```plaintext
MEMBER_TYPE ::= staff | club | member
```

Here, “staff”, “club” and “member” are constants which represent the values of MEMBER_TYPE. These constants have the properties of the type MEMBER_TYPE and not character strings. The identification scheme is incorporated through the specification writer choosing names for the values that correspond to the names meaningful to the application. On the other hand the specification writer could just as easily have chosen “x”, “y” and “z” in these circumstances. There is no explicit identification scheme in this sort of situation.

More typically, the number of possible labels for a label type is extensive. For example, an employee might be identified by a identification number, or a business by its business name and so on. In these cases it would not be feasible or necessary to express these explicitly. If a particular value was required to be identified in the specification, then a constant of that type could be declared.
Explicit scheme

Because an identification scheme requires a 1:1 mandatory relationship between entities and labels, injections can be used to associate labels with entities and thereby introduce both the entities and labels. For example, consider the example of a customer of a business. We have the entity type, say CUSTOMER, and their customer identification, say CUST_ID which is intended to be a set of values drawn from the set of STRING, we could define a function customerID and then define

\[
\text{[CUSTOMER]} \quad \text{CUST}_\text{ID} = \text{STRING}
\]

\[
\text{customerIS} : \text{CUST}_\text{ID} \to\to \text{CUSTOMER}
\]

CUST_ID has been made equivalent to the set of all possible string values STRING but in practice will be some subset of this. The injection in this way allows us to refer to the customer using a given customer identification number, say C123, as in customerID C123 or given a customer (say c) we can refer back to their customer identification number as in customerID c.

Numeric values

Some entity types are measured (identified) numerically. For example, one such group is the set of entity types which are physical quantities such as mass, length, time etc and quantities derived from those. There are many entity types which are also measured (identified) numerically. When we want to access their numerical properties one approach could be to equate the type to an appropriate numerical type such as integer or real numbers. This allows us access to the numerical properties. For example we could write

\[
\text{LENGTH} = \mathbb{R}
\]

However, in doing so we have lost the independent concept of LENGTH (except as a name). If there are other types (MASS, TIME, MONEY ...) which are numerically based and handled the same way, then we allow the possibility of inappropriate type combinations or numerical calculations (e.g. adding a mass to a length). We place the burden of ensuring the integrity of the Z specification heavily on the care of the specification writer.

---

25 In the ORM schema the requirement for an appropriate identification scheme is that every customer that is recorded in any fact must have a customer identification and each identification can only be associated with one customer hence we have made this a total injection.
From the ORM perspective, this approach loses much of the semantics of the ORM schema. Even the units of measurement which are normally included as part of the label type in the ORM schema are lost unless it too is somehow included in the name of the type e.g. say, something like Mass_in_kg.

**Maintaining dimensional consistency**

An alternative approach to dealing with entities which are physical quantities is described by Barden, Stepney and Cooper (1994, pp322-325). In this approach the basic types MASS, LENGTH, TIME etc are replaced with a single basic type UNIT defined in the following way:

\[
\text{UNIT} ::= \mathcal{L} | \mathcal{M} | \mathcal{T}
\]

Variables are defined which connect a number with appropriate dimensions.

\[
\begin{align*}
\text{dimension} &= \text{UNIT} \rightarrow \mathbb{Z} \\
\text{physical} &= \mathbb{R} \times \text{dimension}
\end{align*}
\]

Operations are defined which allow addition, subtraction, multiplication and division of dimensional quantities while determining the correct dimensions of the result. Units of measurement (kg, m, s, ms$^{-2}$ etc) can also be associated with particular dimensions or combinations of dimensions.

This approach is quite sophisticated and likely to be useful in some scientific or engineering application but probably more complex than needed in the typical business application.

**A compromise approach**

An approach which is in between the previous two approaches is to use a injection to relate the numerical value to the entities being referred. For example,

\[
\text{[LENGTH]}
\]

\[
\begin{align*}
\text{METERS} &= \mathbb{R} \\
\text{lengthIS} : \text{METERS} &\rightarrow \text{LENGTH}
\end{align*}
\]

---

26 The original example has been simplified as the additional details do not add anything for my purposes.
Here I have used the name of the label type to bring into the Z specification the unit of measurement although it is just an alias for the set of real numbers. If we write lengthIS 23 the type of this is LENGTH so attempting something like the expression

\[
\text{lengthIS } 23 + \text{lengthIS } 15
\]

is incorrect. However, if we had two variables X and Y of type LENGTH we can add these lengths together as in

\[
\text{lengthIS}^\prime X + \text{lengthIS}^\prime Y
\]

since these are both real numbers.

Suppose now that another numerically measured quantity is involved such as MASS.

\[
\text{[MASS]}
\]

KILOGRAM = = \mathbb{R}

massIS : \text{KILOGRAM} \rightarrow \text{MASS}

If we have a variable M of type MASS, while a type checking program would allow

\[
\text{lengthIS}^\prime X + \text{massIS}^\prime M
\]

since the two terms here are both real numbers, visually it is glaringly obvious that there is a problem here.

This approach is a compromise, of course, but it addresses some of the problems of simply reducing every numerically based entity type in the Z specification to the same numeric type while not being very elaborate.

*Identification schemes involving more than one label type*

A simple example of an identification scheme involving the use of more than one label type is shown below in which an entity type customer is identified by the label types FirstName and LastName.
Where more than bridge type is required to identify an entity, schema types can be used to encapsulate the identification scheme. In the example above we can create a Customer identification scheme as shown below:

```
CustomerIDScheme
firstName: FIRSTNAME
lastName:  LASTNAME
```

We can use an injection to relate the identification scheme to the corresponding entity Customer and CustomerIDScheme as follows

```
[CUSTOMER]

CustomerIS: CustomerIDScheme \rightarrow CUSTOMER
```

Each component of the identification scheme can be referenced using the dot notation.

We prefer to use a schema type to refer to the components rather than a cartesian product such as

```
FIRSTNAME = = STRING
LASTNAME = = STRING

CustomerIS: (FIRSTNAME x LASTNAME) \rightarrow CUSTOMER
```

Accessing the components of the identification scheme is less intuitive because we have to use functions such as `first_` and `second_` to project out the required components. Furthermore, it is quite possible that more than two components might make up an identification scheme and projecting out the required component then becomes more tedious whereas the schema type approach with its ability to use the dot notation to project out the required component is not affected by an increase in the number of components.

**Identification schemes where an idea type is involved**

The last case I consider is where an entity type is identified through one or more idea types and possibly by one or more bridge types. Where an idea type is involved the attached object type is an entity type. This may suggest a component of the identification scheme which is
based on the using an entity rather than the label type. However, the basic philosophy is that
an identification scheme must be composed entirely of label types i.e. components ultimately
of the types STRING or NUMBER. Hence, where idea types are involved we add the label
type components of the identification schemes of the attached entity types to whatever other
label types directly make up the identification scheme. If the attached entity type itself has an
identification scheme that involves another entity type then we repeat this process
progressively until all components are either STRING or NUMBER. This is consistent with
Halpin (2001, p202) who describes it in this way (note that I use the term label instead of
value)

"Each value is referenced directly using a sequence of one or more objects,
which in turn may be values or entities. In the latter case, the reference chain
continues, recursively until finally the referencing objects are all values."

It is assumed that no cycles of reference occur that lead to some infinite chain i.e. in resolving
the identification schemes a chain of references forms a cycle. Issues to do with these cycles
should have already been resolved as part of the NIAM process and therefore prior to the
translation process.

In the diagram below, for example, a Track is uniquely identified by a track# and CD. CD is
an entity type which, in its turn, is identified by a CD title. Ultimately therefore, a Track is
identified by the labels track# and CD title.

![Figure 32](image)

In translating these identification schemes into Z, the schema type approach mentioned earlier
groups all components together.
Translating this into Z, we have the following basic types

[TRACK, ARTIST, CD, DATE]

TRACK# = N  
CDTITLE = STRING  
ARTISTNAME = STRING  
DDMMYY = STRING

-TrackIDScheme-
  track#: TRACK#
  cdTitle: CDTITLE

-ArtistIDScheme-
  artistName: ARTISTNAME
  dateOfBirth: DDMMYY

cdIS: CDTITLE \rightarrow CD
trackIS: TrackIDScheme \rightarrow TRACK
artistIS: ArtistIDScheme \rightarrow ARTIST
dateIS: DDMMYY \rightarrow DATE
The following schema types correspond to the idea types.

<table>
<thead>
<tr>
<th>Artist_performs_Track</th>
</tr>
</thead>
<tbody>
<tr>
<td>track: TRACK</td>
</tr>
<tr>
<td>artist: ARTIST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Track_is_on_CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>track: TRACK</td>
</tr>
<tr>
<td>cd: CD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Artist_was_born_on_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>artist: ARTIST</td>
</tr>
<tr>
<td>dateBorn: DATE</td>
</tr>
</tbody>
</table>

**Populating fact types and synchronizing label types**

Populating a fact type, i.e. instantiating a fact type with facts, is performed by considering relevant facts and replacing the entities in the facts with their corresponding labels based on their identification scheme. Before we have an identification scheme determined for an entity type, it may be that when using the current labels available the facts as shown do not obey the constraints. This is because the constraints apply to the entities involved and not on the labels. For example, suppose that we have a fact type

The student … is enrolled in the course …

We know that students can only be enrolled in one course at any single point in time. Hence a uniqueness constraint applies to the student role as shown below.

![Diagram showing the enrollment relationship between student and course](image)

**Figure 34**

However, if we simply use a persons first name to populate this fact type we get what appears to be an error, e.g.,
This could happen if there are two distinct and different students who have the name Bill. The fact type and constraint is correct but the way in which we refer to the entity types Student and Course is not adequate for the purpose of populating the fact type and also obeying the constraint. On the other hand, if a (unique) student number were available to refer to students in the example above then our apparent violation of the constraint would not occur.

If it is necessary to use a combination of label types to identify an entity then we can represent the combination as <LabelName₁ = L₁, LabelName₂ = L₂, LabelName₃ = L₃ ... >. For example, if a student's first name and surname were adequate as an identification scheme then we could populate the fact type above as follows:

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; Firstname = Bill, Surname = Jones &gt;</td>
<td>Bach of Science</td>
</tr>
<tr>
<td>&lt; Firstname = Mary, Surname = Green &gt;</td>
<td>Bach of Computing</td>
</tr>
<tr>
<td>&lt; Firstname = Angela, Surname = White &gt;</td>
<td>Bach of Science</td>
</tr>
<tr>
<td>&lt; Firstname = Bill, Surname = Brown &gt;</td>
<td>Bach of Medicine</td>
</tr>
</tbody>
</table>

We have seen that in more complex identification schemes that some idea types describe relationships between entities and at the same time they also form part of the identification scheme of an entity type. This dual role requires that when we try to populate a fact type that has an entity that also happens to use that fact type for its identification, we need to ensure that the labels on either side of the fact type are synchronized.

For example, if we use the track and CD fact type from an earlier example,

Track … is on CD …

then when populating the fact type we should get something like this:
i.e. the CD title that identifies a track must always be the same as that of the CD that it is on.

It would not be sensible in this context to have a fact which states that a track which is identified to be on one particular CD (Blue Hills say) is actually on another (Orange Fields say). While this may appear redundant, it is a redundancy introduced due to the selection of the labels required to identify the entities involved but not of the facts or underlying entities.

When we make identification schemes explicit we need to also introduce constraints that ensure that the labels in fact types involved in identification schemes stay synchronized. We can express the required constraint in the following way.

Assume \( F \) is a binary idea type which is also used as part of an identification scheme for an entity type \( E_0 \). The entity type on the “other side” of \( F \) is \( E_1 \).

The fact type \( F \) is expressed as

\[
\begin{array}{c}
\text{e}_0 : E_0 \\
\text{e}_1 : E_1 \\
\end{array}
\]

We can define a set of facts of this type

\[ f : F F \]

\( E_0 \) and \( E_1 \) have the identification schemes \( E_0 \text{ID}_\text{scheme} \) and \( E_1 \text{ID}_\text{scheme} \) respectively.

Suppose now that \( E_1 \text{ID}_\text{scheme} \) contains the variables \( c_{i1}, c_{i2}, \ldots, c_{in} \). The identification scheme for \( E_0 \) will necessarily contain a set of variables matching this set which we will label \( c_{o1}, c_{o2}, \ldots, c_{on} \). By matching, we intend that the type of \( c_{ii} \) is the same as \( c_{oi} \), where \( i \) ranges from 1 to \( n \).

We now define the following injections to represent the correspondence between identification schemes and objects

\[
e_{0IS} : E_0\text{ID}_\text{scheme} \rightarrow E_0 \\
e_{1IS} : E_1\text{ID}_\text{scheme} \rightarrow E_1
\]
The required constraint which then ensures that when we populate the fact type with labels that labels are synchronized is then

\[ \forall i:1..n( \forall f:F \mid (e_{i}\text{IS} \sim (f.e_{i})).c_{i} = (e_{i}\text{IS} \sim (f.e_{i})).c_{i} ) \]

**Exceptions**

While I have described a variety of ways of handling identification schemes, I have considered only those types of identification schemes which are relatively straightforward. In the “real world”, situations occur which do not fit into the patterns I have thus far considered. For example, two businesses might merge. For a time, there might be two identification schemes in place for employees. These might at times be mutually exclusive or overlapping before finally a single identification scheme exists. In another situation, a business might prefer to have different identification schemes for different classes of employee. There are many possibilities and not all of them need be very sensible since they might reflect “political” compromises or the legacy of the past. These will need to be handled on a case by case basis.

**Translation procedure revisited**

I have suggested a variety of options for handling identification schemes. These range from a minimal or no incorporation of identification schemes through to a full implementation. The decision as to the extent of implementation schemes clearly should be made right at the beginning of the translation procedure. Although I have not investigated the problems related to adding or removing identification schemes after the translation process has taken place, it would appear to involve more work and may introduce unnecessary problems.

Whether to include identification schemes in the translation procedure depends on the purpose of the specification. If it is chosen to implement few or no identification schemes some care needs to be taken regarding what is to be included and what is to be omitted. This is best done on the basis of a clear understanding of what is required to identify an object. For example, should all labels be omitted? Since not all labels are used for identification we need to be aware that we may be omitting required attributes from the specification. If we haven’t decided on an identification scheme for each object type, how do we know what label types to
include or exclude? Should idea types that are used in an identification scheme be included in
the translation to the requirements specification in Z or can they be omitted?

**Conclusion**

In order for the translation procedure to be complete i.e. all aspects of the ORM schema have
been translated into a Z specification, information about identification schemes and any
constraints that apply to the identification scheme should also be included. I have discussed
how we might handle identification schemes of various levels of complexity ranging from
simple schemes where there is a one to one relationship between labels and entities through to
more complex schemes where an object is identified using a combination of labels and
through the identification schemes of other attached objects. I noted that identification
schemes in which idea types participated required an additional constraint to be applied to the
labels populating each of the participating idea types. I provided the general form of this
constraint.

Using identification schemes means that we distinguish between an object and its label. This
distinction means that we can also distinguish between the properties of the entity as opposed
to the properties of the label(s) used to identify it. Furthermore, in the case of numeric
quantities I also proposed that we did not have to implement necessarily all the properties a
label type might involve but sufficient to allow us to write the specification. Selecting a
particular identification scheme was essentially a trade off between completeness in terms of
properties and avoidance of type errors (e.g. through inclusion of units of measurement) and
simplicity while still being able to express requirements.

Finally, I suggested that an understanding of the identification schemes is important in the
translation procedure in order that the writer of the specification be able to make informed
choices about the entity types and fact types that could safely be omitted from a specification
should they wish to omit identification schemes.
Chapter 10

Evaluation and Conclusion

Assessing the work

I suggested at the beginning of this thesis that I would evaluate the results of my translation procedure according to the criteria for conceptual modeling from Loucopoulos and Zicari (1992, p15). This evaluates it using criteria for “good” or desirable properties for a conceptual model of requirements. I do this below. I have included in italics the relevant quotation for each criteria from Loucopoulos and Zicari (1992) followed by my comments.

Implementation independence

“No implementation aspects like data representation, physical data organization and access as well as aspects of particular user representation, such as message formats, data structures etc.) should be included in a requirements specification.”

An ORM schema is intended to be an implementation independent specification of requirements. It is merely a statement of the types of facts which need to be stored and constraints that apply to those facts from a business perspective. In so far as the ORM schema is implementation independent then the Z schema is also implementation independent.

It can be suggested that representing data requirements in larger data structures (such as in a record/table format) might well be regarded as a form of implementation dependence. In ORM, because fact types represent the smallest logical constructs that can be created without loss of information I would argue that this provides greater implementation independence.

Abstraction

“Only general, (i.e. not subject to frequent changes), static and dynamic aspects of an information system and the UoD should be represented in a requirements specification”.

Batini, Ceri and Navathe (1992, p 15) describe abstraction as

“a mental process that we use when we select some characteristics and properties of a set of objects and exclude other characteristics that are not relevant”.
In this case, the features that have been selected (abstracted) relate to those structures required for the recording of data within some proposed information system. The abstraction process has taken place earlier within the NIAM process when the analyst elicited from users those facts, entities and labels and the corresponding types which were necessary to be recorded. The final translation into a Z schema provides a formal representation of those abstractions.

**Formality**

“Descriptions should be stated in a formalism with unambiguous syntax which can be understood and analysed by a suitable processor. The formalism should come with a rich semantic theory that allows one to relate the descriptions in the formalism, to the world being modelled”.

Formality is often associated just with being expressed mathematically (especially based on some sort of logic system or algebra e.g. Z) however an ORM schema provides a degree of formality as well. When combined with Z as I have in the form of my translation procedure, then we have produced artifacts which provide a formal description together with a clear relationship to the UoD through object and fact types and the requirement that all fact and object types be able to be populated with facts from the UoD.

**Constructability**

“A conceptual schema should be constructed in such a way so as to enable easy communication between analysts and users and should accommodate the handling of large sets of facts. In addition, a specification needs to overcome the problem of complexity in the problem domain, by following appropriate abstraction mechanisms which permit decomposition in a natural manner. This calls for the existence of a systematic approach to formulating the specification.”

I tackle the last section first. The ORM schema is intended to capture data requirements through the analysis of natural language and the representation of facts in a manner which maintains their natural language style. ORM provides suitable abstraction mechanisms for data analysis. Thereafter, my translation from ORM to Z maintains these abstractions. As for a systematic approach, NIAM provides this systematic approach which eventually culminates in the ORM schema. Z is not directly associated with any systematic approach to development of the specification although it is tailored to suit specifications for an information system.
On the first point regarding “easy communication”, I have chosen to interpret this as an ability of the user to understand the schema with little or no assistance. The discussion in Chapter 1 makes it clear that Z schema is not suitable for most users as a presentation medium since most users do not have the mathematical background or aptitude to understand a Z schema. Even if the user had the aptitude, there is the problem of the time it would take to train the users in Z. This is not viable in many cases.

However, because of the fact that we have a direct translation between an ORM and Z schema, we can present the user with an ORM schema instead. The ORM schema will be understandable by a wider audience. In its turn, if the ORM schema is still not suitable it can be directly translated to a form of formal structured English statements (as per Halpin (1989) with his language FORML). This means that there are a variety of forms which can presented to users (or system builders) all of which are translatable across the various forms so that errors are avoided when going from one form to the other.

With regard to the other elements of constructability, ORM was designed to handle large sets of facts and complex domains (at least of the type seen in business systems) and hence the translated Z schema also preserves these features.

**Ease of Analysis**

“A conceptual schema needs to be analysed in order to determine whether it is ambiguous, incomplete or inconsistent. A specification is ambiguous if more than one interpretation can be attached to a particular part of the specification. Completeness and consistency require the existence of criteria against which the specification can be tested. However, the task of testing for completeness and consistency is extremely hard simply because no other specification exists against which the specification can be tested.”

A Z schema has advantages in that type checkers can discover syntax errors and theorem provers can help to discover omissions, ambiguities and inconsistencies once the dynamic aspects of the system are also included in the specification. On the other hand, an ORM diagram provides a visual aspect which also allows the viewer to discover problems with data. A description in a formal English style will help business users who are unfamiliar with requirements expressed in formal or semiformal notations.

Each representation format has its own particular advantages and disadvantages toward discovering ambiguity, incompleteness and inconsistency. It would seem desirable that users
be able to examine a specification and be able to understand it directly. Having a variety of different but equivalent representation formats aimed at different audiences means that there is more chance that the viewer of the requirement will understand the requirements, discover any omissions, unwanted elements or inconsistencies.

**Traceability**

“Traceability refers to the ability to cross reference elements of a specification with corresponding elements in a design specification and ultimately with the implementation of an information system.”

As currently described, traceability relates to elements which are further “downstream” in the systems development lifecycle. As has been well described in the literature on Z, Z schemas can be evolved into design and implementation forms with the ability to prove that the schemas produced are mathematically equivalent to each other. This is much stronger requirement than the ability to “cross reference”.

I would like to extend the idea of traceability to cover also the ability to cross reference “upstream” as well. Because we have a direct translation from ORM to Z schemas we reduce the chance of errors creeping into the system development process when changing from one means of recording requirements to another. If we also allow the fact that an ORM schema can be expressed something like Halpin’s (1989) formal, but English like, language FORML and vice versa, then we have “provable” traceability from statements in English (which users can write for themselves directly or with some help), through a variety of intermediate forms which have different purposes, to some formal specification that can be implemented directly.

**Executability**

“The importance of this property is in validation of the specification. Executability refers to the ability of a specification to be simulated against some facts relevant to the modeled reality. The executability of the description in a schema is subject to the employed formalism.”

Z is not executable directly. To an extent ORM is executable in that facts (i.e. in the sense of pieces of information that are to be recorded) can be stored in or retrieved using the ORM schema to provide a database (of elementary fact types) and the constraints acting as filters to screen out input of inappropriate facts. This approach is described as a “conceptual information processor” by Halpin (2001). Since we are considering just the storage

---

27: downstream” as in closer toward the final implemented system
requirements of the information system then executability would refer to its ability to store data input into the system and produce required information. It should be noted that by insisting that no artificial entity types, fact types or identifiers should appear in the ORM schemas that are used in the translation procedure in this thesis, executability has been enhanced. Furthermore, ORM schemas can be translated directly, with all constraints implemented, into DDL statements for a variety of databases.\(^{28}\) This means that one can easily create a working database in order to test the correctness of a schema with data from the problem domain.

**Summary of evaluation**

I believe that the result of my work has lead to a requirement statement which satisfies the criteria of abstraction (ORM schema provides suitable abstractions), implementation independence (ORM schema provides this) and my translation procedure to a Z schema provides formality. With regard to constructability, the NIAM approach provides a systematic approach to formulating the ORM schema. The object role model schema is designed to handle large sets of data and handle complexity in the problem domain (restricted to domains targeted toward business systems). Being able to represent requirements in a variety of equivalent ways allows for ease of analysis from a number of view points and traceability. Finally, my required form of ORM schema that uses n-ary facts and that does not contain artificial constructs makes it easy to take facts directly from the problem domain and test the validity of the schema.

**Summary of Thesis Chapters**

In this section I provide a brief review of each chapter.

Chapter 1 reviewed the literature on informal, semi-formal and formal methods and determined their acceptance in the IT community as well as what are seen to be their advantages and disadvantages. I agreed with the literature that suggested that hybrid systems which made use of both approaches at various stages could be more beneficial than one only. I provided arguments for producing a translation procedure from ORM to Z.

Chapter 2 reviewed the work on converting data models to Z. In particular, I examined the SAZ method which converts the products of the SSADM method to Z specifications. One of those products which is converted is a form of entity relationship model. With respect to the

---

\(^{28}\) In (Halpin 2001) such a procedure is provided. The procedure is described by Ritson (1994) and was
ER conversion process, I looked at the start and end points of the process and criticized it from the point of view that the process has to undo certain implementation details, leaves in what I believe are unnecessary entity types and criticize the selection process of basic types in the Z schema.

Chapter 3 examined the work on translation procedures from ORM to Z. I found one previous work suggested approaches and guidelines and another work provided a systematic translation process. However, none of the translation procedures were complete in providing a translation of all aspects of an ORM schema nor did they provide a satisfactory conceptual basis for their translation procedures.

Chapter 4 looked at various approaches to modeling in ORM and selected one of these approaches as being the most appropriate for the purposes of translation since it did not introduce any artificial entities into the schema. Such artificial entities were construed as a form of implementation dependence and hence were avoided.

Chapter 5 considered the fundamental concepts of ORM and examined the abstractions available in Z in an attempt to find the ones most suited to “carrying” those abstractions. I suggested an approach which I believe is “intuitive” and relatively simple.

Chapter 6 provides a pattern for translation into a Z specification for each type of constraint in ORM.

Chapter 7 described the translation procedure and provided fairly complex examples to illustrate and demonstrate most aspects of the translation procedure.

Chapter 8 discussed dynamics. Dynamics were interpreted as the transactions that were to performed on an information system. The purpose of the chapter was to get an idea of the repercussions on transactions of expressing specifications in Z according to my method. I used an example from the literature, produced an ORM schema for that example, translated it to Z schema using my procedure and provided some transaction schemas. While the writing style of the transaction schemas were naturally different, my transaction schema did not appear to be any more or less complex than in the original example. I took this as a good indicator.

implemented in the software package Infomodeler (no longer available).
Chapter 9 was devoted to considering identification schemes. Identification schemes may be omitted from the translation procedure. However, should one want to include the identification scheme, I provide a comprehensive procedure which allows for identification schemes of any complexity which can be expressed using the ORM schema. These identification schemes may include the use of multiple label and/or idea types.

What has been achieved

Background comments

I have reviewed the literature regarding formal and informal specifications and found, as others have, that informal or semi-formal specifications have wide acceptance in the IT community but that formal methods do not. This is so despite compelling reasons for using formal methods based their ability to be precise and unambiguous as well as providing the ability to reason about the system. Hybrid approaches combining informal, semi-formal and formal methods have been suggested as a solution and there have been developments along those lines. This thesis is another such development using ORM and Z.

I chose ORM because it provided a very detailed description of the data requirements of an information system and did not represent data in an implementation dependent manner. Furthermore, an ORM schema is the culmination of the step by step procure NIAM for the data analysis of a system. On the other hand, Z was chosen because it is widely used in the formal methods community and its approach to representing specifications is quite compatible with several ORM concepts.

I now describe what I believe has been achieved.

A complete translation procedure

Previous ORM to Z translation procedures are not complete in that they do not include all the features provided by the ORM model. My procedure translates all standard concepts and constraints.

A prescriptive translation procedure

A few translation procedures from ORM to Z have been proposed but they do not provide a single prescriptive translation from one notation to the other. They might demonstrate possible translations of a concept but leave a great deal of discretion as to the final product. This means that there is still input required in order to complete the translation. My procedure
was intended to be prescriptive and so provide only one possible translation. The translation procedure is prescriptive enough to be automated.

**A mapping process preceding development of a translation procedure**

I have considered the most appropriate approach to take in terms of moving from ORM concepts to concepts in Z. This was the mapping process. I believe that this has lead to a simple and intuitive style of translation. For example, fact types, entity types and label types each have their own form of representation and within each type there is similarity of treatment. Justifications for choosing each representation style were provided. The basis for selecting a representation was justified using the criteria from the framework for evaluation of a conceptual schema shown in Appendix A.

**A basis for selection of basic types**

I have noted that selection of an appropriate set of basic types is a topic that is not addressed well in the formal methods literature. I provide a clear and simple approach to the selection of basic types in the Z schema which I believe provides all the basic concepts of interest in the specification from which other concepts required in the specification can be derived. This may be of interest not only to system developers but also to those involved in teaching Z.

**Explicit implementation of identification schemes if required**

The NIAM procedure requires the establishment of identification schemes for entities. However, in some circumstances identification schemes may not be of interest in a Z specification. Accordingly, identification schemes can be added to the translation procedure. These can be translated fully as I have described a means to implement identification schemes and provided several examples. These can handle all identification schemes of any complexity which can be represented in an ORM schema. In the case of identification schemes which involve idea types, this entails the introduction of further constraints in the Z schema. If identification schemes are included, I provide for different properties for the entities compared to the properties of labels.

**Maintenance of the distinction between entities and labels**

I maintain the distinction between labels and entities. There is a consistent treatment of label types, for example, first names and surnames are both treated as strings but there may be different constraints operating on allowed values of different label types so that they form different subsets. Some specifications in the literature label types appear each with their own...
(non string or numerically based) basic types. This prevents the possibility of comparing or combining them or accessing their properties as string or numeric values.

**Preliminary investigation into dynamics**

I have also made a preliminary investigation, using examples, into the dynamic aspects of Z specifications in order to gain some idea how they might be affected using the Z schema produced of my translation procedure. The results suggest that the Z schemas produced do not appear any more complex than ones produced in the more conventional manner.

**Use of a framework to guide and evaluate the translation procedure**

Using the criteria for evaluating a conceptual schema I believe to be a novel way to assess this type of work. The criteria for evaluation have been a useful guide when making decisions about which direction to take in this own work. Other authors have suggested advantages for hybrid approaches in general and for specific translation procedures but not based on such a framework for evaluation.

**Detailed description of the ORM features that have been translated**

Entity types are translated as basic types. Label types are also translated as basic types but are reduced to the types STRING and NUMBER. Some label types may have constraints on their values and so are subsets of these possible types.

Fact types with arity > 1 are treated in a consistent fashion. The schema type is used as pattern for representation of fact types where each schema type represents one fact type and each component variable of the schema type represents one of the roles of the fact type. In the translation, the name of the schema type is derived from the fact type sentence so that the semantics of the ORM schema is maintained. Though this produces “meaningful” names I acknowledge that they might be considered somewhat clumsy to use.

The translation procedure provides constraint patterns for all constraints which I have described as the standard (most widely used) ones in ORM.

- Internal uniqueness constraints. This covers fact types of any arity and where multiple uniqueness constraints apply across the one fact type, each constraint is specified separately and in the same fashion.
• External uniqueness constraints. These apply to multiple binary fact types in which one role is of the same type. Any number of binary fact types is handled using the same pattern.

• Mandatory role constraints are covered.

• Disjunctive mandatory role. Any number of roles may be connected together to the same entity type to form the disjunctive mandatory constraint.

• Equality constraint. This can be applied to multiple roles across two fact types of any arity.

• Exclusion constraint. This can be applied to multiple roles across two fact types of any arity.

• Subset constraint. This can be applied to multiple roles across two fact types of any arity.

• Subtype constraint. I have provided guidelines how subtypes can be represented in Z. Subtypes in ORM are implemented as a form of subset constraint in Z.

• Value constraint. This assumes the translation of label types into the Z schema and the translation into Z of an identification scheme. Value constraints will be applied as subsets of the more general types STRING or NUMBER.

The overall result of my work is then to provide a comprehensive translation procedure that builds on and extends the work of others in this area.

**Further work**

Just as NIAM is used as a systematic method for capturing and representing the data aspects of a system, it is worth considering an appropriate method for capturing and representing the dynamic aspects of the system.\(^{29}\) This, with its own translation procedure which is consistent with the current work, would provide a complete and formal functional description of the system.

Z provides a general purpose toolset which can be used to specify a wide variety of types of systems. Associated with that, there are many ways to express the same requirement(s). This flexibility has advantages in some environments (such as research), but in the commercial

\(^{29}\) In some earlier work the object role model was part of a larger procedure which included a process model (Nijssen and Halpin 1989, Verheijen and van Bekkum 1982). However, today something like use cases (UML) may be more appropriate.
environment flexibility may actually be a disadvantage. Some of the pressures in a commercial environment include

- becoming adept with the specification language as quickly as possible;
- rapid production of requirements and
- creativity needing to be tempered with conformity to standards of presentation among different analysts and over time.

With these types of pressures, a small but well tailored tool set specifically targeted toward data intensive information systems together with a standard approach to specification of such systems appears to be worthy of further investigation. While this work provides an example of something on these lines, research is needed to determine the most appropriate tool set and approach.

Since my translation procedure is as quite prescriptive, automation of the process should be a straightforward process.

With the emphasis on object oriented software development which exists currently, it may be worthwhile to investigate the possibility of either a direct translation from the ORM schema to an object oriented formal language like Object Z or from the product of my translation procedure to something like Object Z.

Object relational database management systems of varying sophistication and complexity are now available (for example Oracle 9i now provides a reasonably mature product). Investigation of the usefulness of this work and object relational systems may be useful.

The use and development of components (as opposed to “objects”) as the building blocks of software is currently of more interest since Microsoft has moved significantly into this area. Since the aim of components is reusability, it will be necessary to ensure that components conform to requirements and are robust. NIAM and the object role model together with a formal description of the data requirements may well be very useful to component developers. Investigation into this area seems worthwhile.

Further work needs to look at the repercussions of the products of the translation procedure when trying to add dynamic aspects of the system requirements and further on to refinement of specifications toward implementation. There may be advantages and disadvantages which currently are not apparent.
Bibliography


Creasy PN, 1983, A Formalisation of the NIAM conceptual information model, Dept of Computer Science, Australian National University


Date CJ, 2000, An Introduction to Database Systems, 7th Ed, Addison Wesley Longman


De Troyer O, 1994, ‘A Logical Formalization of the Binary Object-Role Model’, in ORM-1 International Conference on Object Role Modelling, Townsville, Australia


James JA and Olson N, 1994, ‘Fact based Corporate Data Modelling’, *Proceedings First International Conference on Object Role Modelling ORM – 1*, Townsville, Australia


van Bommel P, ter Hofstede AHM, van der Weide TP, 1991, ‘Semantics And Verification Of
Object Role Models’, *Information Systems*, 16(5) October, pp. 471-495


Woodcock JCP, 1989, ‘Structuring Specifications in Z’, *Software Engineering Journal*


Appendix

Requirements for a Conceptual Schema
from Loucopoulos and Zicari (1992, p 6,7)

Implementation independence

“"No implementation aspects like data representation, physical data organization and access as well as aspects of particular user representation, such as message formats, data structures etc.) should be included in a requirements specification.”

Abstraction

“"Only general, (i.e. not subject to frequent changes), static and dynamic aspects of an information system and the UoD should be represented in a requirements specification”.

Formality

“"Descriptions should be stated in a formalism with unambiguous syntax which can be understood and analysed by a suitable processor. The formalism should come with a rich semantic theory that allows one to relate the descriptions in the formalism, to the world being modelled”.

Constructability

“A conceptual schema should be constructed in such a way so as to enable easy communication between analysts and users and should accommodate the handling of large sets of facts. In addition, a specification needs to overcome the problem of complexity in the problem domain, by following appropriate abstraction mechanisms which permit decomposition in a natural manner. This calls for the existence of a systematic approach to formulating the specification.”

Ease of Analysis

“A conceptual schema needs to be analysed in order to determine whether it is ambiguous, incomplete or inconsistent. A specification is ambiguous if more than one interpretation can be attached to a particular part of the specification.Completeness and consistency require the existence of criteria against which the specification can be tested. However, the task of testing for completeness and
consistency is extremely hard simply because no other specification exists against which the specification can be tested.”

**Traceability**

“Traceability refers to the ability to cross reference elements of a specification with corresponding elements in a design specification and ultimately with the implementation of an information system.”

**Executability**

“The importance of this property is in validation of the specification. Executability refers to the ability of a specification to be simulated against some facts relevant to the modeled reality. The executability of the description in a schema is subject to the employed formalism.”