FORMAL CONCEPT ANALYSIS APPLICATIONS TO REQUIREMENTS ENGINEERING AND DESIGN

Thomas Tilley
B.Sc.(Maths & Comp. Sc.), B.Info.Tech.(Hons)

THE UNIVERSITY
OF QUEENSLAND

SCHOOL OF INFORMATION TECHNOLOGY AND
ELECTRICAL ENGINEERING

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY.

November, 2003

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work, except as acknowledged in the text, and that the material has not been submitted, either in whole or in part, for a degree at this or any other university.

Thomas Tilley, B.Sc.(Maths & Comp. Sc.), B.Info.Tech.(Hons)

Gold Coast, November 2003.

Abstract

Currently, the bulk of applications of Formal Concept Analysis (FCA) in software engineering have focussed on software maintenance and re-engineering. In this thesis we broaden the approach by applying FCA to a number of early-phase activities within the software engineering life-cycle.

With respect to the requirements engineering phase, a case study is presented comparing two class hierarchies that model aspects of a mass-transit railway ticketing system. The first hierarchy was produced for an existing Object-Z specification of the system while the second was derived using FCA. Contrasting the two hierarchies revealed that they were essentially the same, however, the differences highlighted specification artefacts in the existing hierarchy.

With respect to the design phase, the thesis discusses the use of FCA for the navigation and visualisation of Formal Specifications written in Z. In response to the continued call for formal methods tool support, we implement and explore a prototype specification browser that exploits the abstractions afforded by FCA.

The research hypothesis is an integrated architecture for navigating formal specifications using FCA. This architecture is realised using ZML and ToscanaJ to produce a practical research tool. The thesis also includes the first broad survey of FCA in the domain of software engineering and an FCA-based methodology for surveying academic literature in general.

Acknowledgments

First, my sincere thanks to Peter Eklund who has always been very much the father figure to everyone in KVO — ever supportive and fiercely protective. I am grateful for his guidance, inspiration and the opportunities he has made possible for me. I am also indebted to Roger Duke for his encouragement, and guidance during the final years of my candidature. Roger always knew when to use the reassuring “that sounds typical for a PhD student” line to greatest effect.

I would like to express my gratitude to my colleagues in the KVO Lab: Richard Cole for many doorway discussions on Software Engineering, FCA and life in general; Nataliya Roberts for driving or riding shotgun during many bleary-eyed trips between UQ and the Gold Coast; and Herr Peter Becker for being the ultimate resource and knowing quite a lot about everything.

This work was supported by an Australian Postgraduate Award (APA) Scholarship and I am grateful to the Distributed Systems Technology Centre (DSTC) CRC for additional scholarship support, part-time employment and computer hardware.

I would also like to thank Bastian Wormuth for his translation of Andelfinger’s thesis; Dr Peter Bruza as my DSTC supervisor; the UQ noodle bar who made the weekly ritual of “Noodle Thursday” a delicious reality; and my parents who always encouraged my scientific pursuits despite the potential risks to me, my friends and the house.

Finally, my deepest thanks to God for His sustaining grace; my loving wife Ann who always supported and believed in me while patiently enduring many lonely nights and delayed submission dates; and my children, Emily and Thomas who can now have their turn on the computer after waiting for four and a half years.

Preface

During my Honours year in 1998 I implemented a remote graphical user interface framework for a spatial database management system using Java's Remote Method Invocation (RMI) technology. The initial stages of the project required an exploration of client-server communication techniques ranging from low-level socket-based implementations through to technologies like RMI and CORBA (Common Object Request Broker Architecture). As a result I developed a taste for middleware that continued into 1999 when I spent 5 months working on software for a distributed meeting system at Boeing Australia before commencing my PhD.

As I searched for a topic at the start of my candidature, Peter Eklund proposed the exploration of distributed knowledge management with a focus on middleware — an idea that would combine my previous work in distributed systems with the knowledge management aims of our research group. His suggestion became the focus of my research until later that year when Rudolf Wille visited Australia. During his visit Rudolf gave a lecture on Barwise and Seligman's "logic of distributed systems" and illustrated the principle of information flow using a simple circuit with two lightbulbs. The states of the system could be represented in a crosstable which is then amenable to Formal Concept Analysis (FCA). In true cartoon fashion the lightbulb illustration gave me an idea. I envisaged a tool where a communication protocol for a distributed system could be specified as a (conceivably quite large) crosstable and the logic verified using Barwise and Seligman's approach. A user could then specify how they would like the protocol implemented — RMI, CORBA, D-COM (Microsoft's Distributed Common Object Model), or sockets — and the desired software artefacts (interfaces, stubs, skeletons, data-types,

etc.). The tool would then automatically generate the required code or sub-system to implement the protocol.

My research now had two threads to be developed in parallel until they merged at some point or until one of them petered out. The first thread was distributed knowledge management and the second, the specification and generation of communication protocols. The protocol generation thread required the investigation of any existing protocol specification techniques which led me to the Open Systems Interconnection (OSI) Formal Description Techniques (FDTs) — LOTOS, Estelle, SDL — and in turn to the world of Formal Methods. While the formal methods literature promised more reliable software the ideas and methodologies had not been widely adopted. Part of the problem was perceived as a lack of tool support that could make formal methods "easier" to use. There was a call for robust tools that offered abstraction and were more intuitive, user-friendly and easy to learn than currently available tools or research prototypes. I felt that each of these requirements could be at least partially addressed by FCA and so I abandoned the pursuit of distributed knowledge management. The focus of my thesis became an exploration of the application of FCA to increase the usability of Formal Methods and formal specification in particular.

Although I did not yet have a clear idea of how to create concept lattices from formal specifications there was some existing work that described the relationship between Conceptual Graphs and FCA. I planned to exploit this work by transforming a formal specification into a series of conceptual graphs and then applying FCA. While I ultimately chose a different implementation route a similar method is now being used successfully by Richard Cole to analyse Java class libraries using FCA. Richard had also suggested a thesis topic back in 1999 — using FCA for software visualisation. At that time I was only aware of Snelting's work but was surprised, as my literature survey unfolded, at the number of papers describing FCA applications in software engineering. As a result the focus of my thesis has widened to incorporate early-phase software engineering applications for FCA of which formal specification is now just a part. Here then is the result: "the application of Formal Concept Analysis to requirements engineering and design".

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis describes the application of Formal Concept Analysis (FCA) to a number of early-phase software engineering activities. FCA is a data analysis technique that describes the world in terms of objects and the attributes possessed by those objects. Thomas Boole's understanding that a *concept* can be described by its *extension* and *intension* represents the philosophical starting point for FCA. The mathematical foundations were laid by Birkhoff [19] who demonstrated the correspondence between partial orders and lattices. Birkhoff showed that a lattice can be constructed for every binary relation between a set of objects and a set of attributes with the resulting lattice providing insight into the structure of the original relation.

FCA arose during the early 1980's from Wille's work to restructure lattice theory [225] and it has been successfully applied in a number of areas including psychology [181, 182, 53, 55], psychiatry [53], biological and social sciences [76, 53, 55], civil engineering [121], experimental design [53], information retrieval [86, 35, 41, 161] and software engineering. Within the field of software engineering FCA has already been used for a variety of tasks including the re-engineering of legacy applications, the identification and maintenance of class hierarchies, configuration management and dynamic program analysis. These approaches are discussed in Chapter 2.

A survey of academic papers reporting the application of FCA in software engineering was conducted during the course of this research. The survey found that the bulk of applications of FCA in software engineering have focused on software maintenance and

re-engineering tasks. In this thesis we broaden the approach by applying FCA to a number of early-phase activities within the software engineering life-cycle.

With respect to the requirements engineering phase, a case study is presented comparing two class hierarchies that model aspects of a mass-transit railway system. The first hierarchy was produced for an existing Object-Z specification of the system while the second was derived using FCA. Contrasting the two hierarchies revealed that they were essentially the same, however, the differences highlighted aspects of the first hierarchy that were specification artefacts.

With respect to the design phase, the thesis discusses the use of FCA for the navigation and visualisation of Formal Specifications written in Z. In response to the continued call for formal methods tool support, we implement and explore a prototype specification browser that exploits the abstractions afforded by FCA.

The next section of this chapter introduces the motivation for this work and outlines the overall structure of the thesis. Section 1.4 then provides a brief overview of some related work. Finally, the chapter concludes with the required background for both FCA and the Z specification language in Sections 1.5 and 1.6 respectively.

1.1 Background

There is a deep philosophical understanding behind FCA — the notion that a concept is a unit of thought that is constituted by its extension and intension. A concept's extension contains all the objects that belong to the concept and the intension consists of all the attributes that the objects have in common. "Formal" in the name distinguishes the mathematisation of a concept from the concepts of the human mind.

In FCA the objects, attributes and the relationship between them are normally represented in a crosstable known as a *formal context*. Again the use of the word "formal" indicates that a "formal context" only encodes some small part of what is usually understood as a context [78, 77]. The "formal" in Formal Methods, however, denotes the ordered and deliberate application of mathematically rigorous processes based on formal logic to the act of specification [142].

Formal methods can be broadly defined as tools and notations with formal semantics that support the unambiguous specification of the requirements for a computer system. They provide a means by which the completeness and consistency of a specification can be explored as well as proofs of correctness for implementations of the specification [217, 29]. The application of formal methods can be of benefit to specifiers, implementers, and testers by providing unambiguous communication, verification, validation, and in some cases mechanised code generation [208].

Despite the potential benefits offered by the integration of formal methods into software development there still continues to be limited adoption in industries outside of those writing safety critical software. While there can be significant advantages obtained by integrating formal methods like Z into the production of software artefacts there are also associated costs. This is the basis of a cartoon from the "Formal Methods Humour" web-page [95] which appears in Figure 1.1. The cartoon reflects on the poor adoption of formal methods by industry. Apparently the only thing harder to sell than formal methods is "electric eel on a bun". Hall makes this observation in the classic "Seven Myths of Formal Methods" paper:

> Formal methods are controversial. Their advocates claim that they can revolutionise development. Their detractors think they are impossibly difficult. Meanwhile, for most people, formal methods are so unfamiliar that it is difficult to judge the competing claims [91].

There have been several attempts to dispel the "myths" surrounding formal methods [91, 28], however, a number of them live on and in particular the myths that "formal methods require highly trained mathematics" and that "formal methods are unacceptable to users". As is the case in software engineering there appears to be no silver bullet. With respect to the need for highly trained mathematics Hall states:

> A formal specification is full of mathematical symbols, which render it incomprehensible to anyone unfamiliar with the terminology. Therefore, it is supposed, a formal specification is useless for non-mathematical clients [91].

Hall then goes on to point out that mathematics is only one part of a specification and that there may be other ways of conveying the specification to help clients understand a

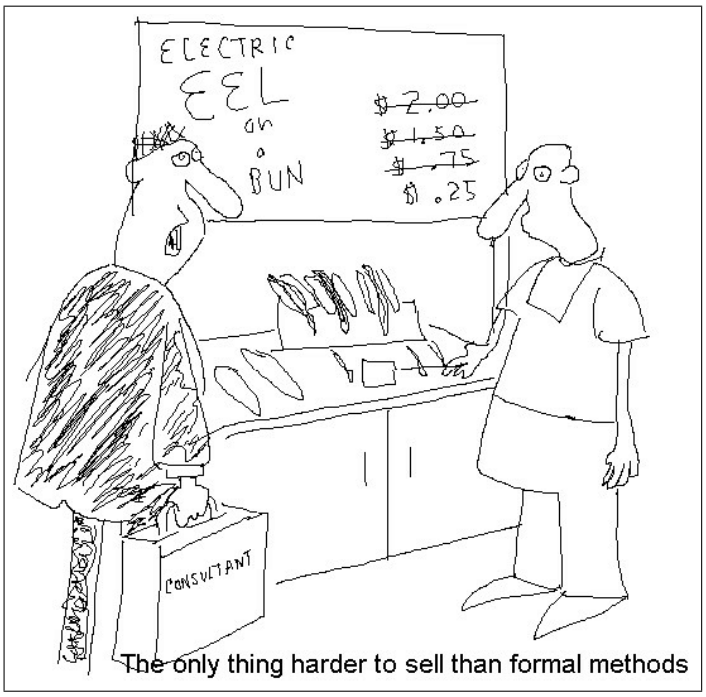


Figure 1.1: Cartoon from the "Formal Methods Humour" web-page [95] that reflects on the adoption of formal methods.

project. Finney [68] also argues that the level of mathematics required to understand Z specifications is higher than that suggested by the proponents of formal methods.

To find reasons why formal methods are not being adopted by industry, Knight, DeJong, Gibble and Nakano [119] conducted a case study where part of the control system for a research reactor was specified using three different formal methods. In a paper describing their results they include the almost humorous statement that "A surprising discovery was that the mathematical notation used in Z was not familiar to the nuclear engineers".

In an attempt to address this problem there have been a number of approaches to provide alternative visual representations of specifications for Z-like languages that have both textual and graphical components within their notation. Typically these approaches use the Unified Modelling Language (UML) [22] to visualise and aid in the understanding of a particular specification aspect. A representative example is the work of Carrington and Kim [117, 116, 115, 114] and these approaches are further discussed in Chapter 4.

Tool support has also been suggested as another path to increase the usability of formal

methods. As the formal methods tools database [71], the `comp.spec.z` FAQ [25], and the WWW Virtual Library [26] demonstrate there is existing tool support for formal methods, however, there continues to be a call for new tools [195, 194]. These calls cite a need not only for tools that have matured from research prototypes into robust, commercial quality software [33, 209], but also for functionality that is not currently supported. There is a need for tools that can present comprehensible specifications and proofs for large systems at different levels of abstraction. German notes that:

> One important problem in current formal methods is that in practice it is difficult to relate formal views of the same system at different levels of abstraction. If we had better practical solutions to this problem, it might be easier to apply formal methods at many stages during the development of a large system [80].

In support of this, Clarke and Wing in their paper on the state of formal methods and future directions list abstraction as a fundamental concept that requires further work:

> Real systems are difficult to specify and verify without abstractions. We need to identify different kinds of abstractions, perhaps tailored for certain kinds of systems or problem domains, and we need to develop ways to justify them formally, perhaps using mechanical help [37].

With reference to formal methods based on Abstract State Machines a similar request is made for "more advanced and industrially satisfactory tool support…for defining, simulating and visualizing…ASMs" [23].

Clarke and Wing go on to list a number of criteria that methods and tools should attempt to address including ease of use, efficiency, and focused analysis. They argue that tools and their output should be as easy to use as compilers. The time taken for analysis should be comparable to that of compilation and individual tools need not be good at analysing all aspects of a system, but they should analyse one aspect well.

The same theme underlies discussion on the Community Z Tools (CZT) mailing list [135] which seeks to promote re-usable tools to increase interoperability and to stop research projects from re-inventing the wheel so they can concentrate on genuine

innovations and improvement [210]. The CZT mailing list was created so that the shortcomings of formal methods tools could be addressed.

The Protocol Engineering Laboratory at the University of Delaware [209] also claims that the need in formal methods is not for new languages — they consider existing languages to be sufficient for ambiguity-free specification — but the need is for more user-friendly and intuitive tools.

1.2 Motivation

Having briefly outlined some of the problems with formal methods the motivation for the thesis can now be unfolded in three parts. First, the majority of FCA applications in software engineering have focused on software maintenance and re-engineering tasks. The thesis seeks to address this by exploring FCA applications to early-phase software engineering activities. While formal methods are applicable to all phases of the software engineering life-cycle [142, 143] the process of formal specification fits within the design phase.

The second motivation for this work is related to formal specification and in particular to existing attempts to increase the usability of Z-like languages by incorporating alternate graphical representations, most notably UML. As an alternative to this approach, the thesis explores the application of FCA for visualising and navigating formal specifications written in Z.

Finally, the third motivation represents a response to the continued call for formal methods tool support. Tool support represents another approach to increase the usability and thereby the adoption of formal methods like Z. In response to this call the thesis describes the implementation of a prototype specification browsing tool. This tool embodies the research hypothesis: an integrated architecture for navigating and visualising formal specifications using FCA.

1.3 Thesis Structure

The overall structure of the thesis reflects the three motivations described above. Chapter 2 presents an overview of FCA support for software engineering. The initial sections of the chapter introduce a framework based on the ISO12207 software engineering standard. The framework is then used to categorise 47 academic papers reporting software engineering applications for FCA. In addition to the ISO12207 categorisation a number of additional classifications are introduced based on the target application language, reported application size, collaboration between authors and citation patterns. FCA is used to present the survey results and an FCA-based methodology for literature surveys in general is discussed. Chapter 2 closes with a brief overview of the techniques described in the survey papers.

Chapter 3 describes an exercise in object-oriented (OO) software modelling where FCA is applied to a formal specification case study using Object-Z. In particular, the informal description from the case study is treated as a set of use-cases from which candidate classes and objects are derived. The resulting class structure is then contrasted with the existing Object-Z design and the two approaches are discussed.

Chapter 4 introduces an approach to navigating and visualising Z specifications using FCA. The approach takes a source specification written in LaTeX and produces a formal context representing the static structure of the specification. A number of line diagrams can then be produced which allow a user to investigate and explore various properties of the specification. The line diagram does not replace, but is intended to be used in conjunction with, the original Z specification. Abstraction through conceptual scaling, nesting, zooming and folding line diagrams allow users to retain context while navigating large specifications and an example based on the *BirthdayBook* specification is presented.

Chapter 5 describes the implementation of a tool developed by the author for interactively exploring Z specifications. The tool implements the ideas introduced in Chapter 4 by exploiting ZML [195], an XML representation of Z, and the open-source, cross-platform FCA tool ToscanaJ [16, 15]. The chapter opens with a discussion about Z mark-up and representation issues including a number of approaches to render Z specifications on the Web and ZML in particular. An overview of a number of FCA

tools (including ToscanaJ) is then presented and the remainder of the chapter describes the implementation of the prototype FCA-based specification browsing tool. Finally, Chapter 6 concludes the thesis and discusses future directions for this work.

1.4 Formal Methods and FCA

This section provides a brief overview of some existing work that uses both formal methods and FCA. For example, Fischer has described an approach for browsing and navigating a software component library by combining formal methods and FCA [70]. Components in the library are associated with formal specifications that capture their behaviour in the form of pre-conditions and post-conditions. Automated theorem provers are used to deduce valid relations between pairs of components for a number of different relation types including refinement and matching. A formal concept lattice is then computed that is used as a structure for navigating the library. Fischer's approach builds on the earlier work of Lindig [128] and both approaches are summarised in Section 2.4.3.

Mili, Boudrigua, and Elloumi have also produced a semi-lattice of specifications where a set of specifications are ordered using the "stronger than" relation [138]. A specification S_1 being *stronger than* another specification S_2 has a number of interpretations including S_1 is more refined, carries more input-output information, or is more specific than S_2. With this ordering the *least upper bound* between two specifications captures the total input-output information carried by each of them and the *greatest lower bound* captures the common input-output information. This approach has applications for combining multiple specifications during specification generation as well as completeness checking during validation. The resulting lattice structure has also been used to organise a software library for component reuse, however, it differs from a concept lattice because it does not admit a universal upper bound [1].

More recently, Ammons, Mandelin, Bodik, and Larus [5] have also incorporated FCA and Formal Methods in their work to debug temporal specifications. While very small

[1] While a concept lattice must be a complete lattice (see Section 1.5.2) two sub-structures derived from the concept lattice have also found applications: the *Iceberg lattice* [191] and the *Galois sub-hierarchy* [87].

specifications can be debugged by inspection, larger specifications are verified using tools that check the specification against a number of programs. There may be hundreds or thousands of execution traces from these checks and these are used as the formal objects in their analysis. Each of the execution traces must be classified by an expert who decides if they are correct or erroneous. By considering transitions within the finite automata that represent the specifications as the formal attributes, a concept lattice can be produced that clusters similar traces together. An expert can then classify clusters of traces rather than classifying them all individually.

1.5 Formal Concept Analysis

FCA is a way of describing the world in terms of objects and the attributes possessed by those objects. This section introduces the FCA notation and conventions used throughout the thesis. The introduction is based on Ganter and Wille's FCA textbook [78] and also seeks to be consistent with the notation used by Davey and Priestly [44].

As briefly mentioned at the start of this chapter, FCA is based on the philosophical understanding that a concept can be described by its *extension* — that is all the objects that belong to the concept and its *intension* which are all the attributes that the objects have in common. For example, the extension of the concept "mammal" includes the objects "humans" and "mice" while the intension includes the attributes "warm blooded" and "has hair". The relationships between the set of objects and the set of attributes is represented by a *formal context*.

1.5.1 Formal Context

A *formal context* $\mathbb{K} := (G, M, I)$ is a triple where G is a set of formal *objects* (from the German "Gegenstände"), M is a set of *attributes* (from the German "Merkmalle"), and I is an *incidence* relation between the objects and the attributes. $I \subseteq G \times M$ is a binary relation where $(g, m) \in I$ is read "object g has attribute m" and is often written as gIm

	small	medium	large	near	far	moon(s)	no_moon
Mercury	×			×			×
Venus	×			×			×
Earth	×			×		×	
Mars	×			×		×	
Jupiter			×		×	×	
Saturn			×		×	×	
Uranus		×			×	×	
Neptune		×			×	×	
Pluto	×				×	×	

Table 1.1: Formal context containing information about the planets. Here G is the set of planet names, $M = \{small, medium, large, near, far, moon(s), no\_moon\}$ and the incidence relation I is represented by the presence of an '×' where gIm.

for convenience. A formal context can be represented as a crosstable [2] where the rows represent G, the columns represent M and the incidence relation I is represented by a series of crosses as shown in Table 1.1

In this example taken from Davey and Priestly [44] the object set G contains the nine planets of the solar system while the attribute set $M = \{small, medium, large, near, far, moon(s), no\_moon\}$. A '×' at the intersection of an object row and attribute column indicates that the object possesses that attribute. For example, the planet Earth has a moon so $(Earth, moon(s)) \in I$. While the inclusion of both *moon(s)* and *no_moon* attributes appears to be redundant in this example, the context has been created so that subsets of the attributes can be used as *conceptual scales*. Conceptual scaling is introduced in Section 1.5.4.

For a subset of the objects, $A \subseteq G$ we can define the set of common attributes $A^{\uparrow}$ as:

$$A^{\uparrow} := \{m \in M \mid (g, m) \in I, \forall\, g \in A\}$$

and dually, for a subset of attributes, $B \subseteq M$ we can define the set $B^{\downarrow}$ of objects having all the attributes from B as:

[2] The terms *context*, *crosstable* and *formal context* are used interchangeably throughout the remainder of the thesis.

$$B^{\downarrow} := \{g \in G \mid (g, m) \in I, \forall\, m \in B\}$$

For convenience $A^{\uparrow}$ and $B^{\downarrow}$ are often write as A' and B'.

A concept can be found by taking a subset of the objects, finding the set of all attributes that the objects possess and then determining the set of all objects with those attributes. For example, starting with the planet Mars, the set of attributes B is $\{small, near, moon(s)\}$. The set of all planets with these attributes A is $\{Earth, Mars\}$ and together these two sets represents the concept $(\{Earth, Mars\}, \{small, near, moon(s)\})$. (A, B) is a *formal concept* of (G, M, I) iff:

$$A \subseteq G,\ B \subseteq M,\ A' = B,\ \textit{and}\ B' = A.$$

The set A is called the *extent* and B the *intent* of the formal concept (A, B).

Given the above definition, then A' represents the intent of the concept (A, B) which can be written (A, A'). Furthermore, A'' is the smallest extent containing A. Consequently, $A \subseteq G$ is an extent iff $A'' = A$. Similarly, $B \subseteq M$ is an intent iff $B'' = B$.

Within the formal context a formal concept represents a maximal rectangle and the set of all formal concepts of (G, M, I) is $\mathfrak{B}(G, M, I)$ (from the German "Begriffe") or $\mathfrak{B}(\mathbb{K})$. For the example shown in Table 1.1 $\mathbb{K}$ contains exactly 12 formal concepts where $\mathfrak{B}(\mathbb{K})$ is the set:

$(\{Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto\}, \{\varnothing\})$,

$(\{Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto\}, \{moon(s)\})$,

$(\{Jupiter, Saturn, Uranus, Neptune, Pluto\}, \{far, moon(s)\})$,

$(\{Jupiter, Saturn\}, \{large, far, moon(s)\})$,

$(\{Uranus, Neptune\}, \{medium, far, moon(s)\})$,

$(\{Mercury, Venus, Earth, Mars, Pluto\}, \{small\})$,

$(\{Earth, Mars, Pluto\}, \{small, moon(s)\})$,

$(\{Pluto\}, \{small, far, moon(s)\})$,

$(\{Mercury, Venus, Earth, Mars\}, \{small, near\})$,

$(\{Mercury, Venus\}, \{small, near, no\_moon\})$,

$(\{Earth, Mars\}, \{small, near, moon(s)\})$,
$(\{\varnothing\}, \{small, medium, large, near, far, moon(s), no\_moon\})\}$.

The concepts of a context are ordered by the *subconcept-superconcept relation* which is defined by

$$(A_1, B_1) \leq (A_2, B_2) \Longleftrightarrow A_1 \subseteq A_2 \wedge B_2 \subseteq B_1$$

where (A_1, B_1) is called a *subconcept* of (A_2, B_2) and conversely, (A_2, B_2) is a *superconcept* of (A_1, B_1). Subconcepts are said to be *smaller* or *less general* than their superconcepts and the superconcepts *larger* or *more general* than their subconcepts.

1.5.2 Formal Concept Lattice

For the set of concepts $\mathfrak{B}(\mathbb{K})$ there is always a greatest subconcept and a smallest superconcept. $\mathfrak{B}(\mathbb{K})$ together with the order relation '$\leq$' forms a complete lattice $\underline{\mathfrak{B}}(\mathbb{K})$. A *complete lattice* is a partially ordered set in which every subset has a greatest lower bound and a least upper bound. $\underline{\mathfrak{B}}(\mathbb{K})$ is called the *concept lattice* of $\mathbb{K}$. Concept lattices are the basic conceptual structure in FCA and are also sometimes referred to as a *Galois lattice* because $\uparrow$ and $\downarrow$ form a Galois connection between G and M [44].

The basic theorem on concept lattices states that the concept lattice $\underline{\mathfrak{B}}(G, M, I)$ is a complete lattice in which the *infimum* is given by:

$$\bigwedge_{t \in T}(A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t\right)''\right)$$

and the *supremum* by:

$$\bigvee_{t \in T}(A_t, B_t) = \left(\left(\bigcup_{t \in T} A_t\right)'', \bigcap_{t \in T} B_t\right)$$

A complete lattice $\underline{L}$ is isomorphic to $\underline{\mathfrak{B}}(G, M, I)$ iff there are mappings $\tilde{\gamma} : G \to L$ and $\tilde{\mu} : M \to L$ such that $\tilde{\gamma}(G)$ is supremum-dense in $\underline{L}$, $\mu(\tilde{M})$ is infimum dense in $\underline{L}$ and gIm is equivalent to $\gamma g \leq \mu m, \forall\, g \in G, \forall\, m \in M$. In particular $\underline{L} \cong \underline{\mathfrak{B}}(L, L, \leq)$.

In the worst case a concept lattice can consist of 2^n concepts where the value of $n = (min(|G|, |M|)$. The complexity is therefore exponential. However, Godin and Mili [86], and Lindig [128] offer experimental evidence that in practice the behaviour is typically polynomial. Incremental lattice construction algorithms have also been demonstrated [85, 83].

1.5.3 Line Diagram

A formal concept lattice $\underline{\mathfrak{B}}(\mathbb{K})$ can be drawn as a specialised Hasse diagram [44] which is also commonly known as a *labelled line diagram* [3]. Each concept is represented by a node in the line diagram and the line segments represent subconcept-superconcept relations. The line diagram corresponding to the formal context in Table 1.1 appears as Figure 1.2.

Each node in the line diagram having exactly one segment down must also have at least one object name. Similarly, each node with a single line segment up must have at least one attribute name. These are known as *irreducible* objects and attributes. For example, the two nodes below the top of the diagram in Figure 1.2 each have one line segment up and are labelled with the attribute names “small” and “moon(s)”.

Rather than labelling each concept with its extent and intent a *reduced labelling* scheme is typically used so that each object and each attribute appear only once on the diagram. Reduced labelling is used in Figure 1.2. In this scheme the label for an object g is drawn below the *object concept* $\gamma g := (\{g\}'', \{g\}')$ while the label for an attribute m is drawn above the *attribute concept* $\mu m := (\{m\}', \{m\}'')$.

The extent of a concept represents all the object labels that can be reached along a descending path from the concept. The set of concepts along the downward path is known as the *down-set* or *order ideal*. Conversely, the intent of a concept can be recovered by collecting all of the attribute labels along upward paths from the concept. The set of concepts along the upward paths are known as the *up-set* or *order filter*.

[3] Although a line diagram is just a representation of a formal concept lattice the terms *line diagram* and *concept lattice* are used interchangeably throughout the remainder of the thesis to denote the labelled line diagram of a formal concept lattice.

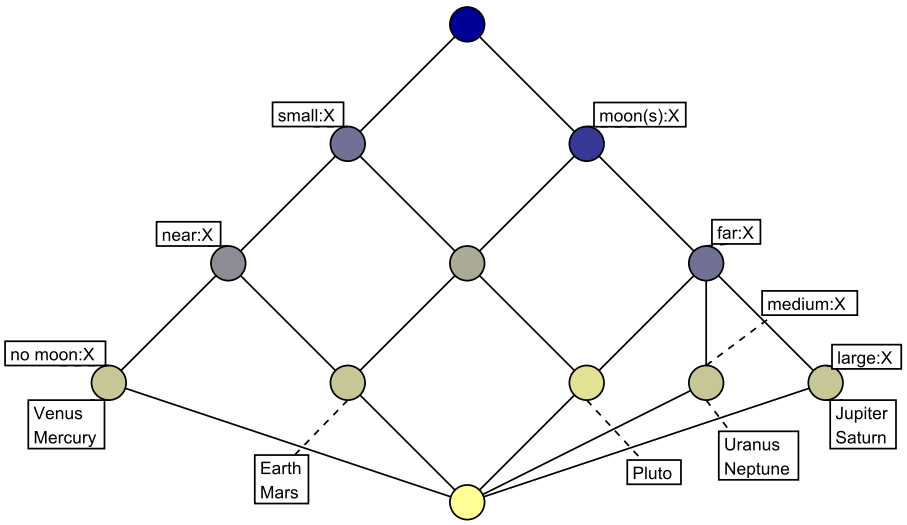


Figure 1.2: The formal concept lattice corresponding to the planet context in Table 1.1.

For example, the extent of the concept with the attribute label "small" in Figure 1.2 can be found by following the downward paths to recover the planets $\{Venus, Mercury, Earth, Mars, Pluto\}$. That is, those planets which are small. The attributes or intent of the planet Pluto can be found by following the upward paths from the concept with the object label "Pluto" to recover the set $\{small, moon(s), far\}$.

This is an interesting feature of FCA. Unlike some other data analysis techniques, the original data from the context can be recovered directly from the line diagram. *Implications* between attributes can also be read from the line diagram. For example, there are at least two planets that satisfy the following attribute implications:

$\{no\_moon\} \Rightarrow \{small, near\}$,

$\{far\} \Rightarrow \{moon(s)\}$,

$\{near\} \Rightarrow \{small\}$,

$\{large\} \Rightarrow \{far, moon(s)\}$, and

$\{medium\} \Rightarrow \{far, moon(s)\}$.

More formally, an implication between attributes in M is a pair (A, B) of subsets $A, B \subseteq M$ denoted $A \Rightarrow B$. The implication is read as "A implies B" where the set A is

	diameter (km)
Mercury	4880
Venus	12,100
Earth	12,756
Mars	6,786.8
Jupiter	143,200
Saturn	120,000
Uranus	51,800
Neptune	49,528
Pluto	≈2,330

Table 1.2: A many-valued context showing the equatorial diameter (in kilometres) for the nine planets.

	small	medium	large
< 25,000 km	×		
≥ 25,000 km and < 100,000 km		×	
≥ 100,000 km			×

Table 1.3: A conceptual scale which maps planet diameters to the sizes *small, medium* and *large*.

the *premise* of the implication $A \Rightarrow B$ and B is the *conclusion.* An implication holds in a formal context $\mathbb{K}$ iff every object that has all the attributes in A also has all attributes in B, $B \subseteq A''$, which is equivalent to $A' \subseteq B'$.

1.5.4 Conceptual Scaling

In addition to one-valued data FCA can also be used to analyse many-valued data sets like the table shown in Table 1.2. A *many-valued context* is a 4-tuple (G, M, W, I) where G is a set of objects, M is a set of many-valued attributes, W a set of attribute values and $I \subseteq G \times M \times W$ where $(g, m, v) \in I$ and $(g, m, w) \in I \Rightarrow v = w$.

A many-valued context is first transformed into a one-valued context by *conceptual scaling*. A *conceptual scale* for a many valued attribute m is a one-valued context which has the attribute values of m among its objects. Table 1.3 presents such a scale which maps planet diameters to the sizes *small, medium,* and *large*.

Let (G, M, W, I) be a many-valued context and for each $m \in M$ let $\mathbb{S}_m := (G_m, M_m, I_m)$ be a scale for m. The *derived context* of (G, M, W, I) with respect to *plain scaling* with the scales $(S_m \mid m \in M)$ is then (G, N, J) where:

$$N := \bigtimes_{m \in M} \{m\} \times M_m$$

and

$$(g, (m, n)) \in J :\Longleftrightarrow \exists_{w \in W} (g, m, w) \in I \text{ and } (w, n) \in I_m.$$

The derived context resulting from the application of the conceptual scale in Table 1.3 to the many-valued context Table 1.2 appears as Table 1.4. The corresponding line diagram is shown in Figure 1.3. Conceptual scales represent a very powerful tool that can be used to store views that partition the data being analysed. Within a conceptual data system multiple views can be stored and applied to effectively query the data.

A number of elementary scale types are available including *nominal, ordinal, inter-ordinal* and *bi-ordinal* scales. Nominal scales are used to scale attributes with mutually exclusive values such as $\{moon(s), no\_moon\}$. Ordinal scales are used where the values of a many-valued attribute are ordered and each of the values implies the "weaker" ones. For example, ordinal scaling could be used with the attribute values $\{strong, stronger, strongest\}$ and the result is a chain of extents which can be interpreted as a hierarchy.

Inter-ordinal scales are used to scale bipolar values which are often used to represent questionnaire answers. For example, the values $\{\leq 1, \leq 2 \leq 3, \geq 1, \geq 2, \geq 3\}$ result in extents which are the intervals of values. Bipolar attributes can also be scaled using bi-ordinal scales where there is a partitioning within a hierarchy. For example, within a marking scheme with values $\{very\ poor, poor, acceptable, good, very\ good\}$ where the value "very good" implies "good" but not "poor".

1.5.5 Nested Line Diagrams

Like conceptual scaling, *nested line diagrams* represent a powerful tool for abstraction and analysis. A nested line diagram is produced by first partitioning the attribute set M of

	small	medium	large
Mercury	×		
Venus	×		
Earth	×		
Mars	×		
Jupiter			×
Saturn			×
Uranus		×	
Neptune		×	
Pluto	×		

Table 1.4: The formal context that results from applying the conceptual scale in Table 1.3 to the many-valued context in Table 1.2.

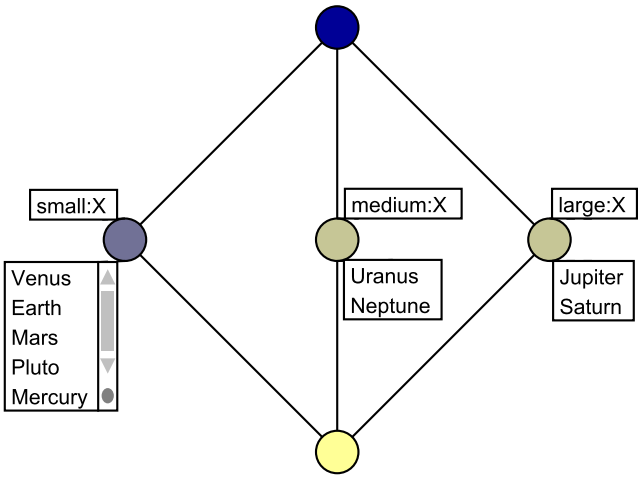


Figure 1.3: The formal concept lattice corresponding to Table 1.4 (a subcontext of Table 1.1 for the size attributes *small, medium* and *large*).

a context into the sets M_1 and M_2. The two concept lattices $\underline{\mathfrak{B}}(G, M_1, I \cap G \times M_1)$ and $\underline{\mathfrak{B}}(G, M_2, I \cap G \times M_2)$ can then be computed. The nested line diagram is the direct product of these two lattices where the elements of $\underline{\mathfrak{B}}(G, M, I)$ are shown as solid circles. For two contexts $\mathbb{K}_1$ and $\mathbb{K}_2$ the *direct product* is given by

$$\mathbb{K}_1 \times \mathbb{K}_2 := (G_1 \times G_2, M_1 \times M_2, \nabla)$$

with $(g_1, g_2)\nabla(m_1, m_2) :\Longleftrightarrow g_1 I_1 m_1$ or $g_2 I_2 m_2$.

As an example, the context from Table 1.1 can be partitioned into the two sets

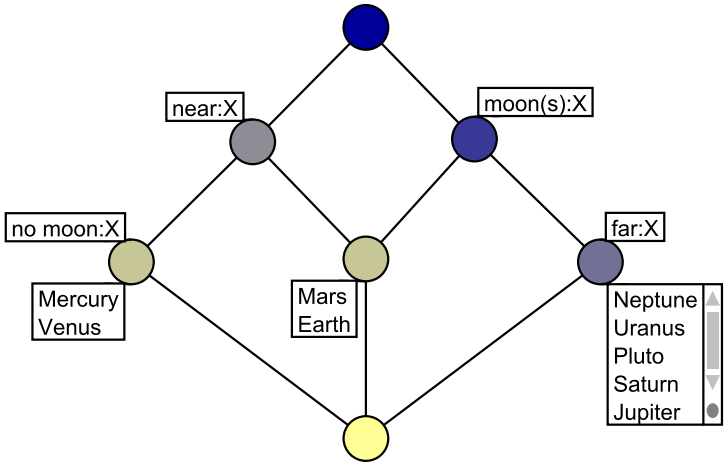


Figure 1.4: The formal concept lattice for a sub-context of Table 1.1 for the attribute set $M = \{near, far, moon(s), no\_moon\}$.

$\{small, medium, large\}$ and $\{near, far, moon(s), no\_moon\}$. The line diagrams corresponding to the two sub-contexts are shown in Figures 1.3 and 1.4 respectively. The resulting nested line diagram is shown in Figure 1.5.

Section 5.2.3 of the thesis introduces the FCA tools ANACONDA and TOSCANA which implement conceptual scaling and nested line diagrams using formal contexts that are stored in relational databases. The normal workflow for these tools is to partition a single large context into sub-contexts which are then used as scales. Multiple scales can be composed together to effectively query and explore the data which can be viewed using nested line diagrams.

1.5.6 Order Embedding

The automated layout of large or complex line diagrams in FCA often produces poor results [41]. One effective approach for drawing lattices up to moderate size is to use an *order embedding* where a lattice with a known layout is used to draw a second lattice. In Figure 1.6 the lattice on the left is embedded in the lattice closest to center to produce the lattice shown right.

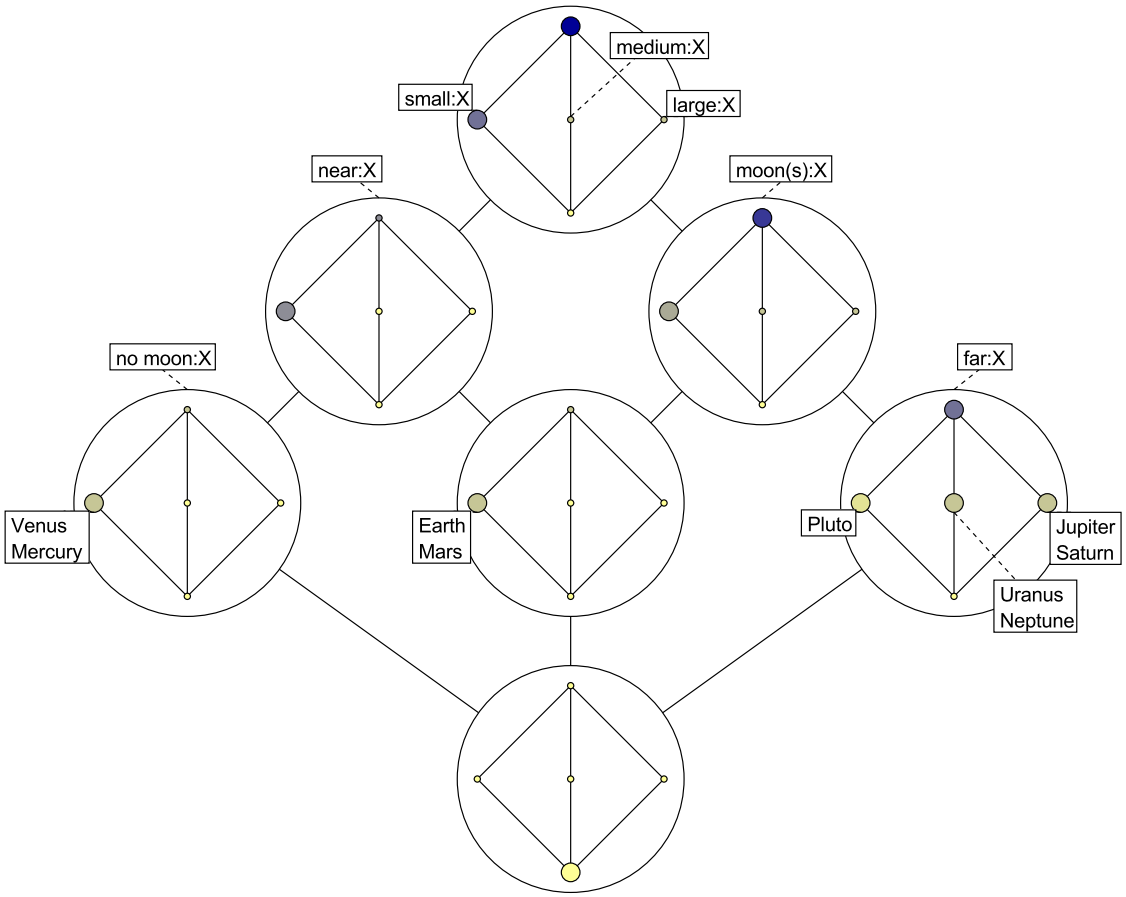


Figure 1.5: Nested line diagram showing the scale from Figure 1.3 nested inside Figure 1.4.

A map $\varphi : M \longrightarrow N$ between two ordered sets $(M, \leq)$ and $(N, \leq)$ is called an order embedding if the map is *order preserving* such that

$$x \leq y \Rightarrow \varphi x \leq \varphi y$$

for all $x, y \in M$ and furthermore, if φ also fulfils the converse implication

$$x \leq y \Leftarrow \varphi x \leq \varphi y.$$

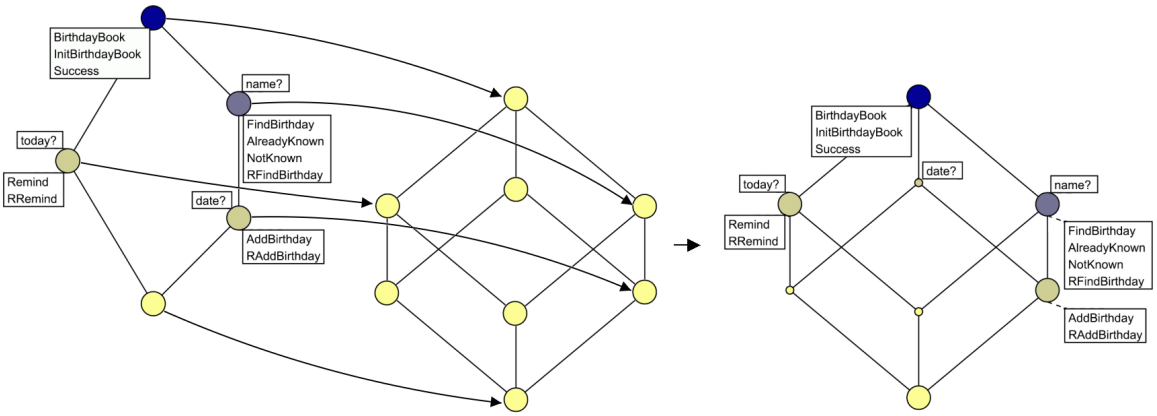


Figure 1.6: The lattice shown right is the result of an *order embedding*. The initial lattice shown left is sometimes called a "reduced line diagram".

1.5.7 Attribute Exploration

A concept lattice can be very large or potentially even infinite. However, despite being unable to determine the entire lattice, parts of it may be known and the context may have a comparatively small set of objects or attributes. The context of such an unknown lattice is called a *conceptual universe* and while it may not be practical or possible to completely determine the context there is an approach for determining sets of "typical" objects or attributes.

Let $\mathbb{U} := (G_{\mathbb{U}}, M, I_{\mathbb{U}})$ be a conceptual universe with a fixed set of attributes M. A *typical* or *representative* set of objects $G \subseteq G_{\mathbb{U}}$ if the concept lattices $(\mathfrak{B}(G_{\mathbb{U}}, M, I_{\mathbb{U}}), \leq)$ and $(\mathfrak{B}(G, M, I_{\mathbb{U}} \cap (G \times M)), \leq)$ are isomorphic such that $(A, B) \mapsto (\tilde{A}, B)$. That is, the intents of related concepts are the same.

Provided that a domain expert has sufficient knowledge, the process of *attribute exploration* [32, 75, 190] can be used to determine a typical set of objects. The process suggests implications to the domain expert who either accepts the implication or rejects it and updates the context with a counter-example. Prior to starting the exploration any known, pre-existing implications — so called *background implications* — can be identified. Once the exploration process is complete the result is a set of valid implications, known as the *Duquenne-Guigues-base*, and a context containing a typical set of objects.

By transposing G and M in the context it is also possible to conduct *object exploration*. The question regarding implications then changes from "Do all the objects having all the attributes of the premise also have all the attributes of the conclusion?" to "Do all the attributes which belong to the objects of the premise also belong to all the objects of the conclusion?". Where both the set of objects and the set of attributes are extensible a domain expert can switch between attribute exploration and object exploration to alternatively fill out the set of objects and attributes respectively.

Attribute exploration can also make use of a three-valued logic where the third value acts as a placeholder for unknown values, e.g. "true", "false", "uncertain". Using this type of logic only part of the conclusion needs to be disproved and any unknown parts can be left open — perhaps to be determined as a later refinement step once the initial requirements have been implemented. A number of tools support interactive attribute exploration and an overview of tools for FCA is presented in Section 5.2. The next section of this chapter introduces the notation used in the Z specification language.

1.6 Formal Specification in Z

Z [47, 92, 183, 184, 230, 232] is a state based formal method that exploits Zermelo-Fränkel set theory and first order predicate logic. The Z specification language was developed by the Programming Research Group [150] at the Oxford University Computing Laboratory in the early 1980's — around the same time that FCA was first introduced. In 2002 Z was standardised as ISO/IEC 13568:2002 [104], however, the work described in this thesis has used the form of Z as introduced by Spivey [183] which is basically a subset of standard Z.

Specifications in Z are composed of named *schema* boxes that describe operations by their input and output behaviour. Schemas are divided into an upper region called the *declaration* or *signature* part and a lower region called the *predicate*, *property*, or more correctly *formula* part. Variables and their respective types are declared in the upper region while the lower region contains predicates describing pre-conditions and post-conditions for the current operation. Models are constructed by specifying and composing a series of schemas and the schemas can be refined to reflect the desired level of system abstraction.

As a result of the mathematical nature of the notation (and the graphical nature of schema boxes) most Z tools are comprised of at least a formatting package for LaTeX [124] and a type-checker.

Schemas are used to represent both the static and the dynamic aspects of a system. The static aspects include the possible states of the system and any invariants that must hold on state transitions. Dynamic aspects include the actual state changes, the possible operations and the relationship between inputs and outputs.

Spivey's *BirthdayBook* specification [184] represents the equivalent of a "Hello World" program for Z. It introduces the notation and ideas behind the Z specification language. Only parts of the *BirthdayBook* specification are presented here and this introduction also seeks to be consistent with the notation described by Diller [47]. The *BirthdayBook* specification is also used as an illustrative example in Chapter 4 and the complete specification appears in Appendix A.

The *BirthdayBook* specification provides an introduction to the Z notation by describing a simple reminder system for recording people's birthdays using a set of names and a set of dates. Spivey's specification of the system also includes schemas to add new name/date pairs into the system as well as operations to check for current birthdays.

The basic data-types in Z are modelled as sets — in this case a set of names and a set of dates:

$$[NAME, DATE]$$

A schema can then be declared to describe the state space of the system as a set of names that are recognised by the system and a partial mapping from the names to the corresponding birthdates:

BirthdayBook

$$known : \mathbb{P}\, NAME$$
$$birthday : NAME \pfun DATE$$

$$known = \mathrm{dom}\, birthday$$

Figure 1.7: A black box specification of the *AlreadyKnown* operation.

The initial state of the system also needs to be described:

InitBirthdayBook

$$BirthdayBook$$

$$known = \varnothing$$

Having declared the basic types, described the state space of the system and the initial state, operations can now be defined. Schemas in Z are described by their input and output behaviour. Diller uses the illustration of a black box specification where the implementation is hidden inside the box and the specifier can describe conditions for the inputs and outputs. The implementation of the box is left as an exercise for the programmer. Z uses procedural abstraction to focus on what has to be done but not how it is done.

For example, to indicate if a name has already been used by the system an *AlreadyKnown* operation can be implemented. A black box representation of the *AlreadyKnown* operation appears in Figure 1.7. In Z inputs are denoted with '?' and outputs with '!'. The *AlreadyKnown* function takes a name as input and produces an output to indicate the success or failure of the operation.

Enumerated data-types can also be declared and a *REPORT* data-type could be used to list the possible result values. This is an example of a free-type definition and it defines *REPORT* as a set containing exactly three values:

$$REPORT ::= ok \mid already\_known \mid not\_known$$

The *AlreadyKnown* schema corresponding to the black box specification in Figure 1.7 could now be written in Z as:

AlreadyKnown

$\Xi BirthdayBook$
$name? : NAME$
$result! : REPORT$

$name? \in known$
$result! = already\_known$

Schema inclusion allows an existing schema to be used inside another schema. For example, the $\Xi BirthdayBook$ declaration in the *AlreadyKnown* schema above includes the *BirthdayBook* state schema within *AlreadyKnown* in both primed and unprimed versions. Primes are used to denote the "after" or "post" states. This schema does not change the state of the system so the pre and post states are the same, that is $known' = known$ and $birthday' = birthday$. The symbols Δ and Ξ are short-hand conventions used to identify those schemas that change the state of the system (Δ) and those that do not (Ξ)[4].

As an example that changes the state of the system the *AddBirthday* schema takes a name and a date as inputs and adds them into the *BirthdayBook*:

AddBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

$name? \notin known$
$birthday' = birthday \cup name? \mapsto date?$

1.6.1 Schema Composition

In addition to simple inclusion, the schema calculus can be used to combine schemas together to model more complex behaviour. For example, if a new state schema is introduced to model the success or failure of operations then a more robust version of the *AddBirthday* operation can be specified. An error reporting schema named *Success* could be defined as:

[4]This is the standard usage of Δ and Ξ within a Z specification however the notation allows a user to re-define their own meanings for these symbols.

Success

$$result! : REPORT$$

$$result! = ok$$

Using schema *composition* or *linking* this new state schema could then be combined with the *AddBirthday* and *AlreadyKnown* schemas to produce *RAddBirthday*:

$$RAddBirthday \mathrel{\widehat=} (AddBirthday \land Success) \lor AlreadyKnown$$

In addition to schema conjunction '$\land$' and disjunction '$\lor$' other forms of schema calculus include negation '$\neg$', implication '$\Rightarrow$', bi-implication $\Leftrightarrow$, piping '$\gg$' and sequential composition '$\fatsemi$'. The definition sign '$\mathrel{\widehat=}$' allows one schema to be defined in terms of others and this is sometimes known as a *horizontal schema*. The resulting schema represents the merge of the linked schemas which can be written out in full as:

RAddBirthday

$$\Delta BirthdayBook$$
$$name? : NAME$$
$$date? : DATE$$
$$result! : REPORT$$

$$(name? \notin known \land$$
$$\quad birthday' = birthday \cup \{name? \mapsto date?\} \land$$
$$\quad result! = ok) \lor$$
$$(name? \in known \land$$
$$\quad birthday' = birthday \land$$
$$\quad result! = already\_known)$$

This style of formal specification allows certain properties of a system to be proved and schemas can also be specified at different levels of abstraction. The behaviour of a system can also be explored without actually implementing the system itself.

Tool support for Z typically includes pretty printers, syntax checkers and type-checkers. It is also possible to animate Z specifications — providing a partial implementation of the specification in software — to aid the specification writer's understanding of the system.

A set-based notation like Z lends itself to animation using set-based functional languages like Miranda [139] or Haskell [152]. Diller also presents a Prolog [160] based animation example [47]. An overview of Z representation issues and tool support are discussed in Section 5.1 of Chapter 5.

1.6.2 Object-Z

Object-Z [52, 174] is an extension of the Z specification language that provides object-oriented structuring mechanisms. As with Z itself there are a number of object-oriented Z variants including Z^{++} [125] and OOZE [3], however, Object-Z has been the most successful.

A class in Object-Z is specified as a box that contains the features and operations of the class and may also include generic parameters. A simple class implementing a generic FIFO (First In First Out) queue from an example by Mahony and Dong [134] is presented in Figure 1.8. A similar example is also used by Smith [174].

The generic parameter *X* in the name of the class box represents the as-yet undefined elements that will be placed on the queue. The general structure of a class in Object-Z consists of the following parts in order:

Visibility List The first item in the class is a visibility list which defines the interface for instances of the class. In the *Queue* class there is no visibility list so all the features are implicitly visible.

Constants Any class constants are declared next. While there are no constants declared in this specification a bounded queue could contain, for example, a *bufferSize* or *length* constant to define the maximum size of the queue.

State Schema A class represents a template for objects that are instantiations of the class. The state of an object is an instance of the *state schema* which is represented by an unnamed schema box. As in Z, state variables are declared in the top part of the schema. Any predicates in the formula part must be true and are referred to as the *class invariant*. In the *Queue* class the value of the head of the queue *h* is only specified for a non-empty sequence of items. The *attributes* of a class consist of the state variables along with any constants that have been declared. State variables

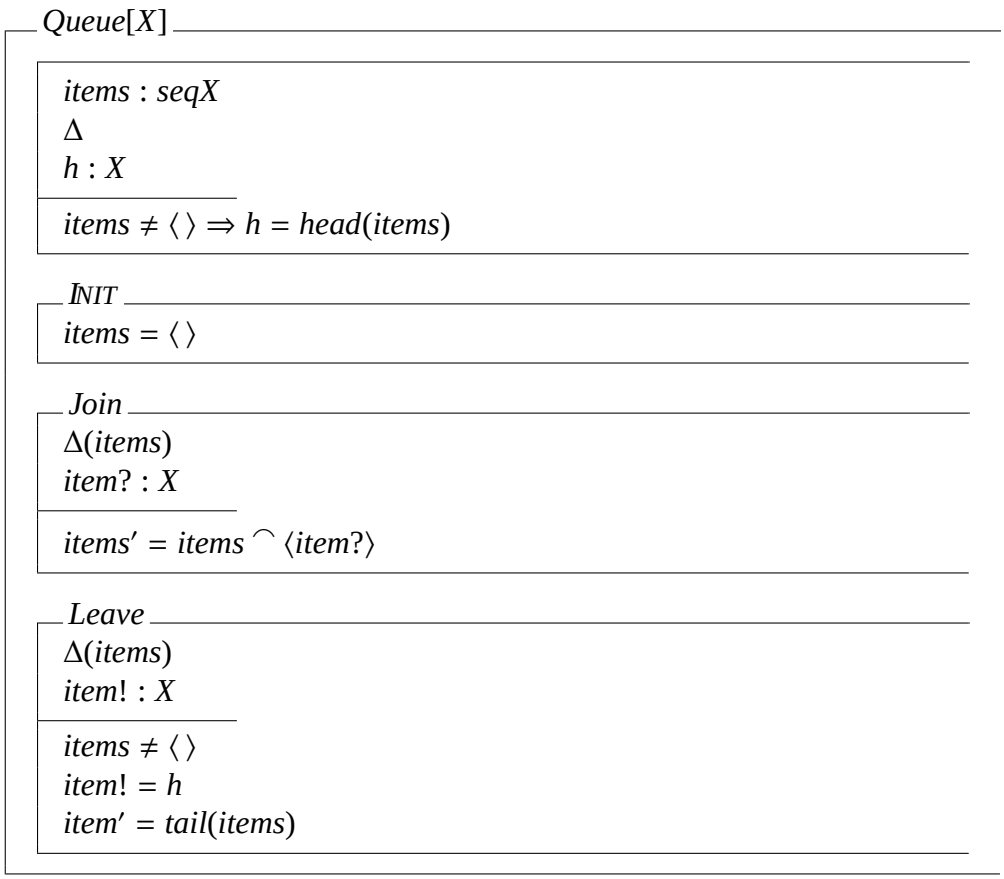


Figure 1.8: Object-Z class for a generic FIFO queue.

declared above the Δ separator in the state schema are called *primary variables* while those below are known as *secondary variables*. Secondary variables are subject to change whenever an operation is performed. In this example the value of the variable representing the head of the queue, *h* may change every time an item is removed from the queue.

Initial Schema The *Initial Schema* is always named INIT and represents the initial state of an object. The class invariant and the initial schema are conjoined to define the *initial condition* — in this case an initially empty sequence of items.

Operation Schemas There are two operation schemas defining the available operations on a *Queue* — a *Join* operation which adds an item to the back of the queue and a

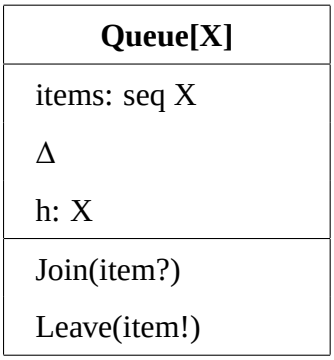


Figure 1.9: Object-Z class diagram showing features of the *Queue* class.

Leave operation which removes the item at the head of the queue. Operation schemas contain a Δ-list showing the primary attributes whose values may be modified by the operation.

While the discussion of UML-like graphical representations and Z is delayed until Chapter 4, Object-Z already includes a number of UML-like diagrams for illustrating aspects of object-oriented specifications. An Object-Z class diagram summarising the features of the *Queue* class is presented in Figure 1.9. Chapter 3 also presents a case study based on an Object-Z specification of a mass transit railway system.

Having already introduced the required background in FCA and Z and discussed the motivation for the thesis, Chapter 2 presents a survey of FCA support for software engineering.

Chapter 2

A Survey of FCA Support for Software Engineering

This chapter presents a literature survey of 47 academic papers reporting software engineering applications for FCA. An early version of the work presented here has been published in a paper co-authored with Richard Cole, Peter Becker and Peter Eklund [200].

The initial sections of the chapter introduce a framework based on the ISO12207 software engineering standard that is subsequently used to categorise the survey papers. Additionally, a number of alternative classifications based on the target application language and the reported application size are introduced. The results of the survey are also presented using FCA and the approaches described in the papers are briefly discussed.

While the first half of the chapter presents a background survey that supports the first of the motivations outlined in Section 1.2, the second half of the chapter represents a new research contribution. An FCA-based analysis of the author collaboration and citation patterns within the set of survey papers is discussed in Section 2.6. This approach is then extended and the use of FCA as a tool for literature surveys in general is presented.

2.1 Understanding Software Engineering

To understand how software engineering can be supported by FCA some understanding of what software engineering is, or at least the processes involved, is necessary. This section of the paper sets out a framework that will be used to classify papers from a software

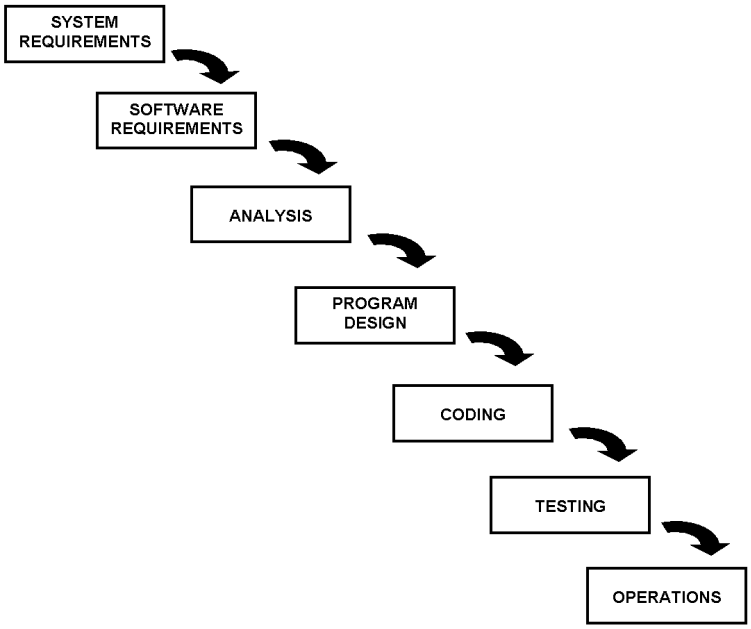


Figure 2.1: The classic waterfall life-cycle model.

engineering development perspective.

The development of software has traditionally been described by life-cycle models. These models grew out of a need to more effectively understand and manage the software engineering process which has been characterised by failed, late, and bug-laden projects. Royce [163] proposed the classic "waterfall" model which consists of seven *steps* or *phases* that proceed in a linear fashion: System Requirements, Software Requirements, Analysis, Program Design, Coding, Testing, and Operations. See Figure 2.1.

The waterfall model focuses heavily on the documentation produced during each implementation phase and there may be some iteration between successive steps. Royce realised that sometimes iterations happen across non-consecutive steps which is undesirable. To address this he proposed some extensions to alleviate the "risk" which largely focused on the production of additional documentation. The spiral model [20] is an alternative life-cycle that directly incorporates risk analysis as one of four major activities that also include: planning, engineering and customer evaluation. Starting in the centre of a spiral the developers work through a planning phase, followed by risk analysis, the engineering of a prototype system and then customer evaluation. The cycle

then repeats and each move around the spiral progresses outwards towards the final system in an evolutionary fashion. Another life-cycle model that is a variant of the waterfall is the "V" model [162] where each step down the left hand side of the "V" has a corresponding validation or verification step on the right hand side. This model emphasises the role of testing where requirements and design documents from the left hand side feed into the validation activities on the right.

In addition to these three examples a number of other life-cycle models exist and the most appropriate model to use for a given project may depend on a number of factors including the type of project, the style of the developers and the organisational maturity of both the developers and the customer. An alternative to the classic life-cycle approaches is to use a meta-model that defines common software engineering activities independently of a particular life-cycle model. Developers can then choose the most-appropriate life-cycle for their project and the activities can be mapped onto the chosen model.

2.1.1 ISO12207 Software Engineering Standard

The ISO12207 Software Engineering Standard [100] describes such a meta-model for software engineering life-cycle processes and the standard includes thirteen activities that can be mapped onto a chosen life-cycle model. The first of the activities is related to starting the methodology, another four are system related and the remaining eight relate to the software itself. The thirteen activities are:

- Process implementation
- System requirements analysis
- System architectural design
- Software requirements analysis
- Software architectural design
- Software detailed design
- Software coding and testing
- Software integration
- Software qualification testing
- System integration

- System qualification testing
- Software installation
- Software acceptance support

The standard notes that “these activities and tasks may overlap or interact and may be performed iteratively or recursively”. From the IEEE Standard Glossary of Software Engineering Terminology [99] the definitions of the software related activities are:

- **requirements analysis** The process of studying user needs to arrive at a definition of system, hardware, or software requirements.
- **architectural design** The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.
- **detailed design** The process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented.
- **coding and testing** Where *coding* is defined as “…the process of expressing a computer program in a programming language” and *testing* is “the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software items”.
- **integration** The process of combining software components, hardware components, or both into an overall system.
- **qualification testing** Testing conducted to determine whether a system or component is suitable for operational use.
- **installation** The period of time in the software cycle during which a software product is integrated into its operational environment and tested in this environment to ensure that it performs as required.
- **acceptance support** Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.

In addition to the eight software related activities defined above an understanding of software maintenance is also required.

2.1.2 Software Maintenance

The term *software maintenance* typically refers to the modification of a software system that has already been deployed to the customer. The process of software maintenance requires iteration through some or all of the previously defined activities and in terms of the waterfall model it could be thought of as a feedback loop to previous stages. The IEEE Standard Glossary of Software Engineering Terminology defines software maintenance as:

- **software maintenance** The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.

The next section of this chapter uses these nine activities within a framework to classify academic papers reporting the application of FCA to software engineering activities.

2.2 FCA in Software Engineering

We conducted a survey of 47 academic papers reporting software engineering applications for FCA. While authors like Snelting [177] have provided an overview of software re-engineering based on concept lattices there has been no broad survey of the literature. The survey papers were analysed using FCA and a formal context was constructed with the papers as the set of objects.

There is also a related body of literature describing the application of FCA to the identification and restructuring of schemas in object-oriented databases, for example, the work of Yahia, Lakhal, Bordat and Cicchetti [233], Schmitt and Conrad [167], and Godin, Mineau, and Missaoui [83]. While a database typically forms the backbone of CASE (Computer Assisted Software Engineering) tools this work is not considered within the context of the survey.

2.2.1 ISO12207 Categorisation

A first classification of the papers that considers the software-related ISO12207 activities as attributes appears in Table 2.1. The intention was not to classify a paper according to a single activity but to record all of the activities supported by the approach described in the paper. Note also that although "coding and testing" appears as a single activity in the standard it has been broken down into two separate attributes for the classification context. This context actually represents a sub-context of the total set of survey attributes and therefore represents a conceptual scale which captures the ISO12207 activities.

References to papers included in the survey use the naming format adopted by the ResearchIndex (formerly known as "CiteSeer") digital library [146]. Paper names are composed of the first author's surname, the last two digits of the year of publication, and the first word of the title (excluding words like "an", "the", "a", etc.). For example Krone and Snelting's paper entitled "On The Inference of Configuration Structures from Source Code" and published in 1994 would appear as *Krone94inference* [122].

The concept lattice corresponding to Table 2.1 appears in Figure 2.2 and it can be seen that 27 out of the 47 papers in total describe applications to both *software maintenance* and *detailed design*. These papers are typically reporting the use of FCA to identify class candidates in legacy code or the maintenance of class hierarchies [1]. Considering the theory behind the subconcept/superconcept ordering within a formal concept lattice this is an obvious application.

An emerging body of literature related to *requirements analysis* can also be seen with 12 of the 47 papers reporting application in this area. It should be noted, however, that papers with common authors are typically reporting work that describes the same example. Across the total set of survey papers it is also noteworthy that there are only two describing applications to *testing* and none of the collection explicitly report application to *software integration, qualification testing, installation, acceptance support* or *coding*.

[1] To avoid confusion the terms "class" or "class candidate" will typically be used to refer to Object-oriented objects as opposed to formal objects in FCA.

	Requirements Analysis	Architectural Design	Detailed Design	Coding	Testing	Integration	Qualification Testing	Installation	Acceptance Support	Software Maintenance
Ammons03debugging [5]					×					×
Andelfinger97diskursive [6]	×									
Arevalo03understanding-a [8]			×							×
Arevalo03understanding-b [9]			×							×
Ball99concept [11]					×					×
Boettger01reconciling [24]	×									
Bojic00reverse [21]	×	×								×
Canfora99case [34]			×							×
Dekel02applications [45]			×							×
Duwel98identifying [58]	×	×								
Duwel99enhancing [56]	×	×								
Duwel00bridging [59]	×	×								
Eisenbarth01aiding [62]		×								×
Eisenbarth01feature [63]		×								×
Eisenbarth03locating [64]		×								×
Fischer98specification [70]		×	×							
Funk95algorithms [74]			×							×
Godin93building [87]			×							×
Godin95applying [84]			×							×
Godin98design [82]			×							×
Huchard99from [96]			×							×
Huchard02when [97]			×							×
Krone94inference [122]			×							×
Kuipers00types [123]			×							×
Leblanc99environment [126]			×							×
Lindig95concept [128]			×							
Lindig97assessing [132]			×							×
Richards02assisting [155]	×									
Richards02controlled [157]	×									
Richards02recocase [158]	×									
Richards02representing [156]	×									
Richards02using [159]	×									
Sahraoui97applying [166]			×							×
Schupp02right [168]			×							×
Siff97identifying [171]			×							×
Snelting96reengineering [175]			×							×
Snelting98reengineering [178]			×							×
Snelting98concept [176]			×							×
Snelting99reengineering [179]			×							×
Snelting00software [177]			×							×
Snelting00understanding [180]			×							×
Streckenbach99understanding [189]			×							×
Tilley03software [201]	×	×								
Tilley03towards [199]		×	×							
Tonella99object [204]			×							×
Tonella01concept [203]			×							×
vanDeursen98identifying [216]			×							×

Table 2.1: Formal context considering the 47 papers in the survey as objects and the ISO software engineering activities as attributes.

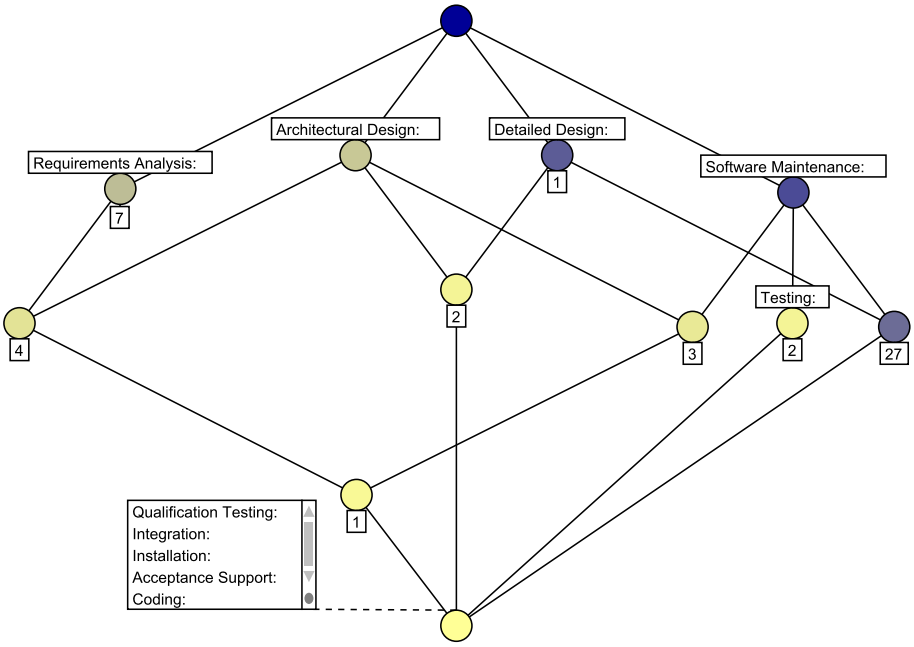


Figure 2.2: The Formal Concept Lattice corresponding to the context in Table 2.1. The objects are the 47 papers included in the survey while the attributes are the activities defined in the ISO12207 standard.

Note that only object counts are shown on the diagram and the node colour also indicates the distribution of objects. A lighter shade implies less objects while a darker shade corresponds to a higher object count.

In addition to the attributes appearing in the context shown in Table 2.1, there were 133 attributes used in total to categorise the papers in the survey. These attributes included the names of the authors, citations of other papers in the survey, the year of publication, inputs, outputs, target application languages (e.g. C++, Java) and the "size" of any reported application target.

2.2.2 Target Application Language

The context in Table 2.2 represents the application of the approach described within a paper to a particular language. The attributes here are the programming languages: C, C++, COBOL, FORTRAN, Java, Modula-2, Smalltalk, and the design or specification languages: OMT, UML and Z. Both procedural and object-oriented languages are represented. The

attribute values record the size of any reported target application in KLOC ("thousand Lines Of Code") — for example, 106 KLOC represents an application containing 106,000 lines of source code. KLOC is also sometimes referred to as SKLOC (Source Thousand Lines Of Code) and is a metric that is often reported to indicate project size in software engineering. Where a paper contains multiple examples for the same language only the largest application size is shown.

While there is some debate about the usefulness of size-oriented metrics like KLOC [154] it does give a raw indication of application size. Within the set of survey papers it may also be indicative of tool support. The application of these techniques to moderately sized projects demonstrates the potential for real-world application.

A number of the papers report application to a specific language but do not report the size of a particular application and the KLOC value for these papers appears as "0" in the context. It is also interesting to note that where a non-zero value repeats in the context it typically refers to the same example being reported in a number of papers. For example, the 1.6 KLOC C application appears in the papers *Funk95algorithms, Krone94inference, Snelting96reengineering, Snelting98concept* and *Snelting00software*. Similar patterns can also be seen for the 106 KLOC FORTRAN, 100 KLOC COBOL and 1.5 KLOC Modula-2 applications.

Figure 2.3 presents a concept lattice that treats Table 2.2 as a simple one-valued context where any KLOC value $\geq 0 \Rightarrow gIm$. It can be seen that 14 of the 47 papers do not report any application to a particular programming or design language. Also of note is the paper *Snelting00software* [177] which reports applications to all of the programming languages except Smalltalk. This is a paper by Snelting that surveys earlier results from a number of papers he has either authored or co-authored.

2.2.3 Reported Application Size

The line diagram in Figure 2.4 also summarises the context in Table 2.2 as an inter-ordinal scale that only considers the maximum reported size in KLOC across all programming languages for each paper. Note that 17 of the 47 papers now appear at the supremum. In

	C	C++	COBOL	FORTRAN	Java	Modula-2	OMT	SmallTalk	UML	Z
Ammons03debugging	0									
Andelfinger97diskursive										
Arevalo03understanding-a								0		
Arevalo03understanding-b								0		
Ball99concept	0									
Boettger01reconciling										
Bojic00reverse		0								
Canfora99case			200							
Dekel02applications					0					
Duwel98identifying										
Duwel99enhancing										
Duwel00bridging										
Eisenbarth01aiding	0									
Eisenbarth01feature	76									
Eisenbarth03locating	1,200									
Fischer98specification										
Funk95algorithms	1.6									
Godin93building								0		
Godin95applying										
Godin98design										
Huchard99from					0					
Huchard02when									0	
Krone94inference	1.6									
Kuipers00types			100							
Leblanc99environment		0			0		0		0	
Lindig95concept										
Lindig97assessing			5	106		1.5				
Richards02assisting										
Richards02controlled										
Richards02recocase										
Richards02representing										
Richards02using										
Sahraoui97applying	47									
Schupp02right		0								
Siff97identifying	28									
Snelting96reengineering	1.6									
Snelting98concept	1.6		0	106		1.5				
Snelting98reengineering		0								
Snelting99reengineering	0				9					
Snelting00software	1.6	0	0	106	9	1.5				
Snelting00understanding		0			12					
Streckenbach99understanding		0			12					
Tilley03software										0
Tilley03towards										0
Tonella99object		21								
Tonella01concept	249									
vanDeursen98identifying			100							

Table 2.2: A Formal Context showing reported application languages for the 47 papers in the survey. The attribute values represent the size of the application in KLOC ("thousand Lines Of Code"). A KLOC value of "0" indicates that the paper reported application to a particular language but no size was quoted.

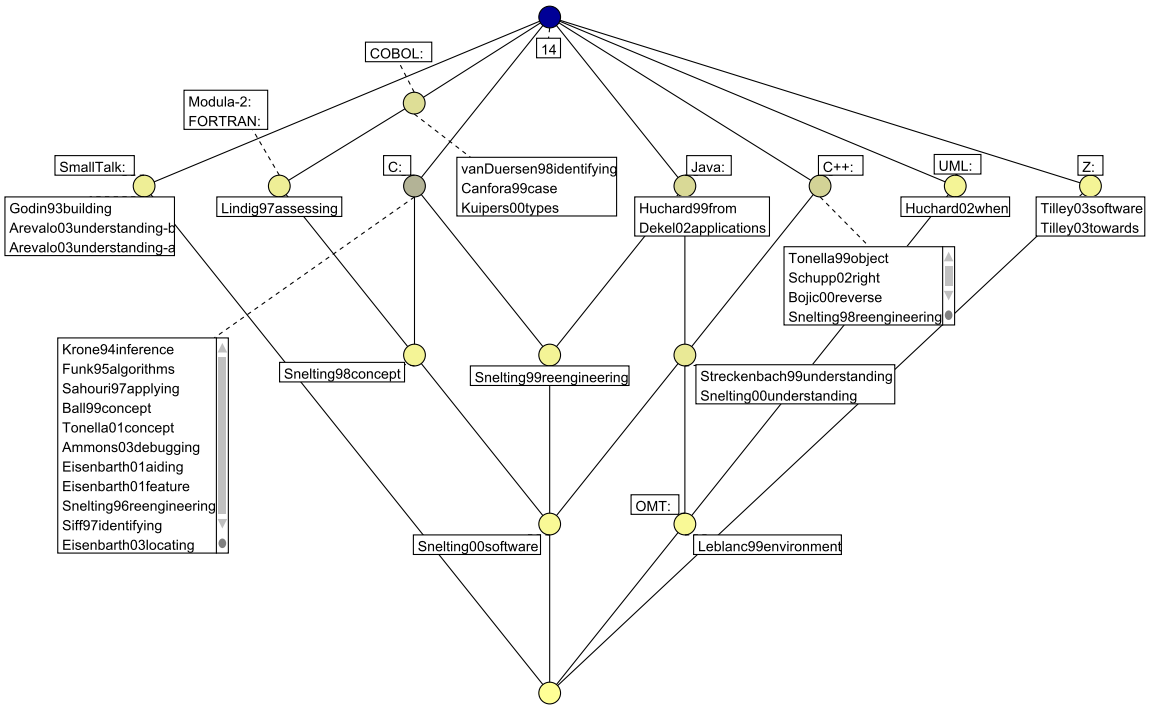


Figure 2.3: Formal Concept lattice based on the context in Table 2.2 showing reported application by language.

addition to the 14 papers that do not report application to a particular language this also includes those papers reporting exclusive application to design or specification languages. KLOC is not considered to be a “meaningful” measure for UML, OMT, and Z.

From Figure 2.4 it can be seen that there are eight papers in the survey reporting application to systems of 100 KLOC or more, however, these actually refer to only five different examples. The analysis of a 106 KLOC FORTRAN system is discussed in the three papers: *Lindig97assessing*, *Snelting98concept* and *Snelting00software*. In addition the 100 KLOC COBOL examples reported by Kuipers and Moonen in *Kuipers00types* and Van Deursen and Kuipers in *vanDeursen98identifying* also describe the same application example.

The largest application in the survey describes the analysis of a 1,200 KLOC semiconductor testing tool written in C. The work by Eisenbarth, Koschke and Simon in *Eisenbarth03locating* [64] is an order of magnitude larger than any of the other examples

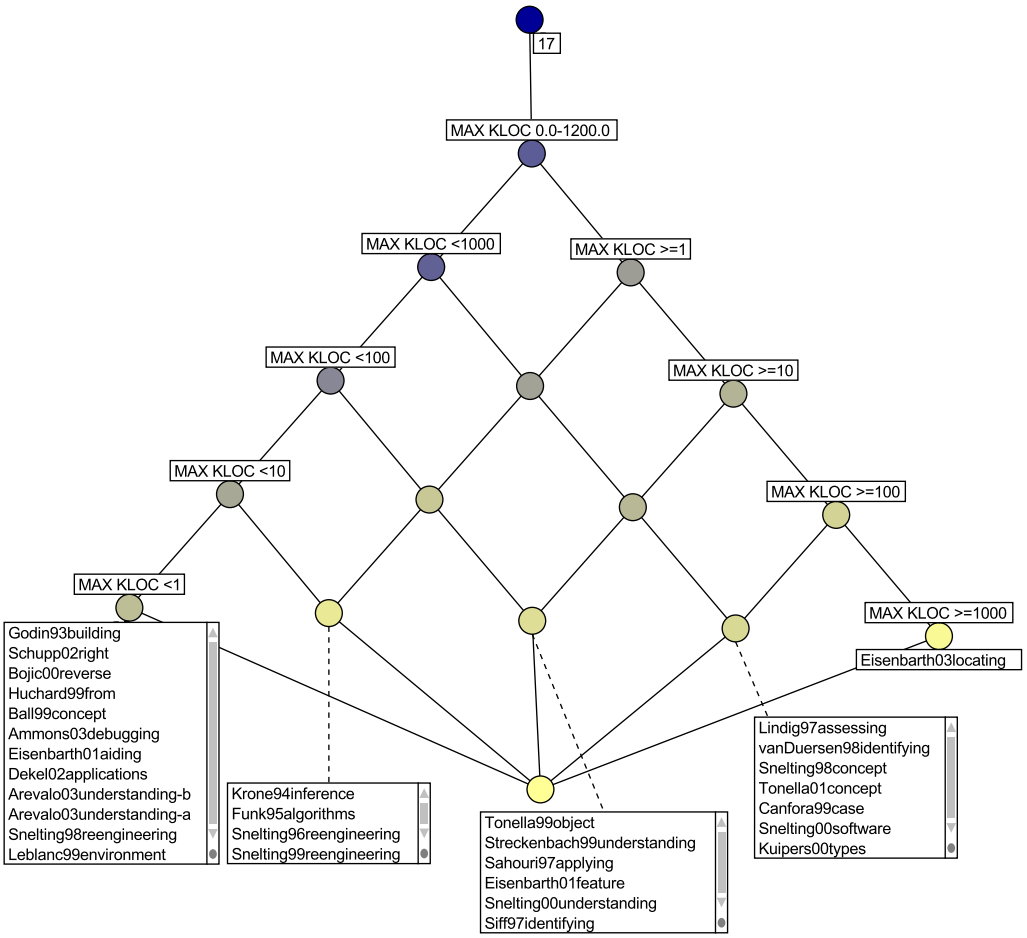


Figure 2.4: An Inter-ordinal scale based on the context in Table 2.2 using the maximum KLOC across all programming languages for each paper.

and demonstrates that FCA-based software analysis tools are capable of handling real-world projects. An overview of the tool implementations described in these papers is presented in Section 5.2.10.

2.3 Support for Late-phase Activities

Figure 2.5 again presents the ISO12207 categorisation of the 47 survey papers, however, the paper names are listed instead of the counts shown in Figure 2.2 . Thirty-three of the survey papers have been classified as *Software Maintenance* applications. Additionally, the work of Eisenbarth et al. [62, 63, 64] has also been categorised as *Architectural Design*. The

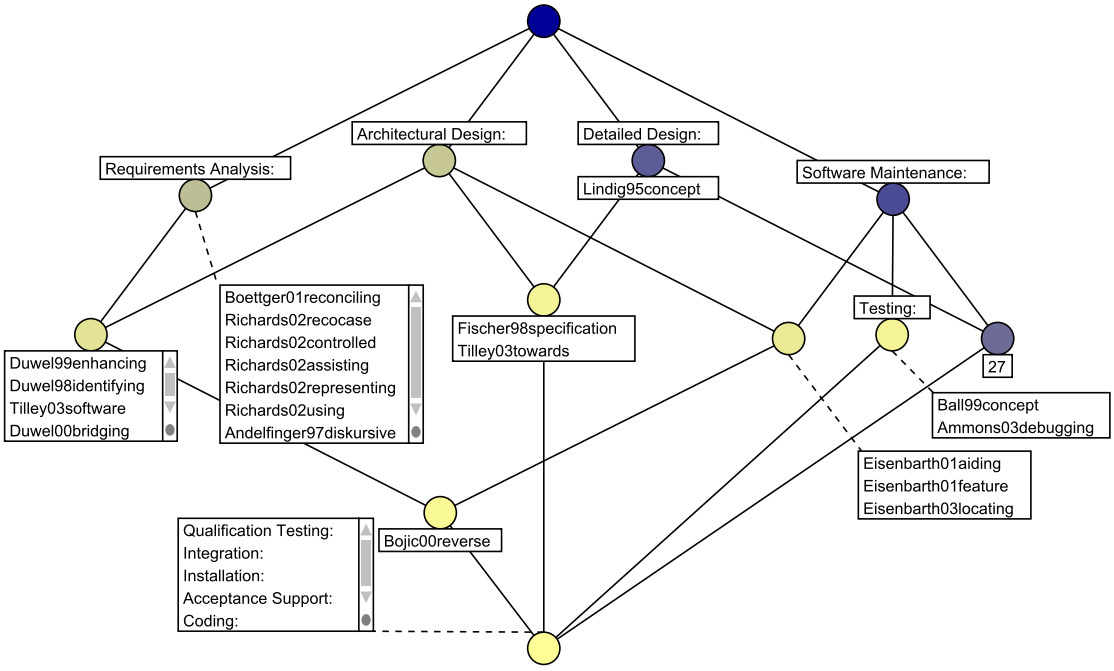


Figure 2.5: The ISO12207 categorisation diagram from Figure 2.2 showing the paper names.

papers by Ammons, Mandelin, Bodik and Larus [5], and Ball [11] incorporate *Testing* and these are the only two papers in the survey to address this activity. Bojic and Velasevic [21] discuss applications to round-trip engineering and recovering UML use-cases. Their work has therefore also been included under the *Requirements Analysis* category even though it is concerned with re-engineering existing systems.

Eisenbarth et al. describe a technique for locating the computational units within software that actually implement a feature or functionality of interest. They combine both static and dynamic analysis and of particular note is the application of their technique to the 1,200 KLOC example mentioned in the previous section. A number of test cases or “scenarios” are constructed which cover the use-cases of interest and these are treated as the formal objects in their analysis. The computational units executed during runs of the program are then considered as the formal attributes. The attribute contingents of object concepts in the resulting lattice are of particular interest since they contain the program artefacts introduced by specific scenarios.

The work of Ammons et al. represents one of the few existing approaches that incorporates both formal methods and FCA. The work was briefly summarised in Section 1.4 but essentially uses FCA to aid in the testing and debugging of temporal specifications. Large specifications are verified using tools that check them against a number of programs and these checks can produce hundreds or thousands of execution traces. A concept lattice is used to cluster program execution traces together so that an expert can assess and classify clusters of traces rather than classifying each of the traces individually.

Ball examines test-coverage by comparing the implicational logic in the concept lattice generated from traces extracted from test programs with dominance and post-dominance relationships extracted by static code analysers. A computer program essentially consists of a large number of instructions and each instruction is identified by its position within the program. A run of a computer program produces a *trace* listing the sequence of instructions that were run. Two notions concerning instructions, with respect to a collection of traces, are important: *dominance* and *pre-domination*. An instruction x dominates another instruction y if any trace prefix that ends on y contains x. In other words x dominates y if the only way to execute y is to have already executed x. Similarly, x post-dominates y if any trace postfix starting with y also contains x. In other words x post-dominates y if any execution of y indicates that x will subsequently be executed. Any additional implications in the concept lattice are also considered to see if they can be removed by the introduction of a new test.

Bojic and Velasevic report a similar approach but additionally the artefacts within the attribute contingents are arranged as UML diagrams using a UML reverse engineering tool. In this way the specific parts of the software architecture related to use-cases can be extracted and viewed. The capability is particularly useful in the preparation of traceability in the software engineering process whereby aspects of the system architecture can be traced back to requirements.

Interestingly, the papers by Ammons et al., Ball, Bojic and Velasevic, and Eisenbarth et al. all deal with the dynamic analysis of software behaviour. Across the collection of papers

a wide variety of inputs for analysis are used including source code, class files, profiler output, system descriptions and documentation. The choice of formal objects include code segments, language features, and the names of packages, classes and methods.

The remaining 27 papers can be broadly categorised into three groups:

- analysis of software configurations
- modularisation of legacy code
- transformation of class hierarchies

and these approaches are summarised in Sections 2.3.1, 2.3.2, and 2.3.3 respectively.

2.3.1 Analysis of Software Configurations

Snelting [175, 176, 122, 74] used FCA to analyse preprocessor commands in legacy C programs including "Xload" and "RCSedit" in order to examine the configuration structure. The formal objects are code fragments included by the preprocessor commands, while the formal attributes are disjunctive expressions governing the inclusion of the code fragments. The concept lattice is constructed and the notion of an interference is introduced. An *interference* is a meet-reducible concept with a non-empty extent. Two types of undesirable interference are identified, those corresponding to illegal configurations — for example, an interference between XWINDOWS and DOS — and those corresponding to orthogonal attributes — for example, an interference between a variable related to the graphics subsystem and one related to the operating system.

In order to make the resulting concept lattices more manageable *horizontal decompositions* are introduced [74]. The decompositions are based on the idea of a horizontal sum where the constituent elements of the sum are usually disjoint. However, experiments with legacy systems revealed that few configuration lattices can be directly decomposed into a horizontal sum of disjoint sub-lattices. In order to simplify the configuration structure the notion of a *k-interference* is introduced. A *k*-interference is a collection of *k* meet-reducible incomparable concepts whose down-set removal yields a decomposition into a disjoint horizontal sum. The concepts involved in such *k*-interferences are of particular interest since they are most likely interferences between orthogonal aspects of the system configuration.

Other techniques to simplify the concept lattice include limiting the nesting depth of preprocessor commands considered and merging rows which differ by fewer than k elements. These techniques are of use when the objective is to get an overview of the configuration structure present in a software program.

2.3.2 Modularisation of Legacy Code

Legacy programs written in languages where access to common data structures is normally the case, e.g. FORTRAN and COBOL, have been considered by Van Deursen and Kuipers [216], Kuipers and Moonen [123], Lindig and Snelting [132], and Canfora, Cimitile, De Lucia, and Di Lucca [34].

Van Deursen and Kuipers compare the use of formal concept analysis for grouping fields within a large legacy COBOL program to that of hierarchical clustering. Hierarchical clustering involves the definition of a distance metric between COBOL procedures, extending the metric to sets of procedures, starting with every procedure in its own cluster and then repeatedly merging the two closest clusters to produce a binary tree of clusters. This approach is generally criticised because it can yield different inputs for the same data if several clusters are equidistant and different results are obtained for slightly different distance metrics. In contrast, the results produced by FCA are always the same, not dependent on the definition of a distance metric, and were much closer to that produced by software engineers familiar with the legacy system. Since the objective was to focus on domain specific procedures rather than those performing system functions, procedures having a high degree of fan-in were judged as being system procedures and were discarded. This judgement was controlled by an operator set threshold.

Canfora et al. follow a similar approach but are interested in organising a legacy COBOL system into components suitable for distribution via the Common Object Request Broker Architecture (CORBA). They consider programs and their use of files representing relational tables. The formal context was pruned by removing objects and attributes in isolated concepts — those concepts that are directly below the top concept and directly below the bottom concept and therefore do not have any intent or extent intersection with

any other concepts. Relational table files having the same structure were also merged. This case arises when several files are used to perform some operation on a table, for example, sorting. Programs that used only a single file were also removed. Canfora et al. apply their rules until no more formal objects or formal attributes can be removed. The result was a concept lattice that was almost horizontally decomposable — in the sense of Snelting et al. — into four domain areas, except for a number of interferences corresponding to operations involving more than one domain area.

The task of deriving object-oriented models from legacy systems written in C has also been considered by Sahraoui et al. [166], Siff and Reps [171], and Tonella [203]. The general approach is to consider C functions as formal objects and the attributes as either commonly accessed data structures or fields within commonly used structures.

Both Siff and Reps, and Tonella are concerned with re-organising the functions into a different, perhaps more fine grained, module structure based on the access of functions to either common data structures [203], or fields within commonly accessed data types [171].

In Tonella's approach a modular structure results from a partitioning of the formal objects. Candidate partitions are generated from a choice of concepts having pairwise disjoint extents. Each formal attribute is then assigned to the chosen concept that has the largest number of objects with that attribute, i.e. for attribute m we find concept c that maximises $m' \cap Ext(c)$ and assign m to that concept. One partitioning set of concepts is considered better than another if the number of objects having an attribute not assigned to their concept (i.e. the concept from the chosen set containing the object) is smaller and the number of concepts is larger. The approach searches over the possible choices of concepts seeking to optimise these two criteria.

2.3.3 Transforming Class Hierarchies

Snelting [177], and Snelting and Tip [178, 179, 180] explain a mechanism to re-organise class hierarchies using FCA. Their aim is to find imperfections in the design of the hierarchies based on how the class is actually used by applications. Variables in C++ are taken as formal objects and methods and fields of the objects to which the variables

refer are taken as formal attributes. A variable is associated with a field or method if that variable is used to access the method or field. A number of rules are employed to account for assignment between variables and conservatively account for dynamic dispatch. The main focus of investigation is the objects that exist during a run of a program and Snelting and Tip access these using static analysis and via the medium of variables.

Schupp, Krishnamoorthy, Zalewski and Kilbride [168] consider class hierarchies in the C++ Standard Template Library (STL). They have classes as formal objects and documented properties of the classes as formal attributes. The notions of "well abstracting", "lacking orthogonality" and "lacking refinement" are then introduced to describe class libraries. However, rather than inspecting various aspects of the structure they attempt to construct the whole concept lattice, render it and then draw conclusions. Inspection of aspects of the STL reveal a very regular structure. An example presented by Tilley, Cole, Becker and Eklund [200] shows three complementary pairs of attributes: unique and multiple associative, sorted and hashed, and pair and simple associative. Complementary attributes are related by *exclusive or* — in other words all objects have exactly one of the two attributes.

Godin and Mili [87] consider a context where the formal objects are messages (methods in Smalltalk) and formal attributes are classes. The aim of their approach is to build analysis-level class hierarchies that can be maintained as the class evolves through design and implementation phases. They consider concepts having an empty attribute contingent, i.e. those not labelled by a class, as new class candidates. Godin, Mili, Mineau, Missaoui, Arfi, and Chau [82] further incorporate static call graph information into the concept lattice.

While Leblanc, Dony, Huchard and Libourel [126] describe an environment for re-engineering class hierarchies, Huchard and Leblanc [96] consider a concept lattice generated with classes as formal objects and attributes derived from method signatures. Their approach thereby includes information about parameter types and return values. Each concept is considered as a candidate for a Java interface.

Huchard, Roume and Valtchev [97] address the problem of representing and analysing data via FCA where relationships exist between the formal objects. The binary inter-object

relationships are represented by a *relational context family*. Their approach is applied to UML class diagrams representing both classes and association relationships between classes where the classes are considered as the formal objects and the variables and methods as attributes.

Tonella and Antoniol [204] attempt to recover design patterns in C^{++} source code using a context in which the formal objects are *n*-tuples (in practice triples are used) whose elements are types in the software, and formal attributes are triples of the form (i, j, r) where i and j are indexes into the *n*-tuple and r is a relation type. For example, an object (A, B, C), being associated with an attribute $(1, 2, \mathit{derived} - \mathit{from})$ would indicate that A is derived from B. Tonella and Antoniol discover as one of the concepts in the concept lattice the well known "adapter pattern".

The work of Arévalo [8], and Arévalo, Ducass and Nierstrasz [9] is also concerned with detecting patterns in software via FCA. While their work is similar to that of Tonella and Antoniol they apply the approach to Smalltalk and also take into account behavioural information related to the derivation of subclasses. These behavioural dependencies result when a method is added, modified, overridden or removed in a subclass.

While not actually related to the transformation of classes, Dekel's paper [45] analyses Java classes to suggest source code reading order for code review and inspection purposes.

Having provided a brief overview of the late-phase approaches, the existing techniques that support early-phase software engineering activities are now discussed.

2.4 Support for Early-phase Activities

While the bulk of the papers in the survey report applications to late-phase activities, 14 of the 47 papers are concerned with early-phase software engineering. The techniques described in 27 of the 33 *Software Maintenance* papers also necessitate design reviews or at least proposed changes to the design of legacy systems and as such they have also been categorised under *Detailed Design*. This section will present those approaches that do not fall under the *Software Maintenance* category, that is those approaches that apply to the development of new systems rather than the re-engineering of existing systems. The main

approaches from these 14 papers are summarised in the following sections.

2.4.1 Requirements Analysis

Andelfinger's thesis in *Andelfinger97diskursive* [6] describes a discursive environment for requirements gathering based on Habermas' philosophical theory of "communicative rationality" — a discursive form of collective reasoning. Habermas described a somewhat idealistic discursive environment that attempts to make any agendas during negotiation obvious and considers all viewpoints equally. Andelfinger presents this environment as a way of addressing the so-called "pragmatic gap" between the views of users and developers.

Within the thesis FCA is used as a question answering and discussion promotion tool. The value of unlabelled concepts is highlighted as it promotes questions about what is missing which may be indicative of incomplete requirements. While the three case studies presented by Andelfinger are not directly related to software engineering they parallel standard problems in requirements gathering. It is interesting to note that the second case study describes gathering requirements for an FCA-based retrieval system for the library at the Centre of Interdisciplinary Technology Research (in German the "Zentrum für Interdisziplinäre Technikforschung"(ZIT)) [161].

2.4.2 Use-case Analysis

Use-cases are a tool used in requirements gathering and analysis where a task is described from a certain perspective or role. Typically these descriptions are written in natural language although sometimes controlled vocabularies are used. Böttger and Richards et al. describe a technique to reconcile use-cases in the papers: *Richards02representing* [156], *Richards02controlled* [157]. *Richards02using* [159], *Boettger01reconciling* [24], *Richards02assisting* [155] and *Richards02recocase* [158]. Their technique reconciles multiple use-cases written by different stakeholders. The approach could be used by system designers to identify both the overlap between use-cases and points of conflict or difference between them.

Their tool RECOCASE (RECONciling CASE tool) exploits a Prolog answering system called ExtrAns and uses LinkGrammar — an English parser that uses link grammar theory — to convert sentences into syntactically legal "flat logical forms" (FLFs). A context is produced where the sentences are the objects and the FLFs are broken into word phrases which are treated as the attributes. A number of the papers also report a brief study of line diagram comprehensibility for comparing use-case descriptions using second year university Analysis and Design students.

The work of Düwel in *Duwel99enhancing* [56] and Düwel and Hesse in *Duwel00bridging* [59], and *Duwel98identifying* [58] attempts to identify class candidates in use-cases. The use-cases themselves are considered as objects in a formal context and nouns identified within the text are considered as attributes. A case study that contrasts this approach with an existing design for a mass-transit railway system is reported in *Tilley03software* [201] and the details are presented in Chapter 3.

2.4.3 Software Component Retrieval

In *Lindig95concept* [128] Lindig describes a retrieval system that could be used for retrieving software components from a library indexed by keywords. The formal context is constructed using the components as objects and the keywords as attributes. An example based on 1,658 online documents relating to Unix commands is presented. The retrieval system provides a query by refinement interface in which a boolean query, B, is mapped to the formal concept, (B', B''). The lower cover of this concept within the lattice is then offered as a set of possible refinements to the user. Retrieval applications are also discussed in *Godin95applying* [84]. Outside of the software engineering domain FCA has also been used for more general information retrieval applications, for example the work of Carpineto and Romano [35] for text document retrieval and the ZIT library retrieval system discussed by Wille and Rock [161].

Fischer builds on the component retrieval work of Lindig in the paper *Fischer98specification* [70]. The approach combines formal methods and FCA for browsing and navigating a software component library. Components in the library

are associated with formal specifications that capture their behaviour in the form of axioms describing pre-conditions and post-conditions. The formal specifications are then used instead of simple keyword searches to retrieve components based on explicit properties required for the components selection or based on implicit similarity with other components.

A formal context is constructed using the component specifications as both the objects and attributes. Additionally, functions or partial functions within the specifications are also considered as formal attributes. Automated theorem provers are used to deduce valid relations between pairs of components over a number of different relation types including refinement and matching. A formal concept lattice is then computed that is used as a structure for navigating and retrieving components from the library. The resulting lattice can also be used to improve the library. Unlabelled concepts and extents containing intuitively "unexpected" components may indicate missing attributes or features that can be added to the library.

The author's own work reported in *Tilley03towards* [199] also explores the application of formal methods, in particular formal specification in Z, and FCA. FCA is used to facilitate the visualisation and navigation of Z specifications and a tool prototype is discussed. The approach and implementation are detailed in Chapter 4 and Chapter 5 respectively.

The work shares a number of parallels with that of Fischer described above. For example, both approaches provide browsing and navigation over specifications describing pre-conditions and post-conditions. However, while both Fischer and Lindig used the concept lattice as a navigation structure, the work described here also uses line diagrams to aid in a user's understanding of a formal specification. Fischer experimented with lattice visualisation but found the underlying concept lattice too large and complex for presentation. These problems are addressed here via a number of abstraction mechanisms which are described in Section 4.3.

2.5 Summary of Results

The preceding sections of the chapter have provided an overview of FCA applications to software engineering via a paper survey. From the initial ISO12207 categorisation of the papers it is evident that the majority describe software maintenance and re-engineering applications and future research is likely to remain in this area. In these later stages the software is already highly structured and therefore more amenable to analysis than in the earlier phases where things are less well-defined. However, the survey does include a number of papers reporting applications to the early-phase approaches of *requirements analysis* and *architectural design*. None of the papers support *acceptance support*, *integration*, *coding*, *installation* or *qualification testing* and the papers by Ball, and Ammons et al. stand out as the only *testing* related applications.

Applications for the approaches described in the papers cover a range of procedural, object-oriented and design languages. The procedural applications are typically looking at ways of re-engineering the code in a modular or object-oriented fashion by exploiting the subconcept/superconcept structure inherent in the concept lattice. However, the common thread running through all the papers is the use of FCA to extract understandable structures that organise the artefacts of software systems.

Eight of the papers describe the use of FCA-based approaches to analyse or re-engineer applications of 100 KLOC or more. The largest of these examples, reported by Eisenbarth et al., is the analysis of 1,200 KLOC of control software for a semiconductor testing tool. These papers demonstrate applicability and scalability beyond mere toy examples to real world software engineering problems with tool support.

2.6 FCA as a Literature Survey Tool

While the survey analysis in the first half of the chapter described attributes that were specific to the domain of software engineering the remainder of the chapter examines two attributes that apply to academic papers in general. Section 2.6.1 explores collaboration between authors within the set of survey papers while Section 2.6.2 explores the “impact”

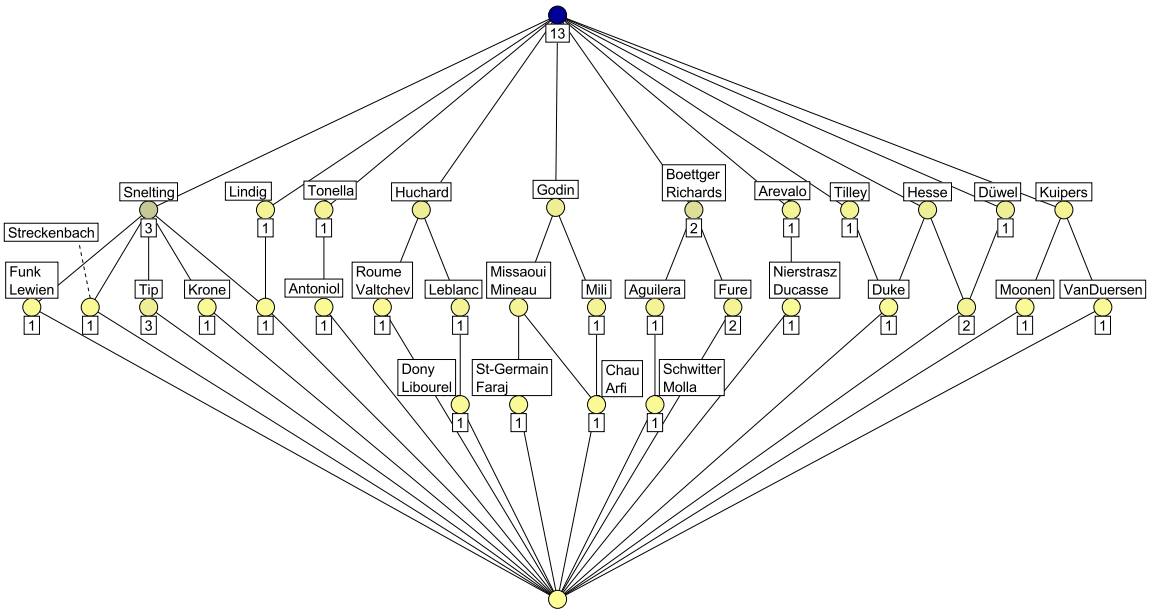


Figure 2.6: Lattice showing collaboration between authors within the set of survey papers. Note that only papers where the authors have worked with different co-authors appear.

of the papers based on the citations within the papers. An existing citation impact mechanism — the ResearchIndex digital library — is also introduced in Section 2.6.2 for comparison.

2.6.1 Author Collaboration

A summary of collaboration between authors within the set of survey papers is presented in Figure 2.6. This concept lattice represents those authors who have collaborated on papers with different authors. There are 13 papers at the top of the lattice whose authors only appear once across the 47 papers or who have only worked with the same co-authors. Only the size of the object contingent which represents the number of papers is shown for each concept.

The diagram can be horizontally decomposed into eight sub-lattices which indicates that research has been performed rather independently within these research groups. The largest of these groups are led by Snelting, Huchard, Godin, Boettger and Richards, and Hesse and there are no joint publications across these groups. Each of the structures below

these researchers represent collaboration across multiple papers containing at least five different authors with the common author, or authors, appearing at the top.

Snelting's collaboration with a number of different authors is the likely reason for the large number of language applications recounted in *Snelting00software*. Snelting has authored or co-authored ten of the papers — the highest number of any author in the survey. Three of these papers are co-authored with Tip and while *Snelting98reengineering*, *Snelting99reengineering* and *Snelting00understanding* represent updated versions of the same paper it can be seen from Figure 2.3 that each paper includes examples with applications to different languages.

While the structure below "Snelting" has a high degree of "fan-out" the structures beneath "Huchard", "Godin", and "Boettger" are more linear. The papers below "Boettger and Richards", for example, result from common subsets of authors across a number of papers that are typically reporting different aspects of the same work. *Boettger01reconciling* includes Schwitter and Mollá among the authors and this paper provides details of the ExtrAns tool which they implemented [170, 169].

The 13 papers represented by the count at the top of Figure 2.6 do not share any intersection of authors with other papers in the survey set. They have been omitted to increase the readability of the line diagram. These papers represent concepts that are only connected to the top and bottom elements of the lattice. Alternatively, they can be considered as sub-lattices resulting from horizontal decomposition that contain only a single concept in addition to the top and bottom elements.

Figure 2.7 presents an image produced by Snelting's KABA tool [189, 177] showing horizontal decompositions in Java code. The image is taken from *Snelting00software*. The problem of displaying diagrams with a high degree of fan-out at the top and high fan-in at the bottom is overcome by effectively turning the top and bottom elements of the lattice into rails. This style of display would also be a suitable representation for the author collaboration in Figure 2.6.

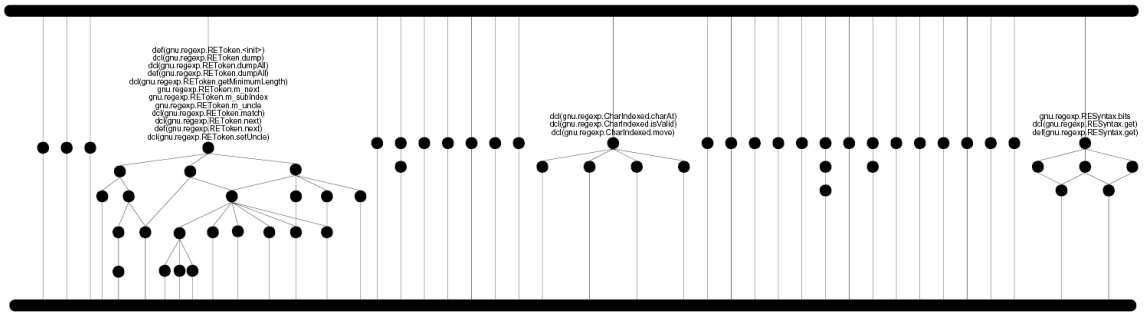


Figure 2.7: Image produced by Snelting's KABA tool showing horizontal decompositions in Java code. This image appears as Figure 8 in *Snelting00software* [177].

2.6.2 Citation Patterns

As a whole the papers included in the survey embody the ideas of a particular community or sub-culture — the community of researchers applying FCA to software engineering. In addition to identifying collaboration structures within this community it is also possible to gain some insight into the perceived "impact" or influence of a particular paper. The citation patterns within the set of survey papers provide some measure of a paper's influence, where papers with the most impact are considered to be highly cited.

Figure 2.8 presents a line diagram showing the transitive closure of citations within the set of survey papers. If a paper *B* cites paper *C*, and paper *A* cites paper *B* then *A* transitively cites *C*. For example, the paper *Ball99concept* cites among others the paper *Siff97identifying* which in turn cites *Lindig97assessing*. So *Ball99concept* indirectly cites *Lindig97assessing*. The algorithm to compute the closure is discussed later in this section.

The attribute labels linked to the top of the concepts represent papers which have been cited by other papers within the survey collection while the object labels linked to the bottom of concepts represent papers which contain citations. At the top of the line diagram there are 9 papers listed and these papers have not cited any of the other literature within the survey set. There are a number of explanations for the location of these papers. The earliest papers in the survey, *Godin93building* and *Krone94inference*, by definition have no earlier work to cite within the survey collection. *Andelfinger97diskursive* is a paper written

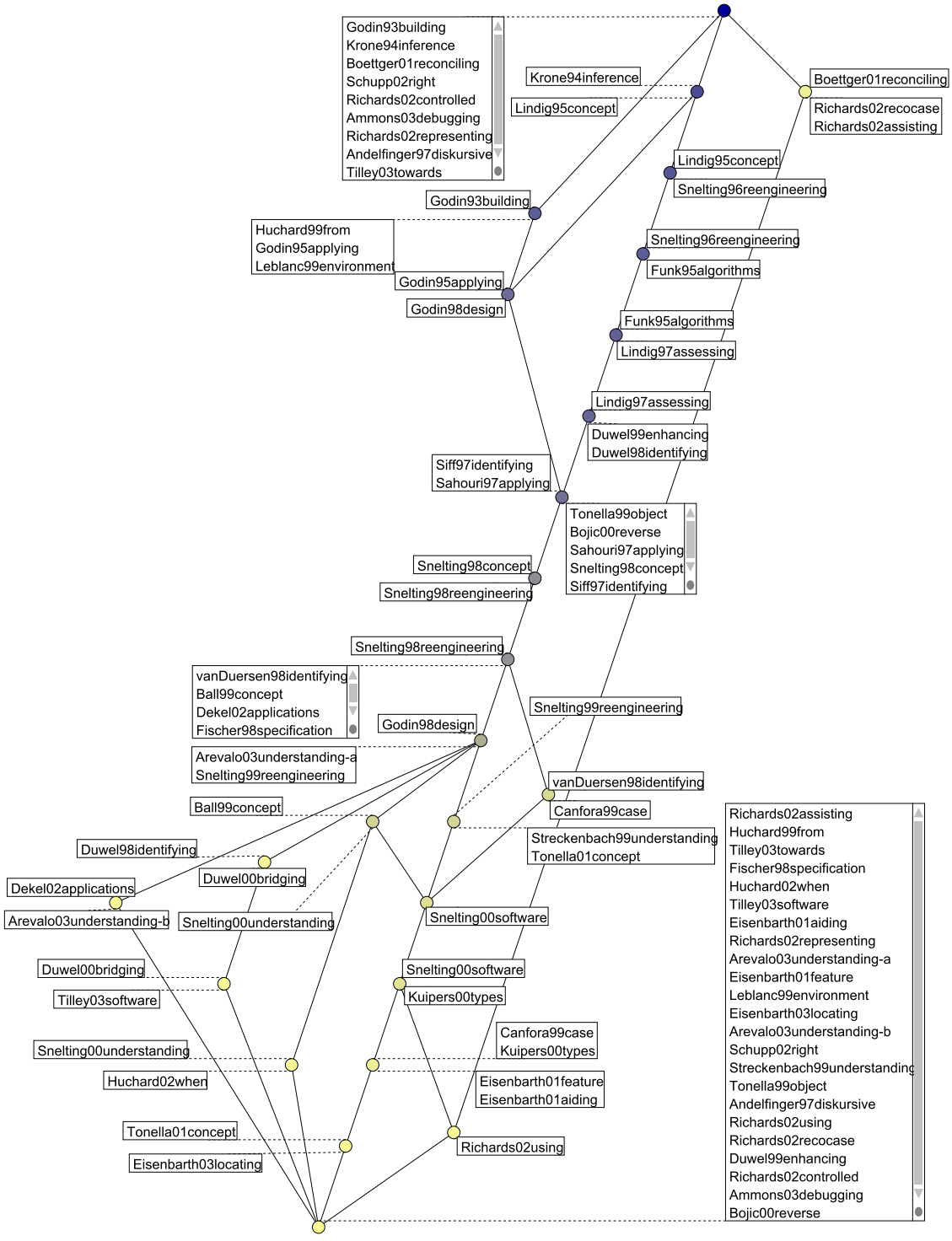


Figure 2.8: Formal concept lattice showing transitive closure of citations within the set of survey papers.

in German and the list of citations was unavailable. The remaining papers all cite the work of Wille as FCA background but do not build directly on any of the work described in the other papers.

At the bottom of the diagram are 20 papers which are not cited within the survey collection. Papers with earlier publication years appear to have been ignored by the community while more recent papers may not have been around long enough to be cited yet. *Andelfinger97diskursive* also appears in this list and this could be because it is the only German language paper in an otherwise English language set.

Papers with the most impact appear at the top of the diagram with long chains (i.e. containing many concepts) beneath them. It can be observed that most of the work is nearly linear in terms of citations which reflects some coherence within the community. For example, Snelting and Godin cite each other's work before large forks appear in the structure.

The structure down the right hand side of the line diagram is also interesting. The papers by Böttger and Richards et al. either contained no citations within the survey set or they cited their own work in *Boettger01reconciling*. *Richards02using*, however, cites *Snelting00software* which connects their work back into the main "trunk".

Computing the Citation Closure

The algorithm used to compute the closure of citations within the set of survey papers is presented in Figure 2.9. The algorithm takes as input a context $\mathbb{K}$ representing citations of other papers in the survey. G is the set of papers and M is the set of cited papers. Here $G = M$. For example, the paper *Ball99concept* in Table 2.3 cites the papers *Siff97identifying, Snelting96reengineering* and *Snelting98reengineering*. Note that uncited papers have been excluded from the attribute set to increase the readability of the table.

The outer loop of the algorithm contains a terminating condition. The context $\mathbb{K}_{old}$ stores a copy of the context at the start of each iteration and if no further changes occur then the algorithm terminates and $\mathbb{K}$ contains the citation closure. This is a simple adaptation of the algorithm used for computing the closure on a set of functional dependencies during relational database normalisation [65]. The next two loops iterate along each row and if

	Ball99concept	Boettger01reconciling	Canfora99case	Dekel02applications	Duwel98identifying	Duwel00bridging	Funk95algorithms	Godin93building	Godin95applying	Godin98design	Krone94inference	Kuipers00types	Lindig95concept	Lindig97assessing	Sahraoui97applying	Siff97identifying	Snelting96reengineering	Snelting98concept	Snelting98reengineering	Snelting99reengineering	Snelting00software	Snelting00understanding	Tonella01concept	vanDeursen98identifying
Ammons03debugging																								
Andelfinger97diskursive																								
Arevalo03understanding-a								×		×									×					
Arevalo03understanding-b				×						×									×					
Ball99concept																×	×		×					
Boettger01reconciling																								
Bojic00reverse														×		×								
Canfora99case														×		×								×
Dekel02applications								×								×		×	×					
Duwel98identifying														×										
Duwel99enhancing														×										
Duwel00bridging					×					×				×					×					
Eisenbarth01aiding			×								×	×		×	×	×	×		×					×
Eisenbarth01feature			×								×	×		×	×	×	×		×					×
Eisenbarth03locating	×		×								×	×		×	×	×	×		×				×	×
Fischer98specification											×		×	×	×	×		×	×					
Funk95algorithms											×						×							
Godin93building																								
Godin95applying								×																
Godin98design								×	×		×													
Huchard99from								×																
Huchard02when								×											×			×		
Krone94inference																								
Kuipers00types														×		×	×	×	×		×			×
Leblanc99environment								×																
Lindig95concept											×													
Lindig97assessing							×				×						×							
Richards02assisting		×																						
Richards02controlled																								
Richards02recocase		×																						
Richards02representing																								
Richards02using		×																			×			
Sahraoui97applying									×					×		×								
Schupp02right																								
Siff97identifying														×	×		×							
Snelting96reengineering											×		×											
Snelting98concept							×				×		×	×		×	×							
Snelting98reengineering								×			×			×		×	×	×						
Snelting99reengineering							×	×		×	×			×		×	×	×	×					
Snelting00software	×										×			×		×	×		×	×				×
Snelting00understanding	×							×		×	×			×			×	×	×					
Streckenbach99understanding																	×	×	×					
Tilley03software						×								×					×					
Tilley03towards																								
Tonella99object											×			×		×	×							
Tonella01concept							×				×			×		×	×			×				
vanDeursen98identifying														×		×		×	×					

Table 2.3: Formal context showing direct citations within the set of survey papers. Here the objects are the papers and the attributes are the paper citations. Note that uncited papers have been excluded from the attribute set to increase table readability.

$(p, cp) \in I$ then paper p is citing paper cp. The innermost loop traverses the (indirect) citations *ic* within the cited paper *cp*. The citations for paper p are then updated if there is a citation *ic* in paper *cp* that is not already in the context p.

The closure computed for the direct citations in Table 2.3 appears in Table 2.4 and this is the context represented by Figure 2.8. While all of the survey papers used to compute the citation closure were included in the context they have been excluded from the attribute sets in Tables 2.3 and 2.4 to increase table readability.

Although the closure context was computed automatically the direct citation context used as input to the closure algorithm was constructed by hand. There is, however, an alternative – the automatically generated citation counts available via ResearchIndex [146].

The ResearchIndex Digital Library (CiteSeer)

ResearchIndex is a scientific digital library project that was originally a demonstration site for the NEC Research Institute's CiteSeer software. CiteSeer was designed to automatically gather and index citations from papers published on the World Wide Web (WWW). Computer Science literature was chosen as the domain for the project and the library now claims to be "Earth's largest free full-text index of scientific literature". Over 530,000 documents written by more than 602,300 authors are indexed by the system.

Citation indexing links articles based on a bibliography of cited articles or references within one paper being matched against the titles of other documents within the database. This process is automated in ResearchIndex and both papers cited within a document and papers citing a document can be retrieved. This form of document linking also allows research trends over time to be investigated. Papers cited by a document reflect the time before publication while papers citing a particular document represent the time after publication.

The ResearchIndex digital library is constructed autonomously and papers are being continually added to the database. Search results from Web search engines containing terms like "papers", "publications" and "postscript" are used to find potential papers for indexing. Papers in Postscript and PDF format are downloaded from the Web and converted to text. To check if the document is a research publication a search of the text is made for

Inputs

Let $\mathbb{K} := (G, M, I)$ be a context containing direct paper citations

Variables

Let $\mathbb{K}_{old} := (\varnothing, \varnothing, \varnothing)$ be an initially empty context

Let p be a survey paper

Let cp be a cited paper

Let ic be an indirect citation

Outputs

Let $\mathbb{K}$ be an updated context containing the closure of paper citations

Algorithm

WHILE $\mathbb{K} \neq \mathbb{K}_{old}$ **LOOP**
  $\mathbb{K}_{old} := \mathbb{K}$
  FOR each $p \in G$ **LOOP**
    FOR each $cp \in M$ **LOOP**
      IF $(p, cp) \in I$ **THEN**
        FOR each $ic \in M$ **LOOP**
          IF $(cp, ic) \in I$ and $(p, ic) \notin I$ **THEN**
            $I := I \cup (p, ic)$
          END IF
        END LOOP
      END IF
    END LOOP
  END LOOP
END LOOP

Figure 2.9: Algorithm to compute the citation closure within the set of survey papers.

	Ball99concept	Boettger01reconciling	Canfora99case	Dekel02applications	Duwel98identifying	Duwel00bridging	Funk95algorithms	Godin93building	Godin95applying	Godin98design	Krone94inference	Kuipers00types	Lindig95concept	Lindig97assessing	Sahraoui97applying	Siff97identifying	Snelting96reengineering	Snelting98concept	Snelting98reengineering	Snelting99reengineering	Snelting00software	Snelting00understanding	Tonella01concept	vanDeursen98identifying
Ammons03debugging																								
Andelfinger97diskursive																								
Arevalo03understanding-a							×	×	×	×	×		×	×	×	×	×	×	×					
Arevalo03understanding-b				×			×	×	×	×	×		×	×	×	×	×	×	×					
Ball99concept							×	×	×		×		×	×	×	×	×	×	×					
Boettger01reconciling																								
Bojic00reverse							×	×	×		×		×	×	×	×	×							
Canfora99case							×	×	×		×		×	×	×	×	×	×	×					×
Dekel02applications							×	×	×		×		×	×	×	×	×	×	×					
Duwel98identifying							×				×		×	×			×							
Duwel99enhancing							×				×		×	×			×							
Duwel00bridging					×		×	×	×	×	×		×	×	×	×	×	×	×					
Eisenbarth01aiding	×		×				×	×	×	×	×	×	×	×	×	×	×	×	×	×	×			×
Eisenbarth01feature	×		×				×	×	×	×	×	×	×	×	×	×	×	×	×	×	×			×
Eisenbarth03locating	×		×				×	×	×	×	×	×	×	×	×	×	×	×	×	×	×		×	×
Fischer98specification							×	×	×		×		×	×	×	×	×	×	×					
Funk95algorithms											×		×				×							
Godin93building																								
Godin95applying								×																
Godin98design								×	×		×													
Huchard99from								×																
Huchard02when	×						×	×	×	×	×		×	×	×	×	×	×	×			×		
Krone94inference																								
Kuipers00types	×						×	×	×	×	×		×	×	×	×	×	×	×	×	×			×
Leblanc99environment								×																
Lindig95concept											×													
Lindig97assessing							×				×		×				×							
Richards02assisting		×																						
Richards02controlled																								
Richards02recocase		×																						
Richards02representing																								
Richards02using	×	×					×	×	×	×	×		×	×	×	×	×	×	×	×	×			×
Sahraoui97applying							×	×	×		×		×	×	×	×	×							
Schupp02right																								
Siff97identifying							×	×	×		×		×	×	×	×	×							
Snelting96reengineering											×		×											
Snelting98concept							×	×	×		×		×	×	×	×	×							
Snelting98reengineering							×	×	×		×		×	×	×	×	×	×						
Snelting99reengineering							×	×	×	×	×		×	×	×	×	×	×	×					
Snelting00software	×						×	×	×	×	×		×	×	×	×	×	×	×	×				×
Snelting00understanding	×						×	×	×	×	×		×	×	×	×	×	×	×					
Streckenbach99understanding							×	×	×	×	×		×	×	×	×	×	×	×	×				
Tilley03software					×	×	×	×	×	×	×		×	×	×	×	×	×	×					
Tilley03towards																								
Tonella99object							×	×	×		×		×	×	×	×	×							
Tonella01concept							×	×	×	×	×		×	×	×	×	×	×	×	×				
vanDeursen98identifying							×	×	×		×		×	×	×	×	×	×	×					

Table 2.4: Formal context representing closure of citations within the set of survey papers. Note that uncited papers have been excluded from the attribute set to increase table readability.

a bibliography or reference section. The citations in the bibliography are then exploited for document linking using a process known as Autonomous Citation Indexing (ACI) [81]. ACI works by automatically matching the citations contained in the bibliography section to the titles of other documents in the ResearchIndex database. There are many different citation styles and the system must be able to recognise variations that refer to the same document.

ResearchIndex also classifies some documents as being either *authorities* or *hubs*. Authorities are documents that are considered to be authoritative because they are highly cited by other papers. Hubs are documents, typically survey papers, that cite a large number of authorities in a particular area. They provide a good point of introduction to the literature in a particular field. With respect to Figure 2.8 authorities would appear as attributes at the top of a long chain because they are highly cited, while hubs are likely to appear as objects at the bottom of long chains because they contain many citations.

2.7 Comparing Paper Impact via Citation Count

Table 2.5 presents a summary of citation counts from ResearchIndex for the 47 survey papers in July, 2003. The table also includes the corresponding count for the number of direct citations from the context in Table 2.3 and the citation closure count from Table 2.4. The papers are sorted in descending order of ResearchIndex count and then alphabetically where two or more papers have the same count.

The table contains a number of interesting features. First, it can be seen that the count of citations under closure on its own is not a meaningful measure. For example, *Funk95algorithms* only contains 3 and 4 citations for the ResearchIndex and direct citation counts. However, by virtue of the fact that the paper was published in 1995 it has a count of 29 citations under closure.

It is also interesting to note that although *Fischer98specification* appears to have been ignored within the set of survey papers it has been cited 10 times by papers in the ResearchIndex database. Conversely, the case also arises where there are direct citations within the survey collection but none recorded by ResearchIndex. For example,

Paper	ResearchIndex Citations	Direct Citations	Closure Citations
Lindig97assessing	29	21	28
Krone94inference	27	16	33
Siff97identifying	27	17	26
Snelting98reengineering	24	17	20
Snelting99reengineering	24	3	8
Godin93building	23	10	30
Snelting96reengineering	20	15	30
vanDeursen98identifying	17	6	7
Snelting98concept	16	8	21
Lindig95concept	12	3	31
Ball99concept	11	3	8
Fischer98specification	10	0	0
Sahraoui97applying	8	5	26
Snelting00software	5	2	5
Godin95applying	4	2	27
Godin98design	4	5	15
Canfora99case	3	3	3
Funk95algorithms	3	4	29
Kuipers00types	3	3	3
Eisenbarth01aiding	2	0	0
Tonella99object	2	1	1
Eisenbarth01feature	1	0	0
Huchard99from	1	0	0
Richards02controlled	1	0	0
Snelting00understanding	1	1	1
Ammons03debugging	0	0	0
Andelfinger97diskursive	0	0	0
Arevalo03understanding-a	0	0	0
Arevalo03understanding-b	0	0	0
Boettger01reconciling	0	3	3
Bojic00reverse	0	0	0
Dekel02applications	0	1	1
Duwel00bridging	0	1	1
Duwel98identifying	0	1	2
Duwel99enhancing	0	0	0
Eisenbarth03locating	0	0	0
Huchard02when	0	0	0
Leblanc99environment	0	0	0
Richards02assisting	0	0	0
Richards02recocase	0	0	0
Richards02representing	0	0	0
Richards02using	0	0	0
Schupp02right	0	0	0
Streckenbach99understanding	0	0	0
Tilley03software	0	0	0
Tilley03towards	0	0	0
Tonella01concept	0	0	0

Table 2.5: For each of the survey papers this table shows the number of citations reported by ResearchIndex, the number of direct citations within the set of papers and the total number of citations after computing the citation closure.

Boettger01reconciling contains 3 direct citations but none within ResearchIndex. There are a number of possible explanations for these values. First, ResearchIndex relies on search engine queries to locate papers and the paper or papers that cite it may not be visible. This is one of the limitations of autonomous citation indexing — not all papers are available on-line and some may be hidden behind services that require a subscription for full-text access. Alternatively, it could also be the case that the paper is in the ResearchIndex database but the referencing papers are not, or they may be ignored because they are self citations.

As might be expected, for the 10 most highly cited papers according to ResearchIndex there is a reasonable correspondence with the ordering based on direct citations. However, a major difference appears in the values for *Snelting99reengineering*. While this appears to be a case where ResearchIndex has confused the papers *Snelting98reengineering* and *Snelting99reengineering* this is also a common occurrence within the literature. A number of preliminary and expanded versions of Snelting and Tip's "Reengineering Class Hierarchies Using Concept Analysis" paper have been published and there is some inconsistency in terms not only of citation details but also for the year of publication. Ignoring this inconsistency, it can be seen that 6 of the top 10 most highly cited papers are either authored or co-authored by Snelting. These 6 papers: *Lindig97assessing*, *Krone94inference*, *Snelting98reengineering*, *Snelting99reengineering*, *Snelting96reengineering*, and *Snelting98concept* all appear along the main "trunk" of the closure diagram in Figure 2.8.

While the citation closure lattice provides a useful view of the survey literature, a citation count based on closure is meaningless. Alternatively, there may be other measures based on chain length or the number of papers in the extent that may also provide some insight into a paper's impact.

The ResearchIndex digital library represents an obvious path to automate the construction of a citation context, however, there are two immediate impediments. First, initial experiments indicate that the HTML returned by ResearchIndex queries does not validate using standard HTML tools so a custom parser would be required. While this is only a minor technical challenge the second impediment is ResearchIndex's "no robots"

policy which an automated approach may violate [145].

2.8 Conclusion

This chapter has presented a literature survey of 47 academic papers reporting software engineering applications for FCA. The first half of the chapter provided an overview of the papers using what are essentially conceptual scales based on ISO12207 categorisation, target application language and target application size.

The majority of the reported work has been in the areas of detailed design and software maintenance where FCA has been applied to object-oriented re-engineering and class identification tasks. While these late-phase approaches could be seen as obvious applications because of the specialisation/generalisation relationship between the concepts in a concept lattice, the range of different formal objects and attributes used is surprising. These range from documentation, use-cases and compiled code through to execution traces. Other novel applications have included support for test-coverage analysis. While a number of papers describe some early-phase approaches there are still a number of as yet untouched application areas including *acceptance support, integration, coding, installation* and *qualification testing.*

The second half of the chapter introduced two further analyses of the literature which provided an insight into authorship groups, citation patterns and the impact of papers within the collection of survey papers. Eight main research groups are identified among the authors and of particular note is the impact of Snelting's work. This is most clearly seen in the application language, author collaboration and citation closure diagrams.

While the attributes analysed in the early sections of the paper were specific to software engineering the collaboration and citation diagrams relied on attributes which are common to all academic literature. Although unpublished, a web-page by Kalfoglou [113] reports similar work to classify journal papers using two different classification schemes as well as an example exploring the changing membership of a conference programme committee.

Having demonstrated that the majority of applications support late-phase activities — and more specifically software maintenance — the remainder of the thesis explores FCA

applications to early-phase activities and in particular applications to formal specification. The next chapter presents a case study applying Düwel's approach for identifying class hierarchies for a system specified using the Object-Z formal specification language.

Chapter 3

Class Hierarchy Identification from Use-case Descriptions

As discussed in Chapter 2, one of the existing application areas for FCA in software engineering is the identification and maintenance of class hierarchies for both new and legacy applications. This chapter describes an exercise in object-oriented software modelling where FCA is applied to a formal specification case study using Object-Z. In particular, the informal description from the case study is treated as a set of use-cases from which candidate classes and objects are derived. The resulting class structure is contrasted with the existing Object-Z design and the two approaches are discussed. The work presented in this chapter has been published in a paper co-authored with Wolfgang Hesse and Roger Duke [201]. Hesse provided the methodology, instruction, and insight into the approach while Duke provided the example case study and editorial assistance.

3.1 Motivation

The aim of this work was to perform a comparison between a class hierarchy derived via the application of FCA and an existing class diagram produced as part of an Object-Z case study [52]—in particular applying FCA in connection with use-case analysis to discover class candidates [59]. Moreover, the FCA class decomposition was performed sight unseen, that is, only the use-cases were presented to the class designers — they did not have access to an existing class diagram for the system being modelled. The informal description of

the system was considered as a use-case source and five use-cases were identified. With respect to the process a number of questions were asked:

- What are the differences between the two class hierarchies and are there valid reasons for the differences?
- What support does FCA offer the class designer and to what extent is it, or can it be automated?
- How does the FCA approach influence the quality of the resulting class structure?
- Is FCA a useful mechanism for constructing Object-Z classes?

Currently, the "Object-Z engineer" works in a bottom-up manner, using mainly inheritance and association to create the system. The process is largely based on native experience, and a great deal of Object-Z "training" tries to cultivate this experience. Can FCA help by providing a method that relies less on training and previously acquired knowledge but results in an identical or at least a similar class structure?

The FCA-based methodology for identifying class candidates from a use-case-like problem description is based on the systems analysis work of Düwel and Hesse [59, 57]. Informally the approach can be described as follows:

- (Re-)Structure the problem description and formulate use-cases.
- Mark all relevant "things" occurring in the use-case descriptions.
- Build a *formal context* by taking the marked "things" as *objects* and the use-cases as *attributes*.
- Generate the formal concept lattice for discussion. Check the concept nodes of the resulting lattice for suitable class candidates.
- Discuss, rework and modify the use-case descriptions and the attribute associations of objects.
- Iterate the preceding steps until a satisfactory class structure has evolved.

The next section introduces this approach in more detail using a mass transit railway ticketing system as an example. Section 3.3 describes the progress from the initial informal description to the final concept lattice representing a possible class structure for the example case. Section 3.4 compares the results of the two approaches.

3.2 From an informal description to a first concept lattice

An informal description taken from a case study modelling a mass transit railway ticketing system in Object-Z was used as the starting point for the exercise [52]. The main purpose of the case study was to capture the functionality of the different ticket types. The functionality was specified as perceived by an observer of the railway system. The informal description of the system by Duke and Rose [52] reads as follows:

- The mass transit railway network consists of a set of stations. For simplicity, it will be assumed this set is fixed, i.e. stations are neither added to nor removed from the network.
- The fare for a trip depends only upon the stations where the passenger joins and leaves the network, i.e. the actual route taken by the passenger when in the network is irrelevant. The fare structure can be **updated** from time to time.
- Three types of tickets can be **purchased**:

  Single-trip tickets permit only a single trip, and only on the day the ticket is purchased. The ticket has a value in the range $1 to $10, and the passenger is permitted to leave the network if and only if the fare for the trip just completed does not exceed the ticket's value.

  Multi-trip tickets are valid for any number of trips provided the current value of the ticket remains greater than zero. A ticket's initial value is either $50 or $100. Each time the passenger leaves the network the value of the ticket is reduced by the fare for the trip just completed. If this fare exceeds the value remaining on the ticket, the passenger is still permitted to leave the network and the value of the ticket is set to zero. A multi-trip ticket expires after two years even if it has some remaining value.

  Season tickets are valid for either a week, a month, or a year. Within that period no restrictions whatsoever are placed upon the trips that

can be undertaken.

- As tickets are expensive to produce, they can be **reissued**, i.e. tickets can have their expiry date and value reset. (The type of ticket cannot be changed.) Although tickets are issued to passengers, the essential interaction is between tickets and stations; thus passengers are not modelled.

	reissue ticket	update fare structure	buy single	buy multi	buy season
type of ticket	×	×			
expiry date	×				
value	×		×	×	
fare struct		×			
station		×			
fare		×	×		
network		×	×	×	
passenger		×	×	×	
ticket	×		×	×	×
trip		×	×	×	×
day			×		
single trip			×		
initial value				×	
multi-trip				×	
number				×	
value remaining				×	
month					×
period					×
week					×
year				×	×

Table 3.1: First formal context created from the five use-cases. The corresponding concept lattice is shown in Figure 3.1.

From the informal description five use-cases were identified: *update fare structure*, *buy single ticket*, *buy multi-trip ticket*, *buy season ticket*, and *reissue ticket*. In the first step, the text was cut into five pieces according to bold keywords representing the five “use-cases”. All nouns showing a certain relevance were considered as “things” or objects in the FCA sense. The choice of the nouns was deliberately done in a syntactical, “quasi-automated” way, that is, without further semantic considerations as to whether this choice makes much sense. A formal context was constructed using the identified nouns as the set of objects

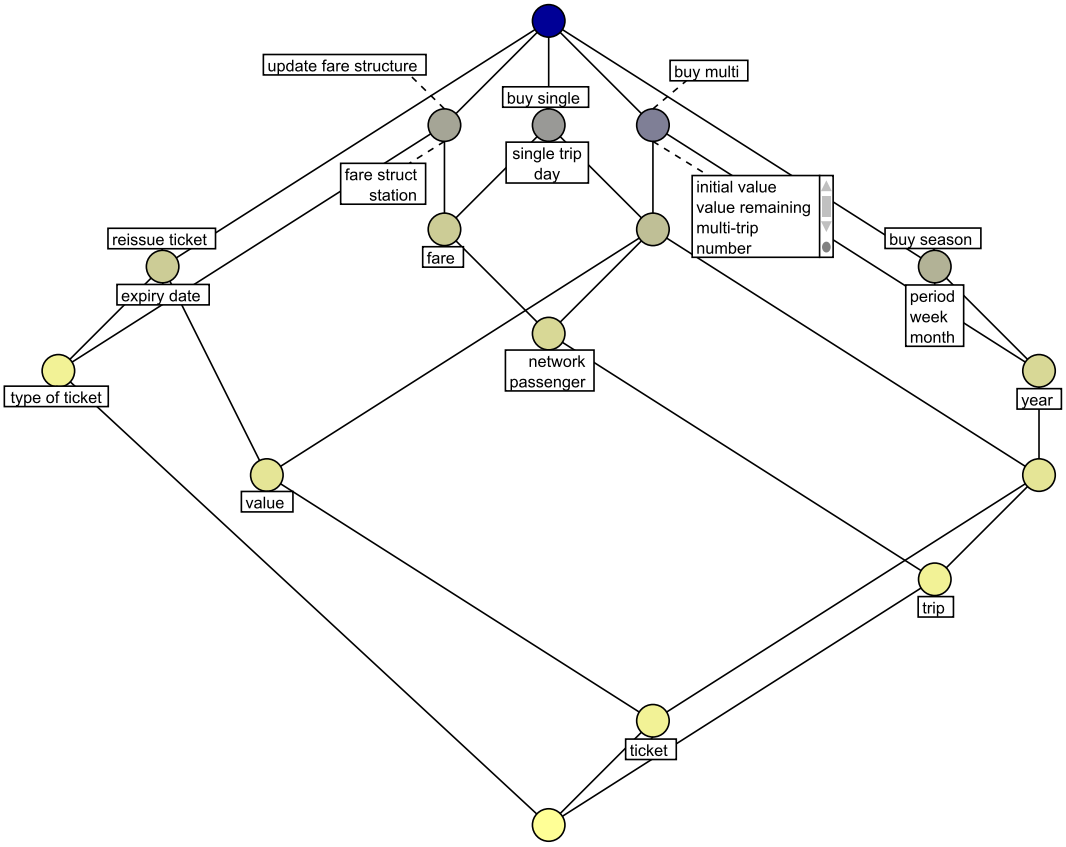


Figure 3.1: Concept lattice of the formal context abstracted from the cross table in Table 3.1.

and the use-cases as the attributes[1]. The result appears in Table 3.1. Here an '×' at the intersection of a use-case column and a noun row indicates that the noun was identified in this use-case description. The corresponding formal concept lattice is shown in Figure 3.1.

A first correction concerns the cut of the text into use-cases where the headline introducing the three types of tickets was mistaken as a part of the *update fare structure* use-case. In fact, the noun *type of ticket* is not addressed in *update fare structure* but it is part of the introductory headline and thus applies to the following three use-cases describing the purchase of the three ticket types.

The initial changes between the context in Table 3.1 and Table 3.2 result from *type of ticket* being removed from the *update fare structure* use-case and added to the three "buy"

[1]To avoid confusion with the standard FCA terminology introduced in Chapter 1 the term "item", when referring to nouns, will be used throughout the chapter instead of "object".

	reissue ticket	update fare structure	buy single	buy multi	buy season
type of ticket	×		×	×	×
expiry date	×				
value	×		×	×	
fare struct		×			
station		×			
fare		×	×		
network		×	×	×	
passenger		×	×	×	
ticket	×		×	×	×
trip		×	×	×	×
day			×		
single trip			×		
initial value				×	
multi-trip				×	
number				×	
value remaining				×	
month					×
period					×
week					×
year				×	×
time		×			×

Table 3.2: Changes to the formal context from Table 3.1 are shown in grey. A new *time* object has been added and *type of ticket* adjusted. The corresponding concept lattice is shown in Figure 3.2.

use-cases: *buy single*, *buy multi* and *buy season*. The differences between the two contexts are shown in grey. Furthermore the item *time* was overlooked during construction of the first context and has been included here in both the *update fare structure* and *buy season* use-cases. Time is explicitly mentioned in *update fare structure* but not in *buy season ticket*. However, based on the implicit assumption in the wording of "Within that period . . . " the text can be extended to "Within that period of *time*. . . ".

3.3 Iterating the FCA steps

The initial steps of identifying objects and attributes within the informal description and the initial corrections to the incidence relation resulted in the formal context in Table 3.2 and the line diagram in Figure 3.2. A first analysis of the lattice shows the use-case ↔ noun

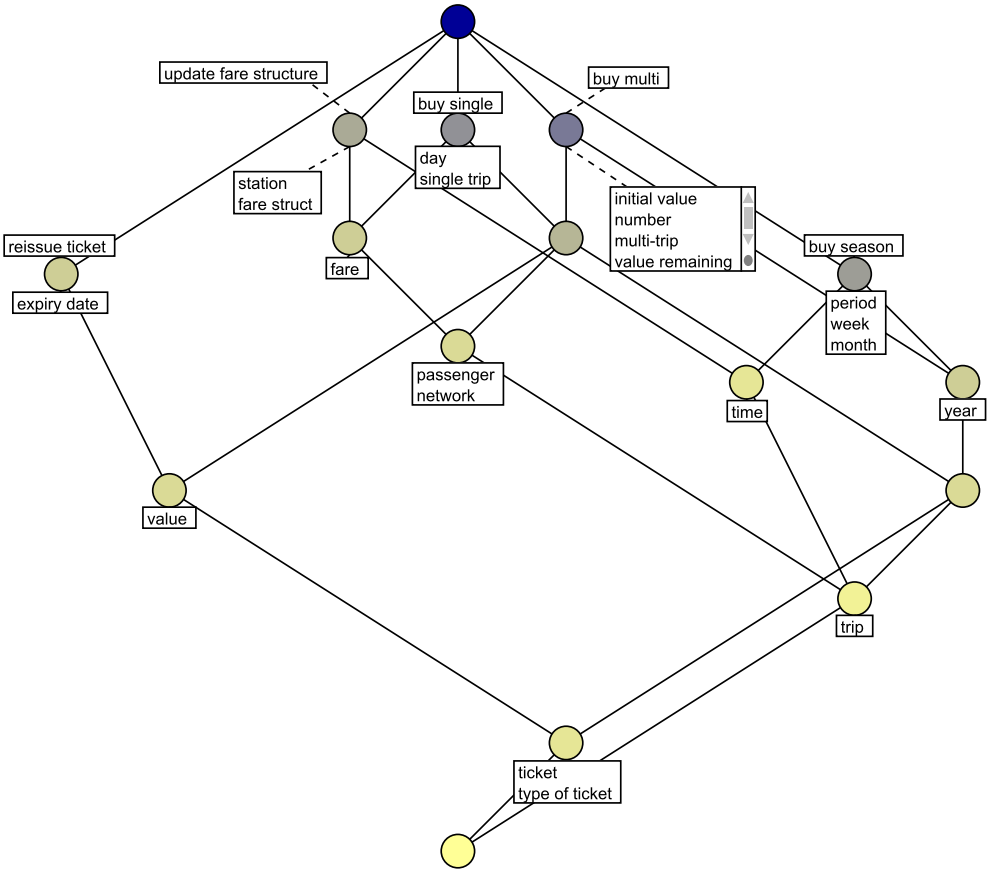


Figure 3.2: Concept lattice of the formal context abstracted from the cross table in Table 3.2. Observe that *time* has been introduced and *type of ticket* now also applies to the three ticket buying use-cases.

dependencies as far as they can be derived from the pure syntactic formulation of the use-cases:

1. If a node marked by a use-case label is selected then all the relevant nouns contained in the use-case can be found among the successor nodes.
2. If a node marked by a noun label is selected then all the use-cases containing the noun can be found among the predecessor nodes.

An immediate consequence is that the higher things occur in the lattice diagram then the more specialised they are—i.e. the lower-most things are used in more use-cases and are therefore the most general ones. A dual argument would also apply to the use-cases if these formed a hierarchy, however, in this example there is no use-case which appears above or below another use-case in the concept lattice.

	reissue ticket	update fare structure	buy single	buy multi	buy season
type of ticket	×		×	×	×
expiry date	×				
value	×		×	×	
fare struct		×			
station		×			
fare		×	×		
network		×	×	×	×
passenger		×	×	×	×
ticket	×		×	×	×
trip		×	×	×	×
day			×		
single trip			×		
initial value				×	
multi-trip				×	
number				×	
value remaining				×	
month					×
period					×
week					×
year				×	×
time		×			×

Table 3.3: Changes to the formal context from Table 3.2 are shown in grey. The *buy season* use-case has been adjusted and the corresponding concept lattice is shown in Figure 3.3.

The idea that things which appear higher in the diagram are more specialised appears to be counter-intuitive but results from the choice of nouns as objects and use-cases as attributes in the construction of the formal context. While it is possible to transpose the objects and attributes in FCA this choice was deliberate because the resulting diagram resembles the typical layer structure of many software architecture diagrams. The upper layers represent functional components while the lower ones represent common services often associated with data clusters. Considering the diagram from the supremum down represents system functionality and use-case refinement for larger examples. Considering the diagram from the infimum up corresponds to the data view where each upward step represents a refinement in the explanation of the data [94].

Further refinement of the structure now calls upon the “contextual knowledge” of the modeller/reviewer. From this point of view a first “semantic” analysis of the lattice can be

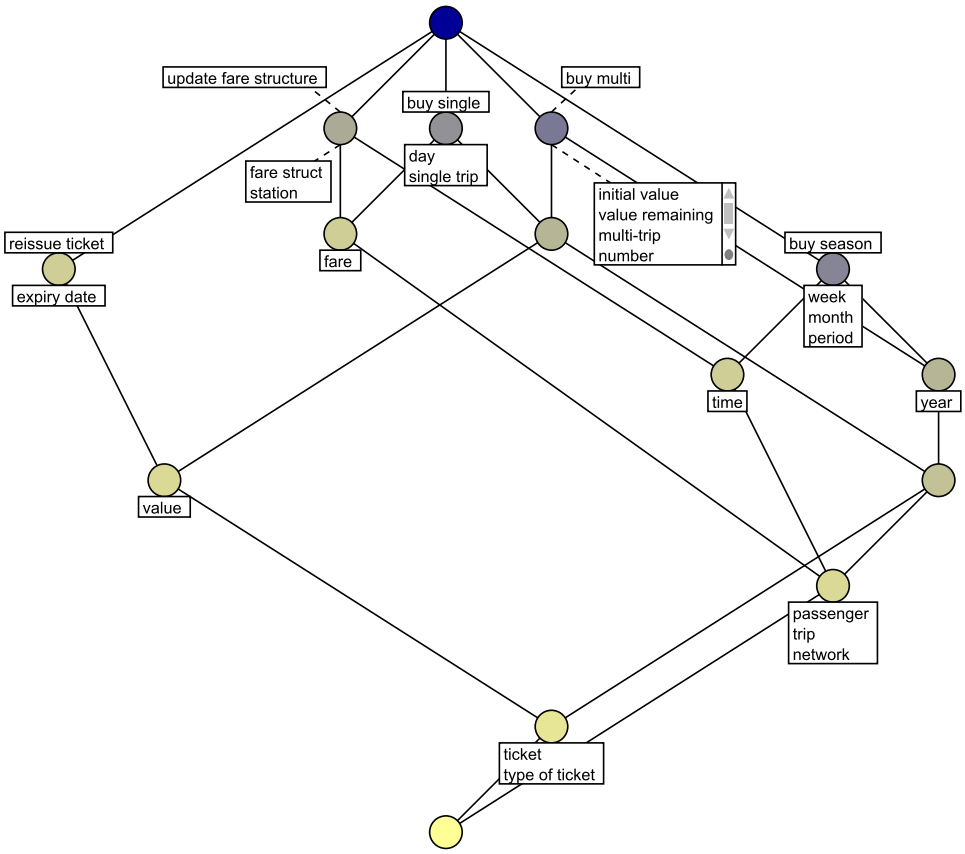


Figure 3.3: Concept lattice of the formal context abstracted from the cross table in Table 3.3. Note that *passenger* and *network* now also apply to the *buy season* use-case.

conducted. Firstly, from Figure 3.2 it can be seen that *passenger* and *network* appear to be too high in the diagram. These items apply to the *update fare structure*, *buy single* and *buy multi* use-cases but not to *buy season*. Given the obvious importance of both passengers and the railway network to a railway ticketing system these two items are less general than expected by the modellers. This inspires a review of the *buy season* use-case; although not mentioned in the corresponding use-case description, a season ticket implicitly involves both a *passenger* and the mass transit *network*. A more explicit version of the use-case description would read:

> **Season tickets** are valid *on the whole network* for either a week, a month, or a year. Within that period no restrictions whatsoever are placed upon the trips that can be undertaken *by the passenger*.

	reissue ticket	update fare structure	buy single	buy multi	buy season
type of ticket	×		×	×	×
expiry date	×				
value	×		×	×	
fare struct		×			
station		×			
fare		×	×		
network		×	×	×	×
passenger		×	×	×	×
ticket	×		×	×	×
trip		×	×	×	×
day			×		
single trip			×		
initial value				×	
multi-trip				×	
number				×	
value remaining				×	
month					×
period					×
week					×
year				×	×
time		×	×	×	×

Table 3.4: Changes to the formal context from Table 3.3 are shown in grey. The implicit *time* references have been added and the corresponding concept lattice is shown in Figure 3.4.

This modification is reflected in Table 3.3 and the corresponding lattice in Figure 3.3.

The context in Table 3.4 represents the recognition that the items *day* and *year* in *buy single* and *buy multi* respectively also imply *time*. Coincidentally, this also corrects an earlier mistake where the modellers had actually missed a reference to *time* in the *buy multi* use-case. The resulting concept lattice is depicted in Figure 3.4 and at this point in the exercise the modellers were shown the existing class diagram of the system for the first time. An initial informal comparison was made and these observations are presented in Section 3.4 of the paper.

Looking at Figure 3.4 it can be observed that while the item *time* has now moved into a lower and therefore more general position, *fare* still appears too high in the diagram. It is reasonable to assume that fares may also apply to some or all of the other ticket types,

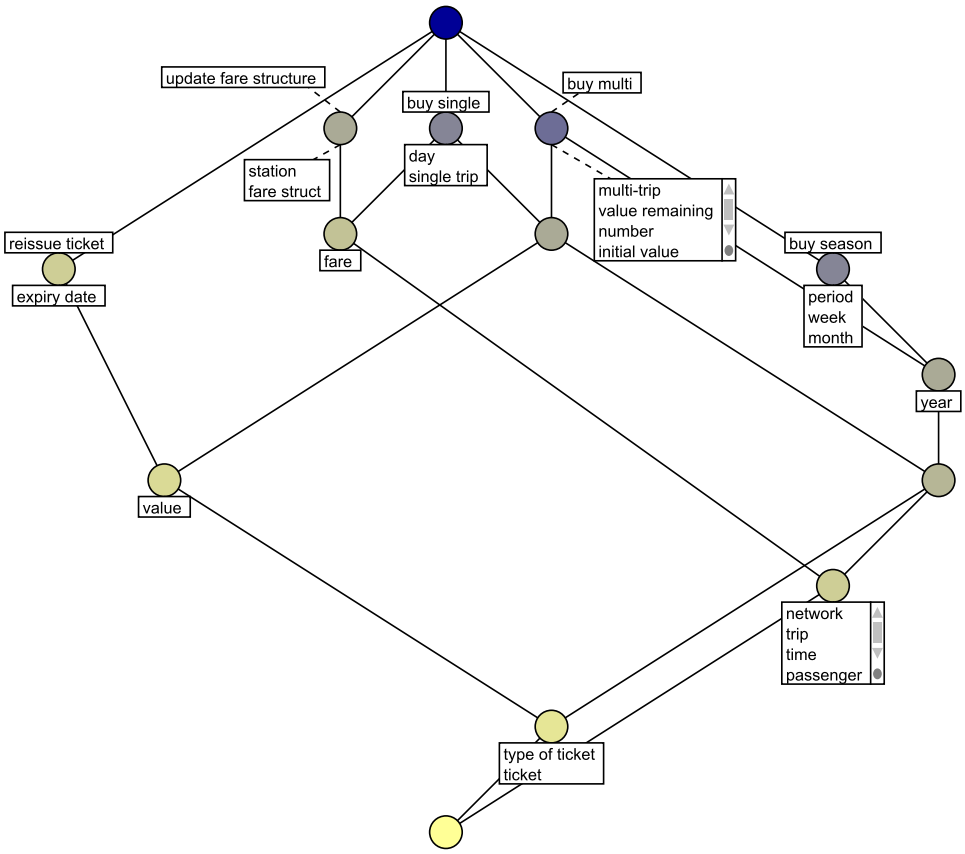


Figure 3.4: Concept lattice of the formal context abstracted from the cross table in Table 3.4. Note that *time* now applies to all three of the ticket buying use-cases.

however, in the diagram *fare* only applies to the *update fare structure* and *buy single* use-cases. This leads to the identification and correction of another mistake from the initial noun analysis of the use-cases; the item *fare* is mentioned in the *buy multi* use-case but was missed by the class modellers during the creation of the earlier contexts. The appropriate inclusion is reflected in Table 3.5. A text mining approach using a dictionary of terms or an ontology relevant to the domain being modelled may facilitate the automated extraction of nouns from use-cases and prevent these kinds of errors.

Furthermore, considering the *station* item, the calculation of a *fare* implies knowledge of the stations by which a passenger enters and exits the mass transit railway network. Both the *buy single* and *buy multi* use-cases include the *fare* item so in Table 3.5 the *station* item has been included for these use-cases as well. The concept lattice resulting from these

	reissue ticket	update fare structure	buy single	buy multi	buy season
type of ticket	×		×	×	×
expiry date	×				
value	×		×	×	
fare struct		×			
station		×	×	×	
fare		×	×	×	
network		×	×	×	×
passenger		×	×	×	×
ticket	×		×	×	×
trip		×	×	×	×
day			×		
single trip			×		
initial value				×	
multi-trip				×	
number				×	
value remaining				×	
month					×
period					×
week					×
year				×	×
time		×	×	×	×

Table 3.5: Changes to the formal context from Table 3.4 are shown in grey. The missed *fare* and implicit *station* information has been corrected. The corresponding concept lattice is shown in Figure 3.5.

"semantic implications" is depicted in Figure 3.5.

3.4 Comparing the two approaches

The aim of this modelling exercise was to perform a comparison between a class hierarchy derived via the application of FCA and an existing class diagram produced as part of an Object-Z case study. Having derived the lattice depicted in Figure 3.4 the modellers were shown the existing class diagram for the first time. One further refinement was made resulting in Figure 3.5. This section compares and contrasts the "final" lattice with the existing class diagram shown in Figure 3.6.

An examination of the up-set for the unlabelled node below the "year" item in Figure 3.5 reveals the apparent similarity between the line and class diagrams. In Figure 3.7 this order filter and the corresponding order ideal are shown in bold. The nodes labelled *buy single*,

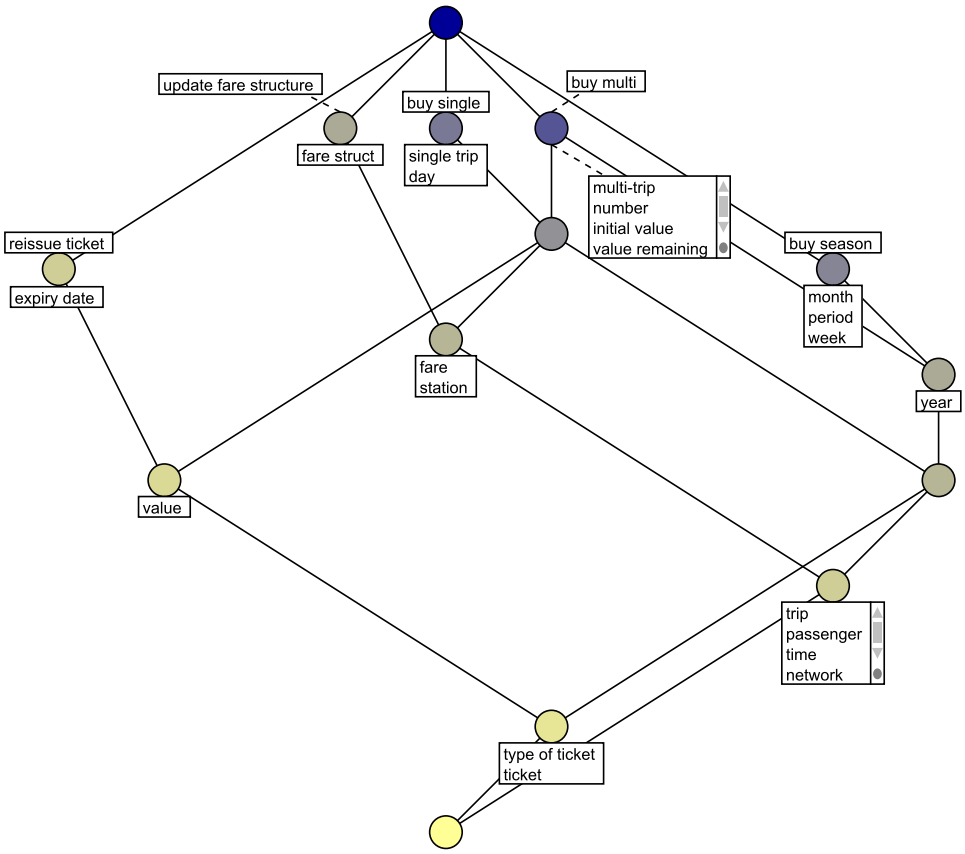


Figure 3.5: Concept lattice of the formal context abstracted from the cross table in Table 3.5. Note that *fare* and *station* now apply to both the *buy single* and *buy multi* use-cases.

buy multi, and *buy season* in Figure 3.7 represent the class candidates corresponding to the classes *SingleTripTicket*, *MultiTripTicket*, and *SeasonTicket* in Figure 3.6. Similarly, the nodes labelled “TripTicket” and “Ticket” in Figure 3.7 correspond with the *TripTicket* and *Ticket* class unions. In this case, the structure shown in bold represents “ticket buying” functionality. An alternative interpretation that considers the re-issuing of tickets would move the corresponding “Ticket” label down to the node containing the *type of ticket* and *ticket* items.

An obvious difference between the two structures is the presence of attributes and possible methods (e.g. *update fare structure*) in the line diagram as compared with Object-Z functions in the class diagram. However, the relationship between the two structures can still be inferred by checking if the attributes required for a particular function are in

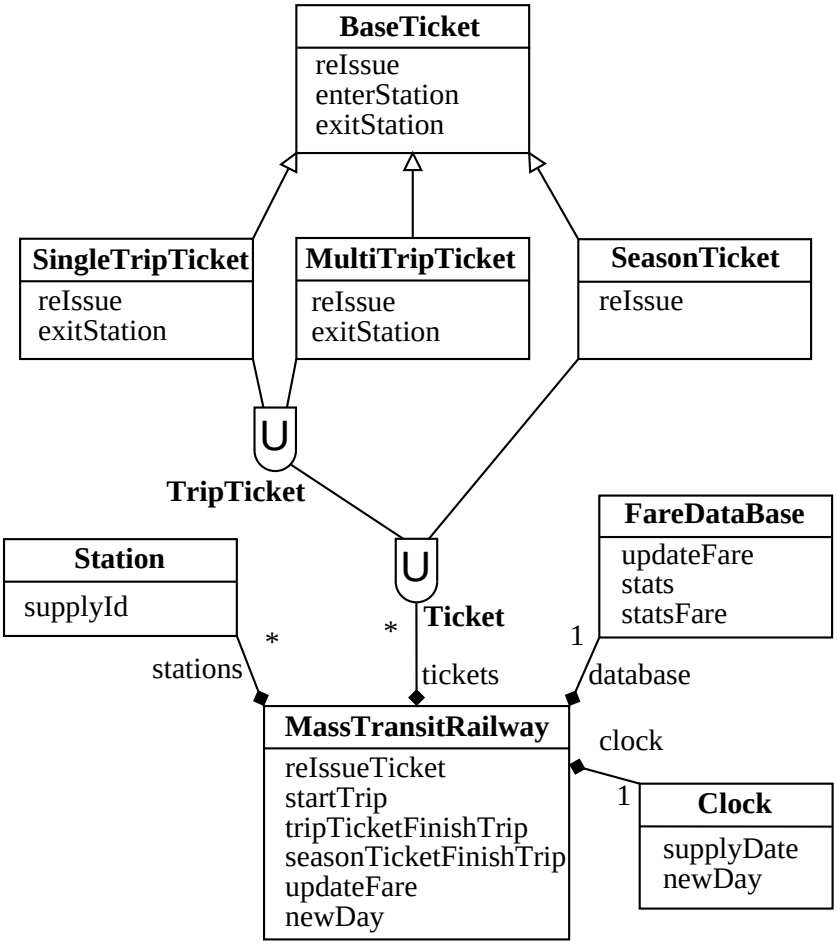


Figure 3.6: Object-Z class diagram for the mass transit railway system. This diagram appears as Figure 9.8 in the original case study [52].

the "correct" place. For example, the Object-Z representation makes use of *EnterStation* and *ExitStation* functions so that the appropriate fare can be calculated and checked for SingleTrip and MultiTrip tickets. The action of entering and exiting stations is assumed domain knowledge and is therefore not present in the use-cases. While the actions themselves do not appear in the line diagram the lattice mirrors the required structure because the *station*, *fare*, and *value* attributes are only available to these ticket types. This information is not required for a SeasonTicket.

Other differences between the two structures include the absence of obvious *Station*, *Clock* and *FareDataBase* classes in Figure 3.5. In addition, there are no references to statistics in the original use-cases and a comparison shows that the *stats* and *statsFare*

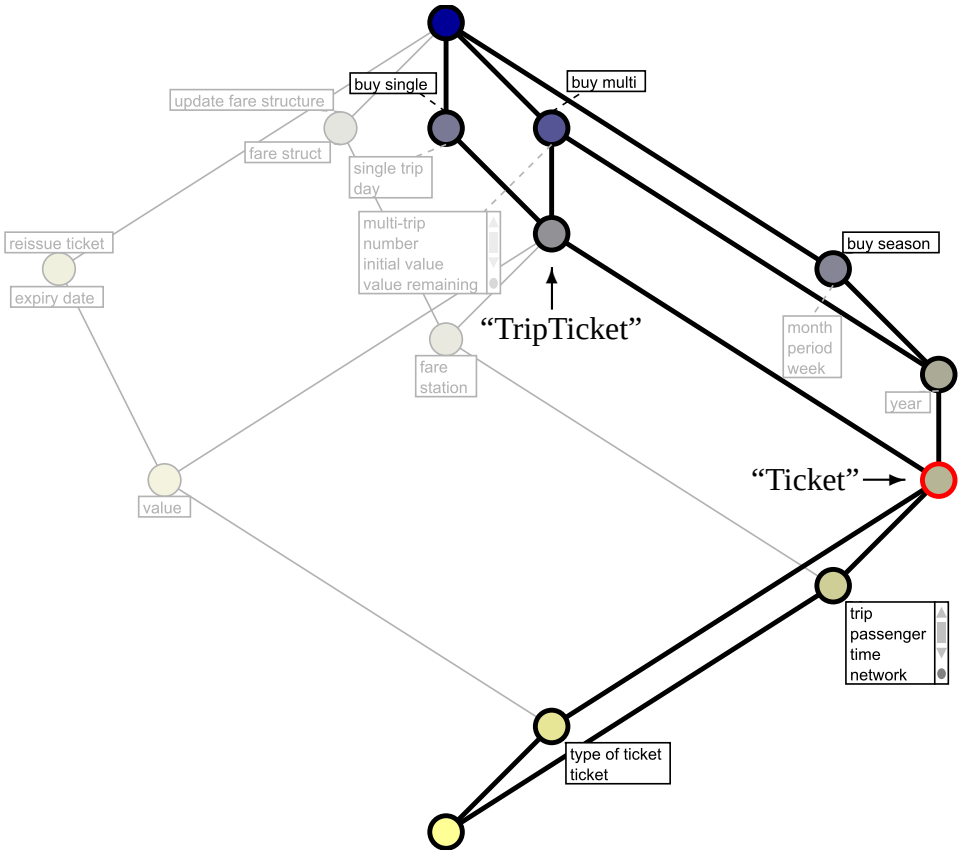


Figure 3.7: The Formal Concept lattice from Figure 3.5 with the ticket class hierarchy shown in bold. The nodes labelled "TripTicket" and "Ticket" correspond respectively with the *TripTicket* and *Ticket* class unions in Figure 3.6.

functions in Figure 3.6 are therefore quite artificial. These differences are largely due to functional artefacts or abstractions required for the Object-Z specification of the mass transit railway. As Duke and Rose state:

> The main purpose of this case study is to capture the functionality of the different ticket types. The approach taken is to specify ticket functionality from the point of view of the passenger, i.e. as perceived by an observer of the railway system. In order to do this, however, it is necessary to conceptualise and abstract various other objects in the system, such as the stations, a database to record the fare structure, and a clock to keep track of the days [52].

A further important question concerns the modularisation of the system, that is, its

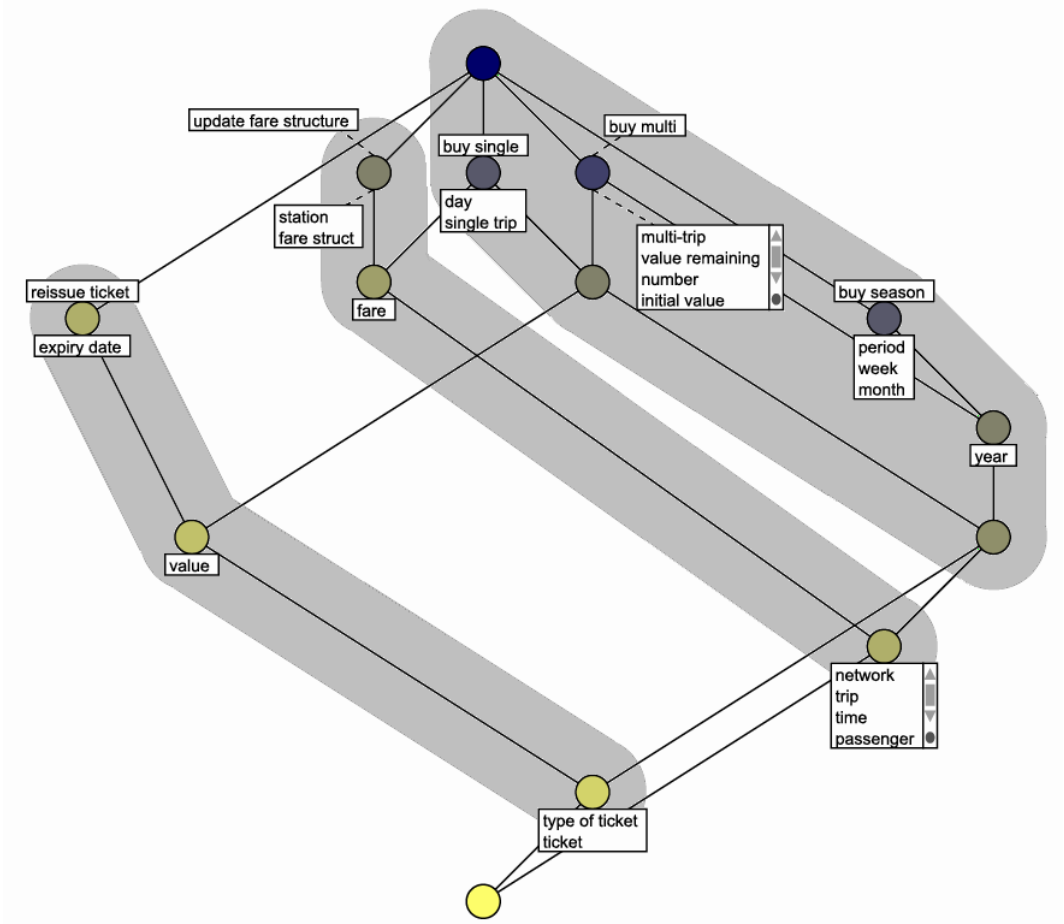


Figure 3.8: Initial package structure based on Figure 3.4.

decomposition into smaller units typically called *components*, *packages* or *modules*. Lindig and Snelting have shown that FCA can support this decomposition by forming so-called *block relations* [132]. Block relations result from filling up a formal context table with additional marks (not contained in the original context) in order to coarsen the lattice structure and obtain more compact concepts. In this case, the attempt to find an appropriate decomposition for the lattice of Figure 3.4 resulted in the initial package structure depicted in Figure 3.8. Three possible packages deal with the purchase of (various kinds of) tickets; the fare structure and its updates; and the re-issuing of tickets.

3.5 Object Exploration

While there are obvious similarities between the resulting line diagram and the original class diagram for this particular example, the approach described here has limitations. Firstly, how does a modeller know when to stop iterating? Statements like “in the correct position” still infer a reliance on the intuition, native experience and training of the modeller. A second question posed at the start of the chapter was “What support does FCA offer the class designer and to what extent is it, or can it be automated?”. The current iterative process relies on the development stakeholders agreeing that no more changes are necessary. While the domain knowledge of the stakeholders will always be required to “verify” that the lattice makes sense there is a way to formalise the checking process that also exploits this background knowledge.

In the example presented here iterations during the analysis result from questions about the “correct” position of objects and attributes in the line diagram. As a consequence of discussion between the modellers and system stakeholders the formal context is modified and the new line diagram examined. This could be seen as an *ad hoc* and informal form of object exploration which was introduced in Section 1.5.7. In the existing approach a perceived contradiction in the lattice results in a context update and another iteration of the process. An alternative would be to formally apply object exploration. Rather than relying on an inconsistency in the lattice being “noticed” the formalised process would instead presents a series of implications to the system stakeholders for discussion. Iterations can then stop when all the valid implications in the context have been explored. An obvious limitation of this approach, however, is the issue of scalability. The number of implications that need to be considered and discussed during object exploration may become unmanageable for even medium-size examples.

3.6 Conclusion

This chapter has presented a modelling exercise to identify class candidates using use-case analysis and FCA. An iterative correction process resulted in a final line diagram which

was then contrasted against a known existing structure. A small, well understood example was chosen and a comparison of the resulting structures demonstrates that they are quite similar. Obvious differences between the two structures rely on information that is not made explicit in the use-cases or they represent artificial constructions related to the specification in Object-Z.

Although it may be possible to automate the initial noun identification within use-cases, later refinements rely on the insight and judgement of those involved in the modelling process. This includes modellers, domain experts and other system stakeholders. The process can be further mechanised by applying object exploration.

The value of this approach then is in the process itself—the construction and discussion of the line diagrams, and in the kinds of questions it forces the designer to ask about the domain structure. The process and resulting diagrams also promote discussion as modellers consider and question the position of attributes in the line diagram and try to adjust the formal context accordingly. This is also consistent with the experiences reported by Andelfinger [6] where FCA was used to facilitate discussions during requirements analysis.

In this chapter FCA was used to support the derivation of an alternative class structure to the existing Object-Z hierarchy. Chapter 4 explores the application of FCA to the specifications themselves as a mechanism for visualising the structure and properties of formal specifications.

Chapter 4

Formal Specification Navigation and Visualisation

This Chapter introduces an approach for navigating and visualising Z specifications using FCA. The approach takes a source specification written in LaTeX and produces a formal context representing the static structure of the specification. A number of line diagrams can then be produced which allow a user to investigate and explore various properties of the specification. The line diagram does not replace, but is intended to be used in conjunction with, the original Z specification. Abstraction through conceptual scaling, nesting, zooming and folding line diagrams allows users to retain context while navigating large specifications and an example based on the *BirthdayBook* specification is presented. A summary of this work has been published in *Tilley03survey* [200] and a more detailed version in *Tilley03towards* [199].

Section 4.1 discusses existing approaches to visualising Z and related languages. These approaches typically focus on mappings to UML or the integration of UML-like graphical notations with Z. Section 4.2 describes an approach to creating formal contexts from Z specifications and the *BirthdayBook* specification is used as an illustrative example. The abstractions afforded by FCA are introduced in Section 4.3. The application of the abstractions to visualising and navigating Z specifications is presented throughout the remainder of the chapter.

4.1 Visualising Z Specifications

In an attempt to dispel the myths that "formal methods require highly trained mathematics" and that "formal methods are unacceptable to users" [91] there have been a number of approaches to provide alternative visual representations of specifications for Z-like languages. Typically, these representations have both textual and graphical components within their notation. Object-Z, for example, incorporates a number of UML-like graphical elements. One of the commonly stated reasons for the poor adoption of formal methods is that they are difficult to use and understand. These graphical notations typically aim to provide an abstraction over the mathematics to make the formal notations easier to understand.

UML [22] has become the de-facto industry language for modelling systems. While it provides a means of system specification it does not have the mathematical rigor of formal methods. However, UML enjoys a popularity that formal methods do not and its graphical nature makes it an obvious choice as an alternative representation language. UML is also implementation oriented which may be helpful to the ultimate implementers of a particular model, however, this is ultimately inconsistent with the aims of a conceptual modelling language [107].

There have been a number of approaches used to introduce graphical representations of Z specifications via UML. The work of Sun, Dong, Liu and Wang [195] provides an XML [219] representation for the Z family of languages called ZML. ZML can be transformed into UML and the mark-up is discussed further in Section 5.1.3. Many of the approaches focus on the structural aspects of the specification [224]. Kim and Carrington [117] argue that beyond the static structure of the specification the dynamic nature and complex constraints must also be visualised for a full understanding of a specification. To accomplish this they introduce two other graphical representations in addition to UML — one for the complex constraints and another for the operation schemas.

UML is an object-oriented notation and while Kim and Carrington's approach focuses on Z the use of UML as a graphical notation for formal specification typically focuses on application to Object-Z. Other work by Kim and Carrington [114, 115], Evans and

Clark [67], and Miao, Liu and Li [136] also combines Z and UML, however, rather than visualising Z specifications via UML these approaches focus on providing a formal basis for various aspects of UML in Z.

"Alloy" is a Z-related, lightweight formal method with both textual and graphical components that offers a straightforward mapping from UML into a formal notation [106, 105, 107]. Lightweight formal methods are "lightweight" in that they offer "less than completely formal" or partial approaches to specification, validation and testing [2, 106]. Typically they trade off completeness or language functionality for efficiency. Alloy is discussed further in Section 5.1.2.

With regard to the graphical elements in Object-Z, Duke and Rose [52] note that graphical notations are useful for presenting material to educate users and to facilitate communication between stakeholders in the development of a system. However, graphical notation should be seen as a complement to the specification. They also point out that graphical notations are descriptive and semi-formal at best. They are not appropriate for formal procedures which are conducted using mathematics. This view is consistent with the use of FCA to visualise Z specifications. The aim is not to provide an alternative to the specifications themselves but rather a representation that can be used alongside, to help navigate and provide insight into, the original specification. The first step towards the visualisation of a Z specification via FCA is to create a formal context from the specification.

4.2 From a Specification to a Formal Context

Given the mathematical nature of the Z notation, specifications are typically written in LaTeX mark-up which is then translated into a rendered form such as PostScript or PDF. Z representation issues and a number of alternative notations are discussed in Section 5.1. By considering the schemas as a set of objects and the mark-up elements of the specification "source" as attributes it is possible to create a formal context from which line diagrams can be constructed. An example context generated from the *BirthdayBook* specification in Oz style LaTeX [118] is presented in Table 4.1.

	known	. .	\power	NAME	birthday	\pfun	DATE	\ST	=	\dom	BirthdayBook	\emptyset	\Delta	name?	date?	\nem	birthday'	\union	(	\map	)	\Xi	date!	\mem	(	)	today?	cards!	n	\cbar	result!	REPORT	ok	already_known	not_known	\land	\lor
BirthdayBook	×	×	×	×	×	×	×	×	×	×																										
InitBirthdayBook	×							×	×		×	×																								
AddBirthday	×	×		×	×		×	×	×		×		×	×	×	×	×	×	×	×	×															
FindBirthday	×	×		×	×		×	×	×		×			×								×	×	×	×	×										
Remind	×	×	×	×	×		×	×	×		×								×		×	×			×	×	×	×	×	×						
Success		×						×	×																					×	×	×				
AlreadyKnown	×	×		×				×	×		×			×								×		×						×	×		×			
NotKnown	×	×		×				×	×		×			×		×						×								×	×			×		
RAddBirthday	×	×		×	×		×	×	×		×		×	×	×	×	×	×	×	×	×			×	×	×				×	×	×	×		×	×
RFindBirthday	×	×		×	×		×	×	×		×			×		×						×	×	×	×	×				×	×	×		×	×	×
RRemind	×	×	×	×	×		×	×	×		×								×		×	×			×	×	×	×	×	×	×	×			×	

Table 4.1: Formal context for the *BirthdayBook* specification.

```
\begin{schema}{Success}
    result! : REPORT
\ST
    result! = ok
\end{schema}
```

Figure 4.1: LaTeX mark-up for the *Success* schema in Oz style.

For example the Success schema shown in Figure 4.1 contains the mark-up elements `:`, `\ST`, `=`, `result!`, `REPORT`, and `ok` (ignoring the common opening and closing schema tags). This corresponds with the *Success* row in the context shown in Table 4.1. Note that simple elements like '`:`' have also been included to produce the "richest" or most detailed context so that as much of the original specification as possible is captured. Various abstraction techniques are available to ultimately hide details that are not relevant to the current interest of the viewer. These techniques are discussed in Section 4.3.

While information hiding abstractions like the schema calculus and the 'Ξ' and 'Δ' conventions facilitate the usability of Z they also complicate the context creation process. For example, consider the *RFindBirthday* schema:

$$RFindBirthday \mathrel{\widehat=} (FindBirthday \land Success) \lor NotKnown$$

A simple parser would include the *FindBirthday*, *Success* and *NotKnown* schema names as attributes in the context but not the content contained in their expansions.

	known	. .	\power	NAME	birthday	\pfun	DATE	\ST	=	\dom	BirthdayBook	\emptyset	\Delta	name?	date?	\nem	birthday'	\union	(	\map	)	\Xi	date!	\mem	(	)	today?	cards!	n	\cbar	result!	REPORT	ok	already_known	not_known	\land	\lor
FindBirthday	×	×		×	×		×	×	×		×			×								×	×	×	×	×											
Success		×						×	×																					×	×	×				
NotKnown	×	×		×				×	×		×			×		×						×									×	×			×		
RFindBirthday	×	×		×	×		×	×	×		×			×		×						×	×	×	×	×					×	×	×		×	×	×

Table 4.2: Sub-context of Table 4.1 highlighting composition in *RFindBirthday*.

While the resulting context would be suitable for visualising relationships between schemas it could not, for example, provide an accurate view of type usage in *RFindBirthday*. Again the aim is to provide as much detail as possible during the generation of the context.

An obvious solution to this problem is to mandate expanded versions of specification for context creation. The context in Table 4.1 was produced from a source specification that included expanded versions of the *RAddBirthday*, *RFindBirthday* and *RRemind* schemas. This expansion could be performed by the specification writers or automated using tools. The ZML mark-up, for example, can render expanded schemas automatically from horizontal schema specifications.

An alternative solution is to perform automated expansion using information contained in the context itself. Given that names must be declared and defined before they are used in Z then the existing explicit declarations of the schemas can be used to infer the expansion. Provided that parsing for context creation proceeds in a linear manner through the file then the corresponding row for any named schema is already defined. If the rows of the context are considered to be bit vectors then expansions can be included by taking the logical "OR" of the current schema row along with the rows of any named schemas within it. For example, consider the *RFindBirthday* schema and a sub-context of Table 4.1 containing only the named schemas. The sub-context appears in Table 4.2.

Observe that the result represents the logical OR of the composed schemas plus the schema conjunction and disjunction operators (`\land` and `\lor`). Again, note that this

	known	. .	\power	NAME	birthday	\pfun	DATE	\ST	=	\dom	BirthdayBook	\emptyset	\Delta	name?	date?	\nem	birthday'	\union	(	\map	)	\Xi	date!	\mem	(	)	today?	cards!	n	\cbar	result!	REPORT	ok	already_known	not_known	\land	\lor
AddBirthday	×	×		×	×		×	×	×		×		×	×	×	×	×	×	×	×	×															
Success		×						×	×																						×	×	×				
AlreadyKnown	×	×		×				×	×		×			×								×		×							×	×		×			
RAddBirthday	×	×		×	×		×	×	×		×		×	×	×	×	×	×	×	×	×	×		×	×	×					×	×	×	×		×	×

Table 4.3: Sub-context of Table 4.1 highlighting composition in *RAddBirthday*. The invalid "bit" is shown in grey.

context was produced from an already expanded version of the specification so the schema definition operator (`\sdef`) and the referenced schema names have not been included in the context. However, if the same technique is applied to the *RAddBirthday* schema:

$$RAddBirthday \mathrel{\widehat{=}} (AddBirthday \land Success) \lor AlreadyKnown$$

the result includes an extra invalid "bit", shown in grey in Table 4.3. The bit is invalid because the Δ and Ξ operations are mutually exclusive — a schema operation either changes or preserves the value of state variables. These attributes should therefore be excluded from the expansion process. In keeping with the view of the object rows as bit vectors this exclusion could be performed using a logical "AND" operation and an appropriate bit-mask. If there are any Δ operations among the schemas named in a horizontal schema then the result must also be a state-changing schema.

As expected, the resulting addition also includes the schema conjunction and disjunction operators and precedence ordering brackets in the *RFindBirthday* row. Having created a formal context from a specification it is now possible to produce a concept lattice.

4.3 Abstraction in FCA

While it is possible to render the line diagram of the concept lattice corresponding to the entire context the result is often overwhelmingly complex and of little value. There are, however, notable exceptions. Snelting [177] describes a project to modularise a 20 year old aerodynamics system written in FORTRAN that was used for aeroplane development. The

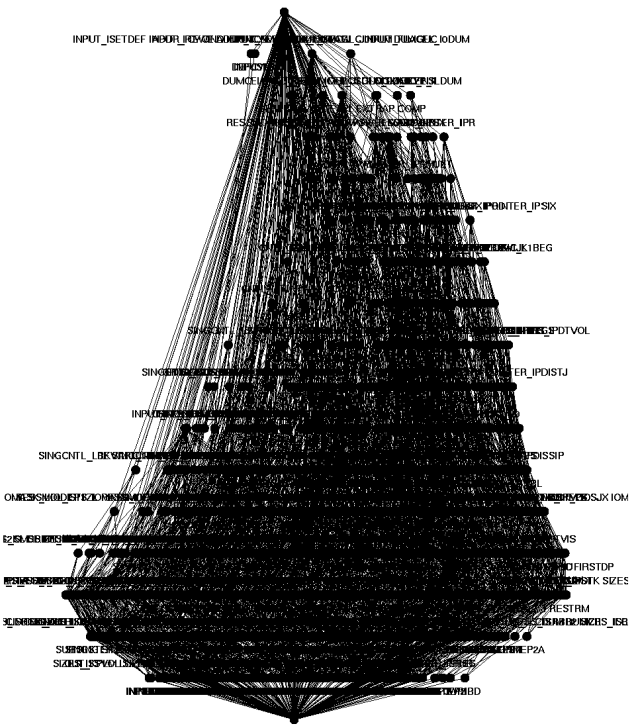


Figure 4.2: Module structure of an aerodynamics system written in FORTRAN. Despite the complexity of the diagram the concept lattice was still useful as a quality metric. This image appears as Figure 3 in *Snelting00software* [177].

106KLOC source resulted in a lattice with more than 2,249 concepts which is shown in Figure 4.2. Despite the unreadability of the diagram the concept lattice was still useful as a quality metric. The lattice was not horizontally decomposable and on the basis of this result the remodularisation project was cancelled and a new system developed from scratch.

The assumption that the whole context will be rendered as a single line diagram is a common source of naive criticism about the scalability of FCA. Rather than rendering the entire context as a single lattice a number of abstraction mechanisms are typically used. One approach to reduce the visual complexity of line diagrams is to create a line diagram of a sub-context containing only the attributes of interest. This is the simplest form of conceptual scaling. Other abstraction mechanisms include: introducing a number of many-valued contexts (an elaboration on conceptual scaling), nested line diagrams, zooming, and folding [42]. These mechanisms facilitate the ten tasks of conceptual knowledge

processing identified by Wille [227]: *exploring, searching, recognising, identifying, analysing, investigating, deciding, improving, restructuring* and *memorising*. Where:

- **Exploring** shall mean looking for something of which one has only a vague idea.
- **Searching** shall be understood as looking for something which one can more or less specify but not localise.
- **Recognising** is understood with the meaning of perceiving clearly circumstances and relationships.
- **Identifying** shall mean determining the taxonomic position of an object within a given classification.
- **Analysing** in the scope of conceptual knowledge processing is understood as examining data in their relationships while guided by theoretical views and declared purposes.
- **Investigating** means to study by close examination and systematic inquiry.
- **Deciding** shall mean resolving a situation of uncertainty by an order.
- **Improving** has the meaning of enhancement in quality and value.
- **Restructuring** means to reshape a given structure, which, within the scope of our discussion is conceptual in nature.
- **Memorising** is understood as a process of committing and reproducing what has been learned and retained.

The application of these methods to formal specifications is demonstrated in the following sections.

4.3.1 Scaling

Conceptual scaling was introduced in Section 1.5.4 and in its simplest form a sub-context can be considered as a conceptual scale. This abstraction focuses the user's attention on the objects and attributes of interest by rendering a line diagram containing only the relevant

	DATE	NAME	REPORT
BirthdayBook	×	×	
InitBirthdayBook			
AddBirthday	×	×	
FindBirthday	×	×	
Remind	×	×	
Success			×
AlreadyKnown		×	×
NotKnown		×	×
RAddBirthday	×	×	×
RFindBirthday	×	×	×
RRemind	×	×	×

Table 4.4: A sub-context considering the basic data-types from the *BirthdayBook* specification as attributes.

concepts. A scale represents a query that can be posed to reveal something about the structure or nature of the specification. A scale can also be thought of as a view over the data and this is the terminology used for scales in the Cernato FCA tool. Cernato is discussed further in Section 5.2.5. In terms of the conceptual knowledge processing tasks defined above conceptual scaling supports *exploring*, *searching*, *identifying*, and *memorising*.

Within existing FCA tools scales are normally created by a conceptual system engineer in conjunction with a domain expert. The scales attempt to capture the knowledge of domain experts so that it can be stored and applied by non-expert users. For example, in an FCA-based information retrieval project for the Center of Interdisciplinary Technology Research at TU-Darmstadt a conceptual knowledge system was designed that contained over 150 scales [161].

A sub-context of Table 4.1 containing only the basic data-types in the *BirthdayBook* specification is presented in Table 4.4. The corresponding line diagram in Figure 4.3 represents a scale showing data-type usage across the schemas in the specification.

It can be seen from Figure 4.3, for example, that the robust versions of the *AddBirthday*, *FindBirthday* and *Remind* functions all use the *REPORT* attribute. In terms of data-type usage this is the only thing that distinguished them from their less-robust counterparts.

Scales can be defined on a per-project basis like the data-type scale described above or they could be pre-defined based upon Z language constructs. For example, a sub-context

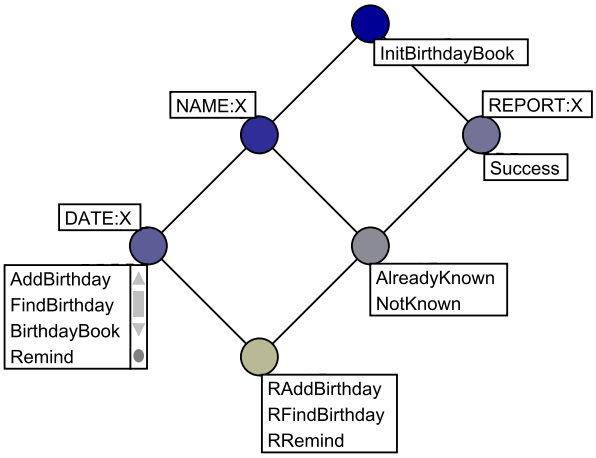


Figure 4.3: Line diagram of the concept lattice corresponding to the context in Table 4.4.

	\Delta	\Xi
BirthdayBook		
InitBirthdayBook		
AddBirthday	×	
FindBirthday		×
Remind		×
Success		
AlreadyKnown		×
NotKnown		×
RAddBirthday	×	
RFindBirthday		×
RRemind		×

Table 4.5: Formal context considering Δ (\Delta) and Ξ (\Xi) operation-types from the *BirthdayBook* specification as attributes.

based on the 'Δ' and 'Ξ' conventions is shown in Table 4.5. Note that only a count of the number of schemas in the object contingent is shown in Figure 4.4 rather than the actual schema names. This can be useful when only an idea of the distribution of the objects is required rather than a complete list of the objects at each concept. The two state schemas and the *InitBirthdayBook* function appear at the top of the diagram. The two versions of the *AddBirthday* function appear under the concept labelled `\Delta` on the right and the remaining six `\Xi` schemas on the left.

An alternative to creating scales during the construction of a conceptual knowledge

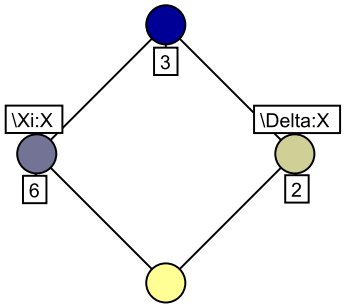


Figure 4.4: Line diagram of the concept lattice corresponding to the context in Table 4.5.

system is to generate scales on demand. Dynamically generated scales have been applied in FCA-based email and document management systems [39, 38]. The process could be semi-automated if the appropriate meta-data for the mark-up elements in the context was available. For example, a type-usage scale could be automatically generated by selecting a sub-context containing all the data-types. The ZML mark-up uses tag types to identify elements, e.g. `<type>DATE<\type>` for the *DATE* data-type. If the initial context is created from a ZML version of the specification then the tag-types can be stored as meta-data and exploited for scale generation. Other scales could be based on input and output variables, for example. Although the automatic layout of line diagrams is still a problem within the FCA community, a simple order embedding within a larger, well known lattice layout could be used to create usable, dynamically generated scales [41].

4.3.2 Visualising Schema Composition

A sub-context showing the attributes which are schemas is presented in Table 4.6. The sub-context contains references to schemas from within the object schemas and therefore represents schema composition. For example, the structure of the *RAddBirthday* schema can be seen in Figure 4.5. Traversing upwards through the lattice from the concept with the object label "RAddBirthday" recovers the attributes *AddBirthday, AlreadyKnown, Success* and *BirthdayBook*. This type of visualisation supports the conceptual knowledge processing tasks of *recognising* and *identifying* where the relationships between schemas can be clearly perceived.

	BirthdayBook	AddBirthday	FindBirthday	Remind	Success	AlreadyKnown	NotKnown
BirthdayBook							
InitBirthdayBook	×						
AddBirthday	×						
FindBirthday	×						
Remind	×						
Success							
AlreadyKnown	×						
NotKnown	×						
RAddBirthday	×	×			×	×	
RFindBirthday	×		×		×		×
RRemind	×			×	×		

Table 4.6: A sub-context representing schema composition within the *BirthdayBook* specification.

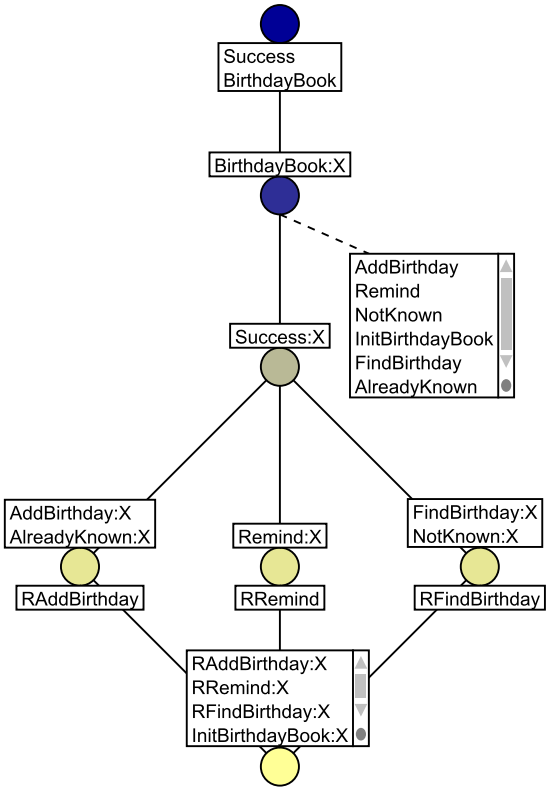


Figure 4.5: Line diagram based on Table 4.6 showing composition.

	BirthdayBook	InitBirthdayBook	AddBirthday	FindBirthday	Remind	Success	AlreadyKnown	NotKnown	RAddBirthday	RFindBirthday	RRemind
BirthdayBook	×										
InitBirthdayBook	×	×									
AddBirthday	×		×								
FindBirthday	×			×							
Remind	×				×						
Success						×					
AlreadyKnown	×						×				
NotKnown	×							×			
RAddBirthday	×		×			×	×		×		
RFindBirthday	×			×		×		×		×	
RRemind	×				×	×					×

Table 4.7: Formal context considering schema names as both objects and attributes ($G = M$).

If the context in Table 4.6 is extended so that $G = M$ and the diagonal added such that $gIg, \forall\, g \in G$, then the resulting line diagram contains a concept for each schema. Table 4.7 represents this extension and the corresponding line diagram appears as Figure 4.6. From this class-like view the structure of the robust schemas can again be clearly seen. The structure of *RAddBirthday* is made explicit in Figure 4.7 which highlights the sets of nodes both above and below the *RAddBirthday* node.

In Figure 4.7 the only unlabelled node in the diagram can be clearly seen below *Success*. This concept represents the fact that *BirthdayBook* and *Success* schemas are only combined in the robust functions.

4.3.3 Nested Line Diagrams

The power of conceptual data systems comes from the ability to combine pre-defined scales together to produce new views over the data. The contexts of multiple scales can be combined into a single context which can then be viewed as a line diagram or the scales can be combined in a nested line diagram. This supports the conceptual knowledge processing task of *analysing* where the chosen scales represent different theoretical views.

In some systems scales can be nested to arbitrary depth; however, beyond a single level of nesting the diagrams typically become too small to be usable when rendered on

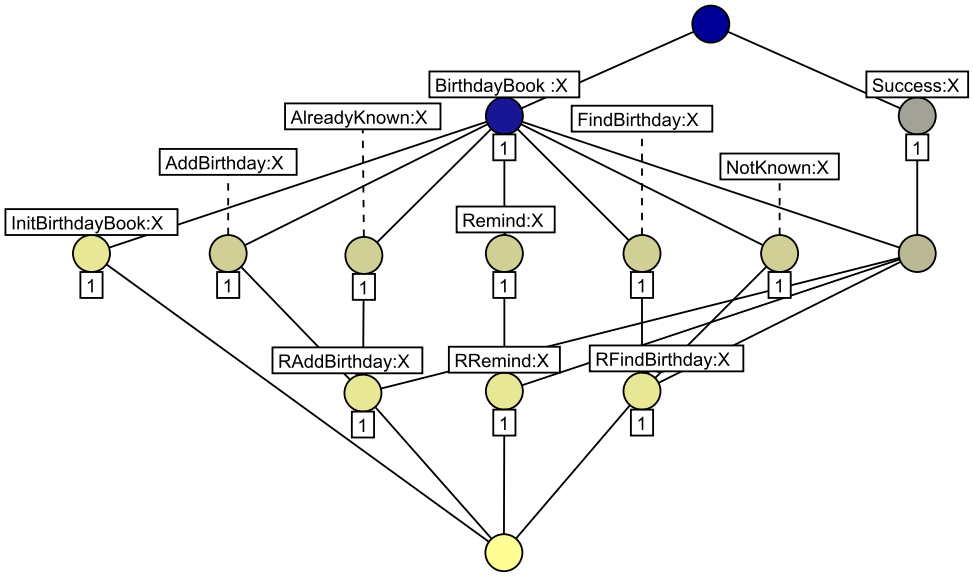


Figure 4.6: Line diagram of the concept lattice corresponding to the context in Table 4.7 showing composition relationships between schemas.

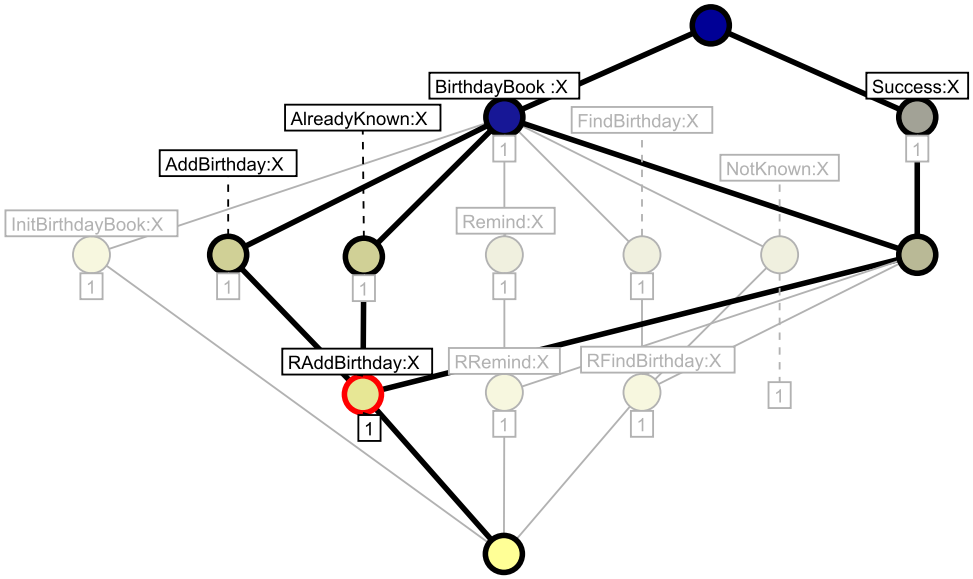


Figure 4.7: Line diagram from Figure 4.6 highlighting the ideal and filter for the “RAddBirthday” concept.

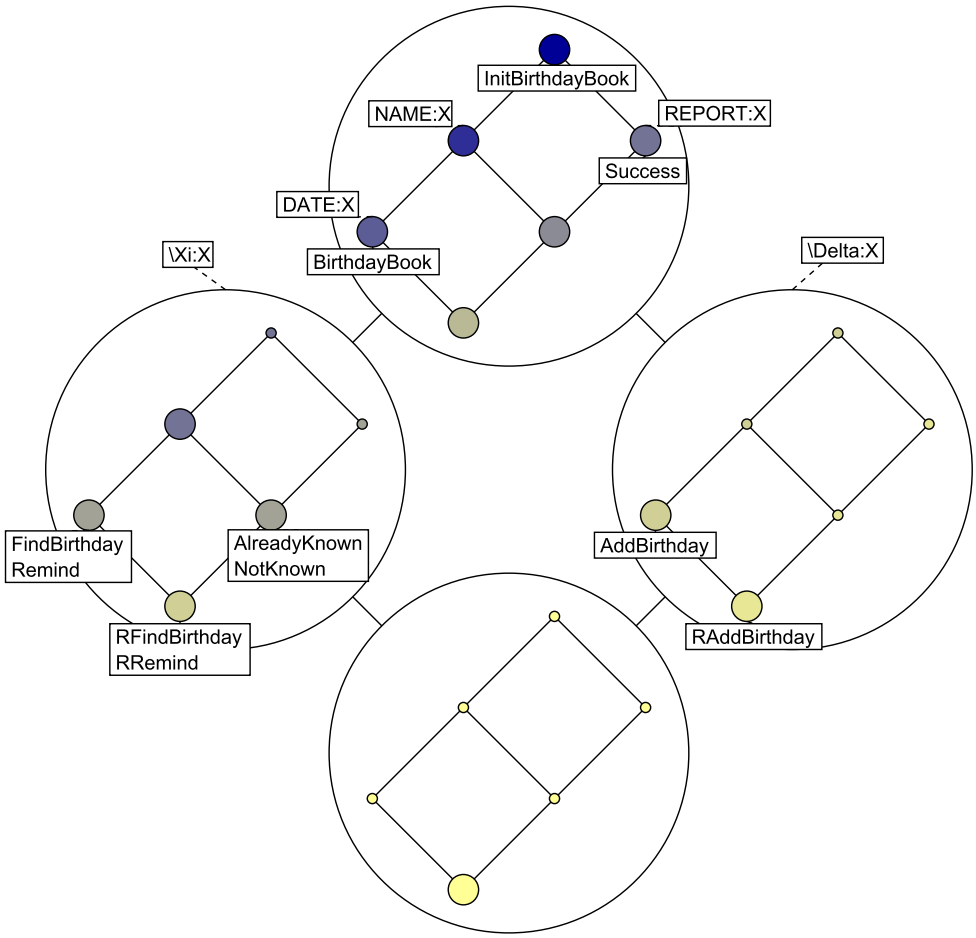


Figure 4.8: Nested line diagram showing the context from Table 4.4 nested inside the context from Table 4.5.

a computer monitor. Figure 4.8 presents a nested line diagram where the context from Table 4.4 is nested inside the context from Table 4.5. For comparison the reverse nesting is presented in Figure 4.9 with the operation-type scale nested inside the data-type scale.

From the outer, rightmost concept in Figure 4.8 it can be seen that the only state changing or 'Δ' operations in the *BirthdayBook* specification are the schemas *AddBirthday* and *RAddBirthday*. While Spivey's simple example is purely illustrative and a person's birthdate does not change over time there is typically a need in most systems for deletion and update as well as insertion. The specification may therefore be incomplete.

This is an obvious omission and although omitted in the original specification extensions are often included when the specification is presented as an example by other

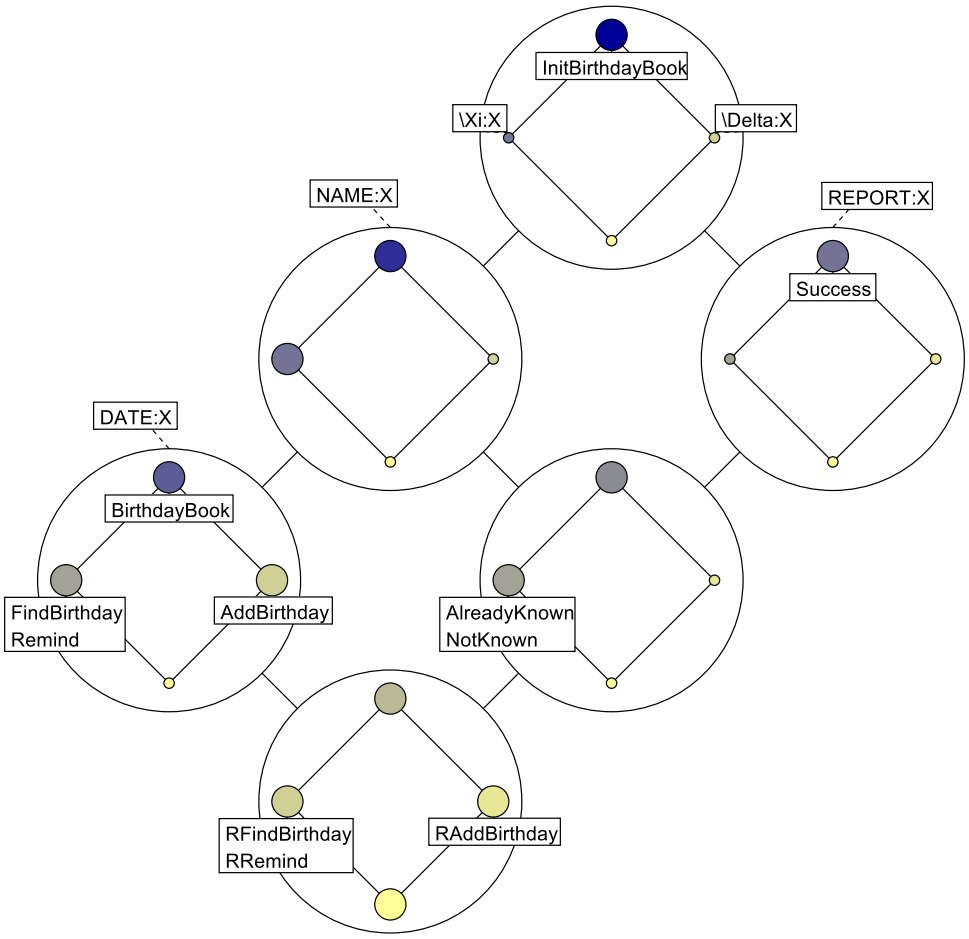


Figure 4.9: Nested line diagram showing the context from Table 4.5 nested inside the context from Table 4.4.

authors. Table 4.8 presents a context showing the basic data-types and operation-types for an extended version of the *BirthdayBook* specification based on the work of Sun et al. [48]. The extensions include a schema to remove birthdays from the system and a schema to edit existing birthday details. The complete specification appears in Appendix A. Figure 4.10 presents a nested line diagram of the extended specification for comparison with Figure 4.8. The two scales represent basic data-type and operation-type sub-contexts from Table 4.8.

Schema Composition Revisited

The structure of the extended specification from a composition point of view is presented in Tables 4.9 and 4.10 using the approach described earlier in Section 4.3.2. In the

	DATE	NAME	REPORT	\Delta	\Xi
BirthdayBook	×	×			
InitBirthdayBook					
AddBirthday	×	×		×	
FindBirthday	×	×			×
Remind	×	×			×
Success			×		
AlreadyKnown		×	×		×
NotKnown		×	×		×
RAddBirthday	×	×	×	×	
RFindBirthday	×	×	×		×
RRemind	×	×	×		×
RemoveBirthday		×		×	
ModifyBirthday	×	×		×	

Table 4.8: Formal context containing the basic data-types and the Δ (\Delta) and Ξ (\Xi) operation-types from the extended *BirthdayBook* specification.

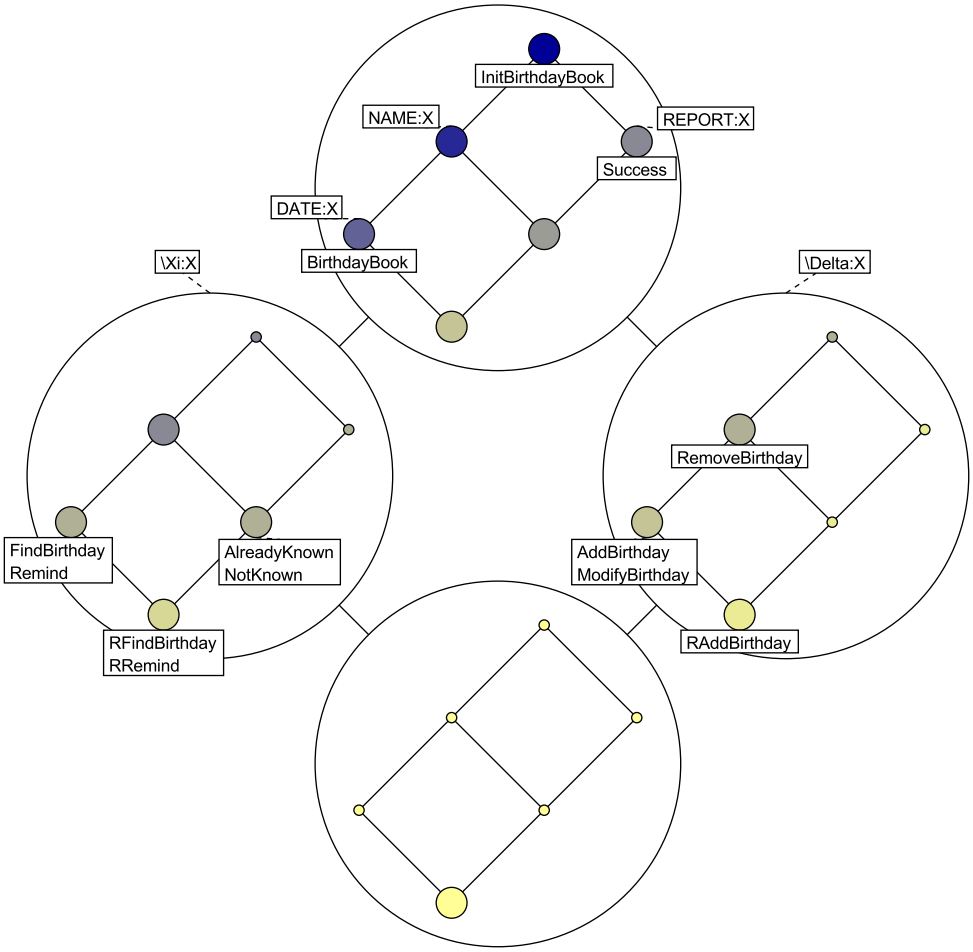


Figure 4.10: Nested Line diagram of the extended *BirthdayBook* specification. The two scales are data-type and operation-type sub-contexts from Table 4.8.

	BirthdayBook	AddBirthday	FindBirthday	Remind	Success	AlreadyKnown	NotKnown	RemoveBirthday
BirthdayBook								
InitBirthdayBook	×							
AddBirthday	×							
FindBirthday	×							
Remind	×							
Success								
AlreadyKnown	×							
NotKnown	×							
RAddBirthday	×	×			×	×		
RFindBirthday	×		×		×		×	
RRemind	×			×	×			
RemoveBirthday	×							
ModifyBirthday	×	×						×

Table 4.9: A sub-context representing schema composition within the extended *BirthdayBook* specification. The corresponding line diagram appears as Figure 4.11.

corresponding Figures 4.11 and 4.12 the implementation of the *ModifyBirthday* operation can be clearly seen. *ModifyBirthday* combines the *RemoveBirthday* and *AddBirthday* functions to first delete and then insert updated details into the system. The lack of robust implementations based on the *Success* schema can also be observed for these two functions.

4.3.4 Zooming

Zooming is another abstraction technique that is also known as *filtering*. In zooming a subset of the object set is presented for display based on the extent of a concept of interest. In conjunction with nested line diagrams zooming allows a detailed or "magnified" view of a particular concept. A user can "drill down" into the concept and this close examination is consistent with the conceptual knowledge processing tasks of *exploring*, *identifying*, and *investigating*.

Figure 4.13 presents the results of zooming the *Date* concept in Figure 4.9. Note that not only is the inner lattice displayed but also the objects *RFindBirthday, RRemind* and *RAddBirthday* from the lower concept are shown as well. The relevant objects in Table 4.11 are shown in grey.

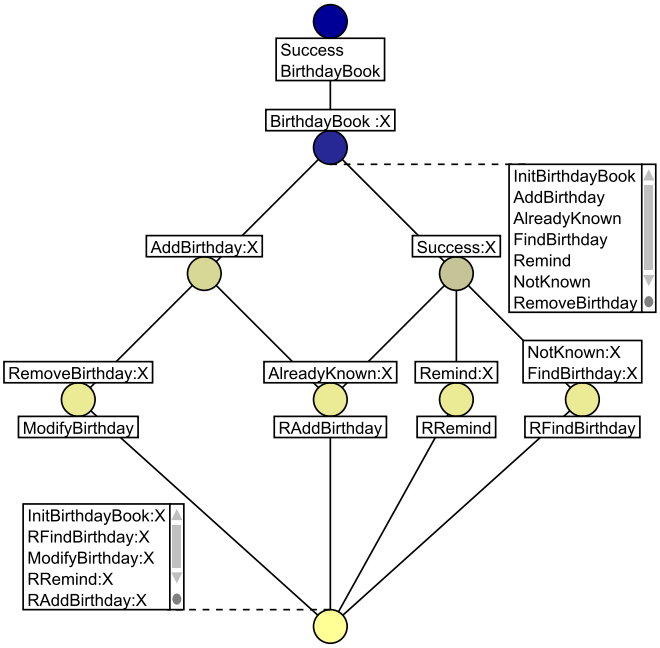


Figure 4.11: Line diagram showing schema composition within the extended version of the *BirthdayBook* specification.

	BirthdayBook	InitBirthdayBook	AddBirthday	FindBirthday	Remind	Success	AlreadyKnown	NotKnown	RAddBirthday	RFindBirthday	RRemind	RemoveBirthday	ModifyBirthday
BirthdayBook	×												
InitBirthdayBook	×	×											
AddBirthday	×		×										
FindBirthday	×			×									
Remind	×				×								
Success						×							
AlreadyKnown	×						×						
NotKnown	×							×					
RAddBirthday	×		×			×	×		×				
RFindBirthday	×			×		×		×		×			
RRemind	×				×	×					×		
RemoveBirthday	×											×	
Modify Birthday	×		×									×	×

Table 4.10: Formal context considering schema names as both objects and attributes for the extended version of the *BirthdayBook* specification. The corresponding line diagram appears as Figure 4.12.

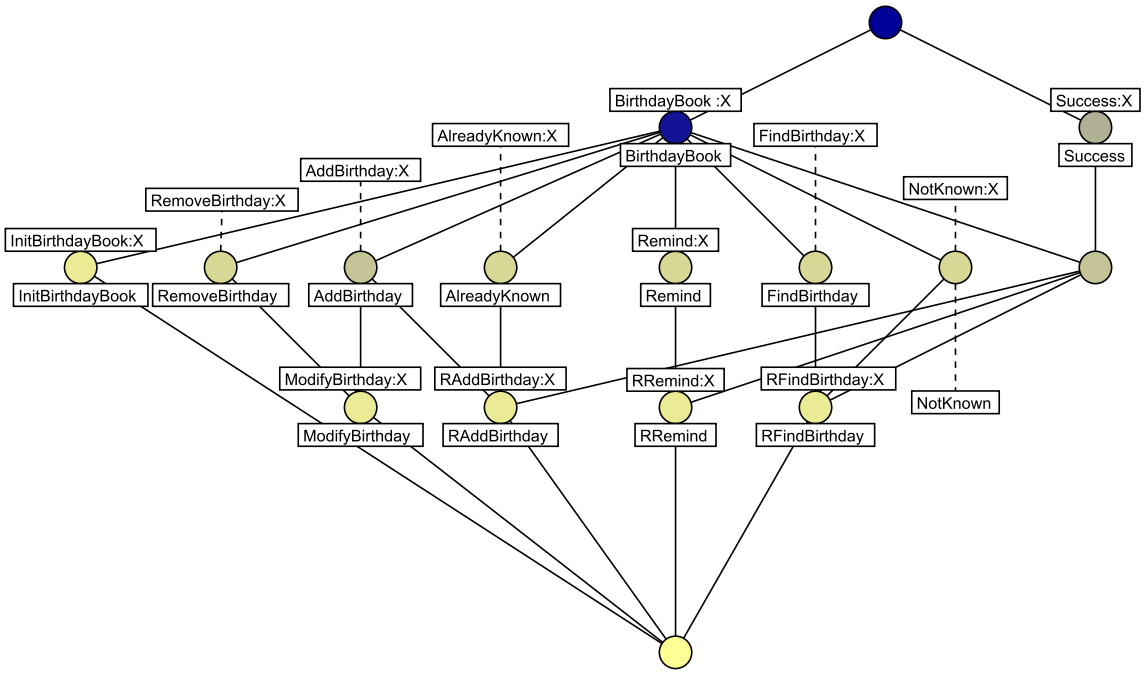


Figure 4.12: Line diagram showing schema composition in the extended *BirthdayBook* specification with schema "self-references" included.

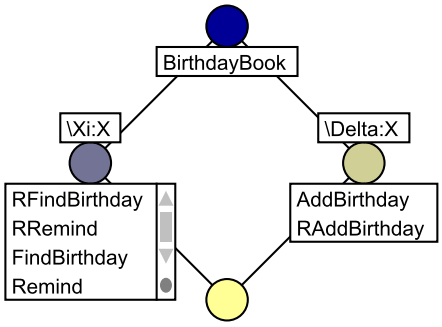


Figure 4.13: Zoomed line diagram showing the *Date* concept from Figure 4.9.

	DATE	NAME	REPORT
BirthdayBook	×	×	
InitBirthdayBook			
AddBirthday	×	×	
FindBirthday	×	×	
Remind	×	×	
Success			×
AlreadyKnown		×	×
NotKnown		×	×
RAddBirthday	×	×	×
RFindBirthday	×	×	×
RRemind	×	×	×

Table 4.11: Formal context from Table 4.4 with the objects and attributes corresponding to the zoomed line diagram in Figure 4.13 shown in grey.

4.3.5 Animation and Folding

The final abstractions to be introduced in this chapter are animation and folding, both of which are used to help users navigate within line diagrams. The term “animation” used with respect to line diagrams should not be confused with specification animation which was introduced in Section 1.6.

Folding line diagrams are scales that are constructed, or unfolded, incrementally. As attributes are added to the scale the line diagram unfolds to reveal the new structure. Animation is used to provide a smooth transition between the changing layouts [13]. This helps users to retain a sense of “where they are” within the structure of the lattice and reflects the conceptual knowledge processing task of *identifying*. Attributes can also be removed to fold or collapse the line diagram and in this way scales can be constructed interactively. Cole and Eklund discuss the use of folding line diagrams for scale construction in a medical document management system [38].

In the initial state a folding line diagram contains a single concept containing all the objects. Figure 4.14 presents three screenshots showing the incremental construction of a scale within Cernato. The example presented here is based on the case study in Chapter 3 and in the image shown top left two attributes have already been added to the scale. With the addition of a third attribute (*buy season*) the diagram again unfolds and the objects are re-distributed accordingly.

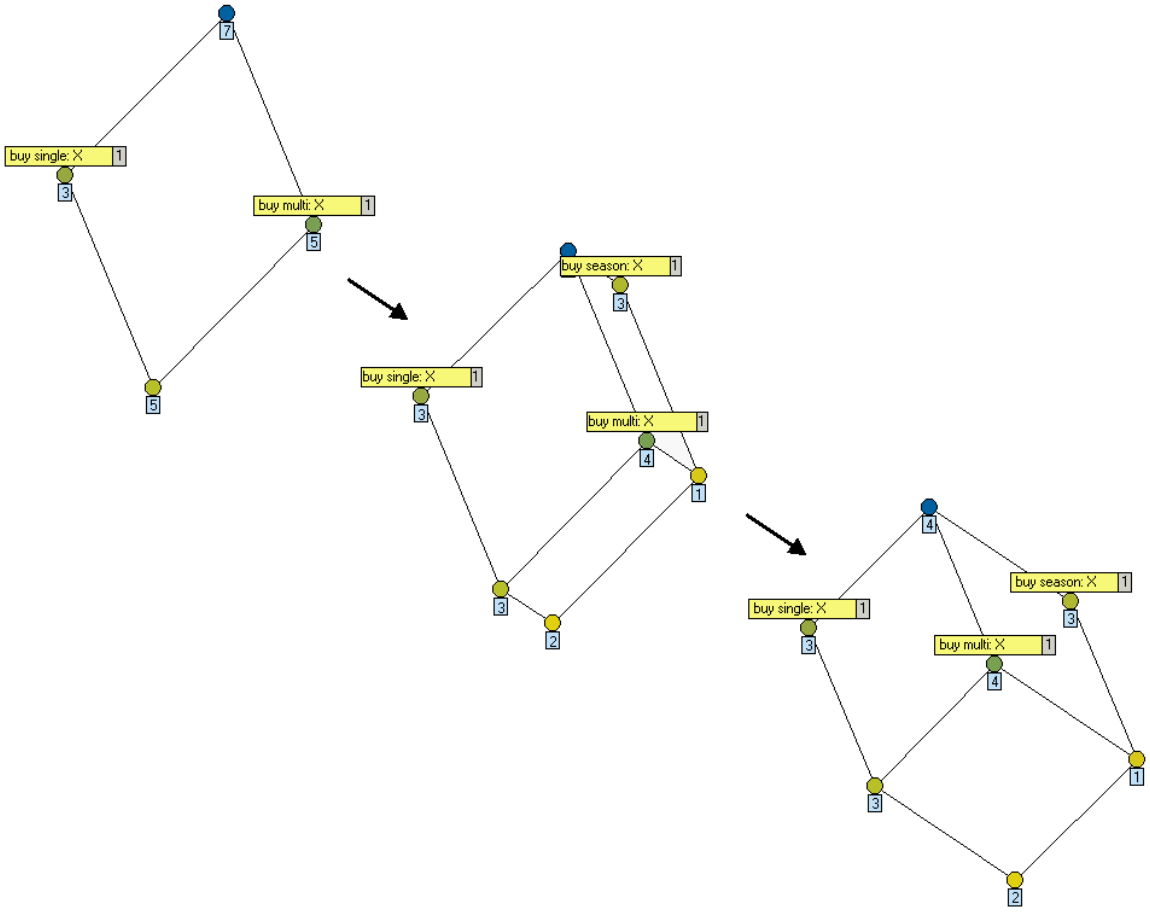


Figure 4.14: Three screenshots illustrating animation in Cernato.

4.4 Conclusion

While the visualisations presented here only represent the static structure of the specifications they provide a basis for the interactive exploration and navigation of Z specifications. Rendering the whole lattice, even for simple examples, is generally not useful, however, the ability to present only those attributes of interest or to view multiple scales via nesting facilitates the exploration of large contexts in a useful manner. Folding line diagrams and animation can also help users retain a sense of "location" within visualisations of large specifications. These abstractions illustrate the ability to handle complexity via information hiding as requested by Wing [229].

Various visualisation techniques such as zooming, nesting, and animation are well

known and used in FCA as mechanisms to navigate and elaborate structured data. This chapter has demonstrated that these techniques can also be applied to formal specifications in an easily understood and natural way. The next chapter introduces a prototype tool that embodies nearly all of the abstractions described here and a number of implementation issues are discussed.

Chapter 5

Specification Browser Implementation

This chapter describes the implementation of a tool developed by the author for interactively exploring Z specifications. The tool implements the ideas introduced in the previous chapter by exploiting ZML [195], an XML representation of Z, and the open-source, cross-platform FCA tool ToscanaJ [16, 15].

Section 5.1 of this chapter discusses Z mark-up and representation issues including a number of approaches to render Z specifications on the Web. In particular the ZML language is introduced. Section 5.2 then provides an overview of a number of FCA tools including ToscanaJ before Section 5.3 describes a tool built using ToscanaJ and ZML. The remainder of the chapter discusses the implementation of the tool including specification transformation, context creation and browser integration issues. Two brief overviews of the implementation have previously been published [199, 200] while Section 5.2 appears in a paper describing FCA tool support [202].

5.1 Representing Z

The mathematical nature of the Z notation and the graphical nature of schema boxes present some difficulties when writing specifications on a computer. The required symbols are unavailable in traditional text editors and there have been various approaches to representing Z in documents in both human and machine readable forms. Since the ISO standardisation of Z [104] the required symbols have been incorporated into the Unicode

```
\begin{schema}{AddBirthday}
    \Delta BirthdayBook \\
    name? : NAME \\
    date? : DATE
\ST
    name? \nem known \\
    birthday' = birthday \union \{name? \map date?\}
\end{schema}
```

Figure 5.1: Oz style LaTeX mark-up for the *AddBirthday* schema.

character set [207, 102, 103]. As a result, fonts that support Z are now available for use in word processors and other applications. Traditionally, however, most Z specifications have been created using LaTeX mark-up.

5.1.1 LaTeX Z Styles

There are a number of LaTeX style or class files available for Z including *f*uzz [187, 186], ZED [185] and Oz [118]. A specification document is prepared using a text editor and the schemas are written using mark-up. For example, the *AddBirthday* schema in Oz style mark-up is presented in Figure 5.1. The complete *BirthdayBook* specification appears in Appendix A.

The document is then processed with a style file to produce a final version of the specification in either PostScript or Portable Document Format (PDF) which can be printed or viewed using a suitable document reading application. This process encapsulates the separation of content and presentation and the required rendering steps are illustrated in Figure 5.2.

In addition to creating human readable specifications, tools can also parse the LaTeX specification mark-up. Most Z tools have traditionally incorporated a formatting system and a type-checker that accepts Z specifications in LaTeX. For example, the *f*uzz type-checker checks for type-usage, syntax and scope errors in documents prepared using the *f*uzz LaTeX style. Other Z tools that process LaTeX mark-up input include CADiℤ [206, 205], ProofPower [127], Wizard [112], Z/EVES [165] and Zeta [88].

CADiℤ also includes macros for creating specifications using mark-up for the *troff*

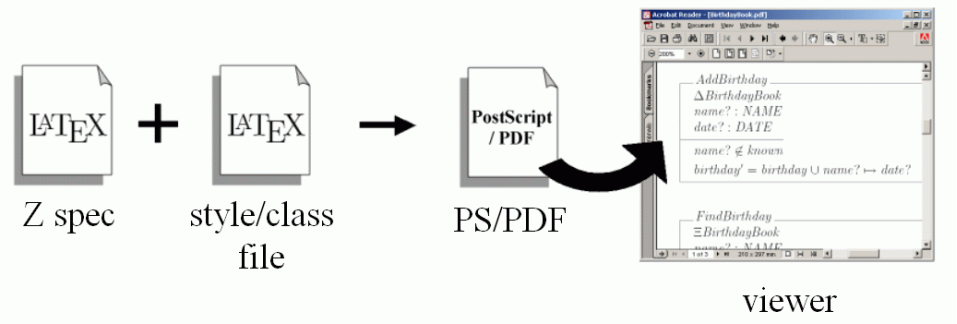


Figure 5.2: Overview of the rendering process from a LaTeX source document to a final PostScript or PDF document.

```
.ZS AddBirthday
\(*DBirthdayBook
name? : NAME
date? : DATE
.ZM
name? notmem known
birthday' = birthday sor { name? mlet date? }
.ZE
```

Figure 5.3: Troff mark-up for the *AddBirthday* schema.

typesetting tool for Unix-based systems. The *AddBirthday* schema in troff [43] mark-up is shown in Figure 5.3.

Z Browser

The Z Browser [147, 137] is an example of a commercial Z specification browsing tool that provides an alternative to Postscript or PDF rendering. The browser takes specifications written in the `zed` style and renders them on the Windows platform using a custom-made true-type font. A screenshot of the browser displaying the *AddBirthday* schema from the *BirthdayBook* specification appears in Figure 5.4.

The browser attempts to address some of the problems for users of large specifications by providing links between data-type and schema definitions within the specification. For example, a user can click on a schema name and the definition of the schema will then be displayed. In addition, all the symbols are linked to an online help system which provides

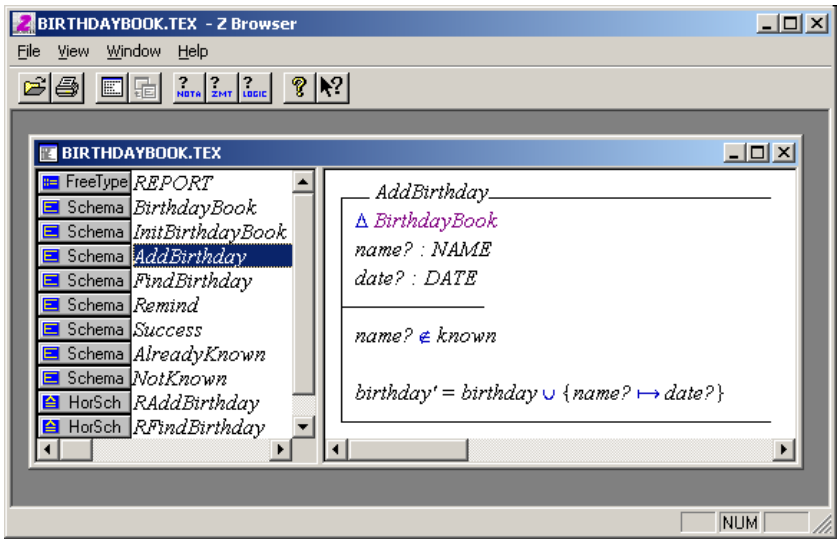


Figure 5.4: Screenshot of the *AddBirthday* schema in the Z Browser.

information about the symbols and also the notational conventions used in Z for novice users.

A symbol mapping file that can be edited by users facilitates the extension of the system to incorporate other LaTeX mark-up. In addition, other applications can also use the Z Browser as a viewer to open specifications and display Z paragraphs via the Windows platform DDE (Dynamic Data Exchange) protocol. This facility supports the potential integration of the browser with other tools.

A plug-in version of the Z Browser is also available for Netscape Navigator on the Windows platform [148]. While the rendered Z symbols are hyperlinked to help files, cross-referencing within specifications is not supported.

5.1.2 Z in ASCII

The LaTeX mark-up approach to Z essentially results in two specifications — one that is rendered and intended to be read by humans and the other a source file intended for use by tools. There have been a number of attempts to incorporate the two to create a representation which is both human readable, retaining as much of the visual appearance of Z as possible, while also being machine readable. These representations are based

```
+-- AddBirthday ---
  %Delta%BirthdayBook
  name? : NAME
  date? : DATE
|--
  name? %/e% known
  birthday' = birthday %u% { name? %|-->% date? }
---
```

Figure 5.5: Z Standard Email mark-up for the *AddBirthday* schema.

on simple ASCII characters so no special tool support or rendering is required. The specifications can be easily incorporated into email and also facilitate the use of Z by people who are not familiar with LaTeX, for example, some university student groups. The *AddBirthday* schema in Z-standard email mark-up is shown in Figure 5.5.

The Alloy notation was briefly discussed in Section 4.1 and it represents an example of a Z-like ASCII-based language. Alloy is a lightweight formal method based on Z that provides a straight forward mapping into UML. The notation is also supported by the Alloy constraint analyser tool, formerly known as Alcoa [106]. While Alloy is only Z-like, ZSL is an example of an actual ASCII-based Z notation.

ZSL

ZSL is an ASCII-based Z notation that is as mathematically expressive as LaTeX mark-up but not as visually expressive. ZSL has two styles: a text style which is very similar to the LaTeX mark-up shown in Figure 5.1; and a box style which is closer to the traditional rendered form of Z. The *AddBirthday* schema in ZSL appears in both the "text" and "box" styles at the top and bottom of Figure 5.6 respectively.

ZSL is designed for use with the ZTC [110] type and syntax checking tool and is also supported by the ZANS animation tool [111]. In addition to ZSL input, the ZTC tool also accepts specifications written in either the `zed` or Oz styles. ZTC can also be used to translate LaTeX mark-ups into ZSL.

```
schema AddBirthday
    Delta BirthdayBook;
    name? : NAME;
    date? : DATE;
where
    name? notin known;
    birthday' = birthday Union {name? -> date?}
end schema


--- AddBirthday ---------------------------------
|  Delta BirthdayBook;
|  name? : NAME;
|  date? : DATE;
|----------
|  name? notin known;
|  birthday' = birthday Union {name? -> date?}
-------------------------------------------------
```

Figure 5.6: ZSL mark-up for the *AddBirthday* schema in both the "text" (top) and "box" (bottom) styles.

5.1.3 Z on the Web

While ASCII-based representations facilitate communication via email or in newsgroups in a text-only form, there has also been work to present Z in a rendered form on the Web. As Ciancarini, Mascolo and Vitali [36] point out there are a number of reasons that support the use of hypertext for representing Z. First, the relationships between the components such as schemas within a specification can be represented in the document via hyperlinks. Second, the ability to publish specifications on the Web supports collaboration and sharing. Finally, the hypertext medium also facilitates the literate-programming-like notion of interleaving text with the specifications. Knuth [120] had proposed *literate programs* as a way of combining source code and descriptive text into a single, compilable document. This idea was further extended by Ryman [164] who also incorporated formal methods with literate programs.

A first approach to representing Z on the Web is the use of in-line GIF images to represent Z symbols in HTML documents. Jacky [108] and Stepney [188] both use this

approach which is platform independent and works with any browser that supports images. Jacky also provides a script-based tool *Z2HTML* [108] which translates LaTeX specifications in either the `zed` or *fuzz* styles into HTML pages with references for embedded symbol images. The main drawback of this approach is that the images do not scale and the appropriate font size must be used. In addition, the images are generally of a low quality when printed. While it is also possible to create hyperlinks to cross-reference any part of the specification these are not generated by the tool and would need to be marked-up by hand.

Applet-based Approaches

An alternative to in-line symbol images is the use of a browser plug-in or Java applet to render Z within web-pages. These approaches allow text and specification to be interleaved which supports the literate programming notion. An example from Section 5.1.1 is the Z browser plug-in, however, this particular implementation has two disadvantages. First, it is both platform and browser specific, and second, it does not support hyperlinking within specifications.

The work of Bowen and Chippington [27] and Ciancarini, Mascolo and Vitali [36] are both applet-based approaches that use the Z Interchange Format (ZIF) [79]. ZIF is a Standard Generalised Mark-up Language (SGML) [101] based representation of Z proposed in an early draft of the Z standard for exchanging specifications between different machines and tools [79]. The *AddBirthday* schema in ZIF mark-up is shown in Figure 5.7. The format, however, ultimately proved difficult to maintain [211] and was not included in the ISO Standard. More recently, an XML alternative based interchange format has been proposed which is discussed in Section 5.1.3.

Ciancarini et al. have created *displets* — display applets — that allow HTML extensions to be declared and rendered in web-browsers. One of their extensions supports the display of Z specifications in ZIF and is supported by a tool called *Zed2HTML*. The tool transforms specifications written using Oz style LaTeX mark-up into a corresponding HTML document that contains the corresponding ZIF representation. Within the document both the ZIF tags and their syntax are defined along with the actual specification as parameters to an applet

```
<schemadef>
    AddBirthday
    <decpart>
        <declaration> &Delta; AddBirthday </declaration>
        <declaration> name?: NAME </declaration>
        <declaration> date?: DATE </declaration>
    </decpart>
    <axpart>
        <predicate> name? &notin; known </predicate>
        <predicate> birthday' = birthday &uni; {name? &map; date?} </predicate>
    </axpart>
</schemadef>
```

Figure 5.7: Z Interchange Format mark-up for the *AddBirthday* schema.

which loads the appropriate displet for rendering. Hyperlinks are also automatically created between data types and their declarations and standard HTML can be interleaved within the specification.

Bowen and Chippington's `ZDisplay` applet also renders Z specified in ZIF. The specification can be provided in-line as a parameter to the applet or as input from a separate file. Their approach does not automatically create hyperlinks between specification components and the Z symbols are implemented as GIF images.

While both of these approaches use the now-deprecated ZIF, the work was conducted during the period in which Z was undergoing standardisation. At the time the SGML-based ZIF was an obvious choice for integration with HTML which is also an application of SGML.

MathML

MathML [221] is an XML-based mark-up language for rendering mathematics on the Web and XML, like HTML, it is also based on SGML. This presents another possible rendering approach for the required Z symbols. MathML is currently supported by the Netscape, Mozilla and Amaya browsers, however, other browsers still require plug-ins [222]. Unfortunately for the aforementioned applets, MathML was not an implementation option. MathML version 1.0 was only released in June 1998, by which time papers describing the applets were already being published.

While MathML removes the need for plug-ins for most browsers, another approach that moves rendering responsibility to the browser is the use of Z-compatible fonts. Prior to the inclusion of Z symbols in Unicode font-based rendering was haphazard because it relied on custom fonts that may not be available on all platforms or individual fonts may have used different character mappings for the same symbol. The inclusion of Z symbols in the Unicode standard means any browser with a suitable Unicode Font can now reliably render the required symbols and the responsibility for rendering can be shifted to the browser itself. This is the approach taken by the Z Mark-up Language — ZML.

ZML

ZML is an XML representation for Z originally developed by Sun, Dong, Lui and Wang [195, 194] at the National University of Singapore. They chose XML over MathML because their original aim in ZML was to provide a Web environment as close as possible to the Oz LaTeX style for Object-Z to minimise translation requirements. They argue that not only are the schema boxes more difficult to construct in MathML but also the large number of tags which focus on the structure of expressions detracts from MathML's use for model abstraction. Figure 5.8 presents the *AddBirthday* schema in ZML and the complete *BirthdayBook* specification in ZML, LaTeX and rendered form appears in Appendix A.

The eXtensible Stylesheet Language (XSL) [220] covers a family of languages used for processing XML documents. One of these languages — XSL Transformations (XSLT) — can be used to transform one XML document into another XML format or to transform XML into HTML. ZML makes use of XSLT to transform the XML-based specification into a HTML document that can then be displayed in a web-browser. The process is illustrated in Figure 5.9. An XSL stylesheet describes the transformation rules which are then applied via an XSLT processor to produce a HTML version of the specification. Note the parallel with the process required for specification in LaTeX mark-up from Figure 5.2.

The structure of XML documents can be described using one of two formats: either a Document Type Definition (DTD) which originated in SGML; or XML Schema [223] — an XML language for describing the structure of XML documents. ZML makes use of both formats. The structure and syntax of ZML is described using XML Schema and this can

```
<schemadef layout="simpl" align="left">
    <name>AddBirthday</name>
    <del>
        <type>BirthdayBook</type>
    </del>
    <decl>
        <name>name?</name>
        <dtype>
            <type>NAME</type>
        </dtype>
    </decl>
    <decl>
        <name>date?</name>
        <dtype>
            <type>DATE</type>
        </dtype>
    </decl>
<st/>
    <predicate>name? &nem; known</predicate>
    <predicate>birthday' = birthday &uni; {name? &map; date?}</predicate>
</schemadef>
```

Figure 5.8: ZML mark-up for the *AddBirthday* schema.

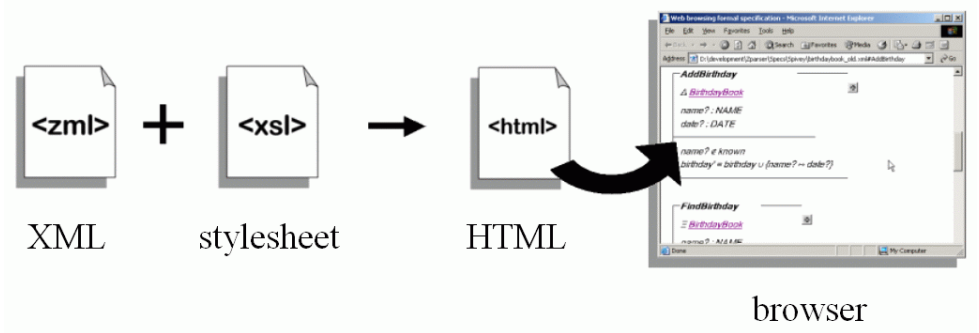


Figure 5.9: Overview of the rendering process from a ZML source document to a final HTML document. Note the parallel with Figure 5.2.

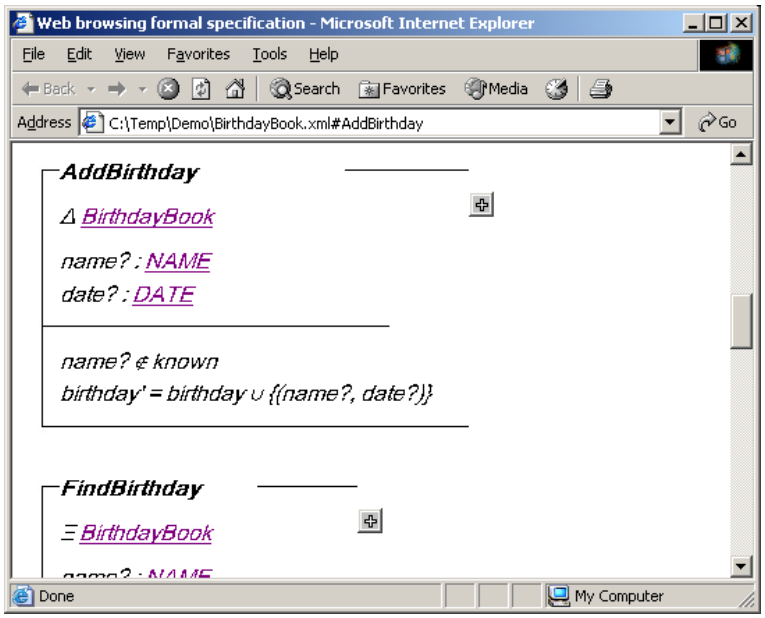


Figure 5.10: Screenshot of the *BirthdayBook* specification rendered using HTML and Unicode in a web-browser. The *AddBirthday* schema is shown. Note the underlined hyperlinks used for schema and data-type definitions.

be used by XML-aware tools and browsers to validate the syntax of a ZML specification. ZML also provides a DTD to describe mappings between LaTeX style Z symbol names (e.g. `uni` for set union), and their corresponding Unicode characters (`#x222a`). HTML supports Unicode for character encoding so a browser with a Unicode font can render the required symbols in a HTML version of the specification. Figure 5.10 presents a screenshot of the *BirthdayBook* specification displayed in a web-browser.

Client-side XSLT processing has been available since version 5 of the Internet Explorer web-browser and Microsoft also provides a 23 Megabyte TrueType Unicode Arial font in their Office 2000 distribution. Equipped with these two tools it is possible to render a ZML specification directly in the browser as illustrated in Figure 5.11. In terms of platform independence, the ZML can be rendered in any browser for which a Unicode font is available and the XSLT processing can be performed server-side to make ZML specifications available to older browsers that do not support XSL.

In terms of the arguments put forward by Ciancarini et al. for using hypertext-based specifications, ZML allows descriptive text to be interleaved with the schemas in a ZML

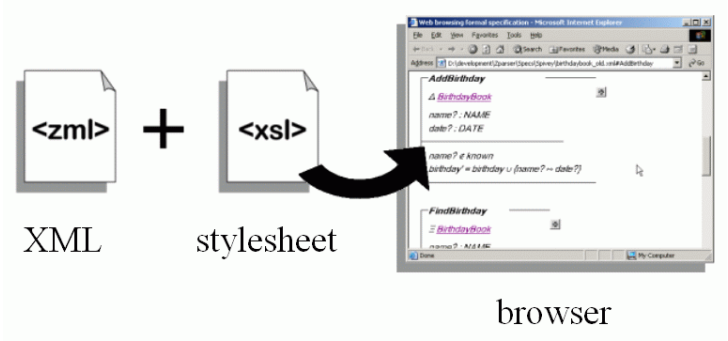


Figure 5.11: The XML to HTML transformation via XSLT shown in Figure 5.9 can be performed within the browser.

document and hyperlinks to schema and data-type definitions are automatically created during XSLT processing. In addition ZML also provides automatic expansions for the 'Δ' and 'Ξ' conventions, schema calculus, and inheritance in Object-Z. This functionality is achieved by exploiting the match facilities in XSL to locate the required definitions within the specification.

Figure 5.12 presents two screenshots of the *RRemind* schema from the *BirthdayBook* specification as displayed in a web-browser. The top schema shows the unexpanded linear form. Note that the *Remind* and *Success* schemas are presented as hyperlinks back to their earlier definitions in the specification. The '⊞' icon denotes that an expansion of the schema is available. Clicking on the icon produces the lower schema showing the expansion of the schema composition. The schema can now be collapsed back to the linear form by clicking on the '⊟' icon. Alternatively, the schema can be further expanded to make the Ξ shorthand explicit.

Although a font is used to display and print the Z symbols, ZML still makes some use of images to render schema boxes. The scalability is only limited, however, by the available Unicode font sizes and in addition to Z and Object-Z, ZML also supports the Timed Communicating Object-Z (TCOZ) notation [133, 134].

This initial version of ZML was influenced not only by Spivey's version of Z [184], but also by Object-Z [174] and TCOZ [133]. The XML element names were chosen to

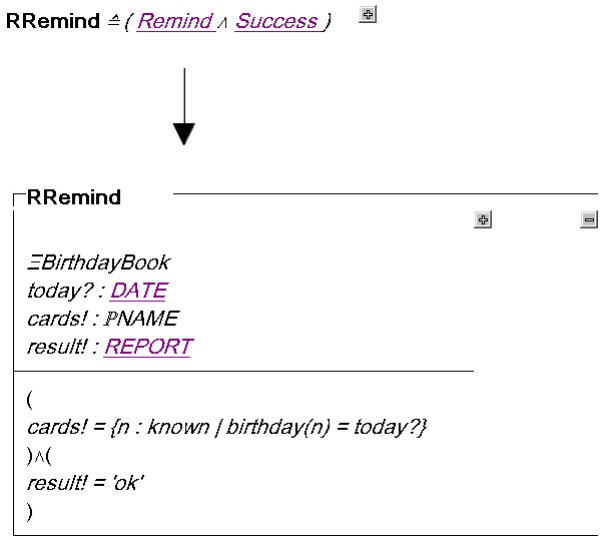


Figure 5.12: Two screenshots of the *RRemind* birthday schema illustrating schema expansion in ZML. The top schema shows the unexpanded linear form. Note the '⊞' expand and '⊟' collapse icons.

provide a straightforward mapping from LaTeX specifications and both animation and type-checking tools based on the format have been reported [51, 196]. Furthermore, Sun et al. have also demonstrated techniques and tools to project Object-Z ZML specifications into UML [195].

An alternate, more detailed version of ZML for machine interchange has also been defined. In contrast with Figure 5.8, Figure 5.13 presents the *AddBirthday* schema using the fully annotated form of ZML [192, 49]. This version of the mark-up has a more extensive tag set. Rather than consisting of the traditional declarative part and a predicate (formula) part the schemas now contain a list of `<declaration>` and `<predicate>` elements. In Figure 5.8 a predicate tag contains a complete predicate however the new version requires each atom to be explicitly identified via tags. The transformation from a LaTeX representation of Z into this version of ZML is no longer quite so "trivial". Additionally, an updated version of the "interchange" format based on Standard Z has also been produced [50].

```
<schemaDef>
    <name>AddBirthday</name>
    <deltaList>BirthdayBook</deltaList>
    <declaration>
        <variable>name?</variable>
        <dataType>
            <type>NAME</type>
        </dataType>
    </declaration>
    <declaration>
        <variable>date?</variable>
        <dataType>
            <type>DATE</type>
        </dataType>
    </declaration>
    <predicate>
        <expression>
            <varName>name?</varName>
        </expression>
        <relationSym>nem</relationSym>
        <expression>
            <varName>known</varName>
        </expression>
    </predicate>
    <predicate>
        <expression>
            <expression>
                <varName>birthday</varName>
            </expression>
            <postfixExpr>'</postfixExpr>
        </expression>
        <relationSym>=</relationSym>
        <expression>
            <expression>
                <varName>birthday</varName>
            </expression>
            <exprConnSym>uni</exprConnSym>
            <expression>
                <left>{</left>
                <expression>
                    <left>(</left>
                    <expression>
                        <expression>
                            <varName>name?</varName>
                        </expression>
                        <exprConnSym>,</exprConnSym>
                        <expression>
                            <varName>date?</varName>
                        </expression>
                    </expression>
                    <right>)</right>
                </expression>
                <right>}</right>
            </expression>
        </expression>
    </predicate>
</schemaDef>
```

Figure 5.13: The *AddBirthday* schema marked-up using the “interchange” version of ZML.

Other XML-based Z Representations

As with the plethora of LaTeX and ASCII mark-up languages for Z there are also a number of XML-based alternatives to ZML. These representations are typically associated with tools and are more aligned with the "interchange" version of ZML described above. Z/EVES [149], for example, uses an XML interchange format for passing specifications between tools. Wordsworth [231] and Toyn [211] have both proposed DTD-based XML representations for Z. Wordsworth's implementation is based on Spivey's notation while Toyn's representation for Standard Z was heavily influenced by the abstract syntaxes of both Zeta [88] and his own CADiℤ tool [205]. CADiℤ is able to export specifications in XML format.

Most recently, however, a new version of ZML has emerged whose authors include Toyn, Sun and Dong from the National University of Singapore, and Martin [135] of the Community Z Tools initiative, among others [211]. This annotated interchange notation is defined using XML Schema but is largely based on Toyn's earlier DTD representation. It effectively represents an XML-based alternative to ZIF which was ultimately dropped from the Z Standard. The format's authors hope that in the future this version of ZML may become an integral part of the ISO Z standard.

Despite this recent advance, however, Section 5.3 describes the implementation of a prototype specification navigation and visualisation tool that exploits the initial implementation of ZML in combination with an open-source FCA tool. There are a number of reasons for using ZML and the early version in particular. First, ZML effectively permits both the editing and visualisation of Z specifications from a single document. Although transformation to HTML is still required this can be handled transparently and automatically by the browser using XSLT. Browser-based specifications can also be delivered and shared online and the hyperlink anchors can be exploited by tools as a way of displaying any schema or data-type declaration within a ZML specification. Furthermore, ZML automatically creates intra-specification hyperlinks and supports the automated expansion of schema inclusions, calculus and inheritance.

In addition to the reasons outlined above, the initial version of ZML was chosen to

implement the tool prototype because the direct mapping between the LaTeX mark-up and the initial version of ZML makes transformation from existing LaTeX specifications easier. The mark-up is also simpler to read which aids debugging. Debugging is further supported by the direct correspondence between the two mark-ups.

Having introduced ZML, and discussed some of the representation issues with Z, the next section now turns back to FCA. Section 5.2 provides an overview of FCA tool support before the implementation of the specification navigation and visualisation tool is discussed in Section 5.3.

5.2 FCA Tools

This section provides an overview of FCA tool support. These tools reflect the recent history of computing which ranges from the early DOS-based implementation of Duquenne's GLAD tool in FORTRAN to platform-independent Java-based tools currently under active development like ConExp and ToscanaJ. Both commercial and open-source software appears in the list which also includes general-purpose and application-specific tools.

In particular, the overview provides some insight into the work-flow and design of the ToscanaJ tool which embodies ideas refined over a number of generations of software. A number of line diagrams summarising features of the general tools are presented in Section 5.2.9.

5.2.1 GLAD

Duquenne's tool for General Lattice Analysis and Design (GLAD) is possibly the earliest software tool that facilitates the analysis of formal concept lattices [54, 53, 55]. GLAD is a DOS-based program written in FORTRAN that has been under development since 1983. The tool facilitates the editing, drawing, modifying, decomposing and approximation of finite lattices in general and is not restricted to the analysis of concept lattices. The lattices to be analysed can be derived from abstract mathematics or applied statistics using techniques like Analysis of Variance. Single-valued data can also be analysed by exploiting

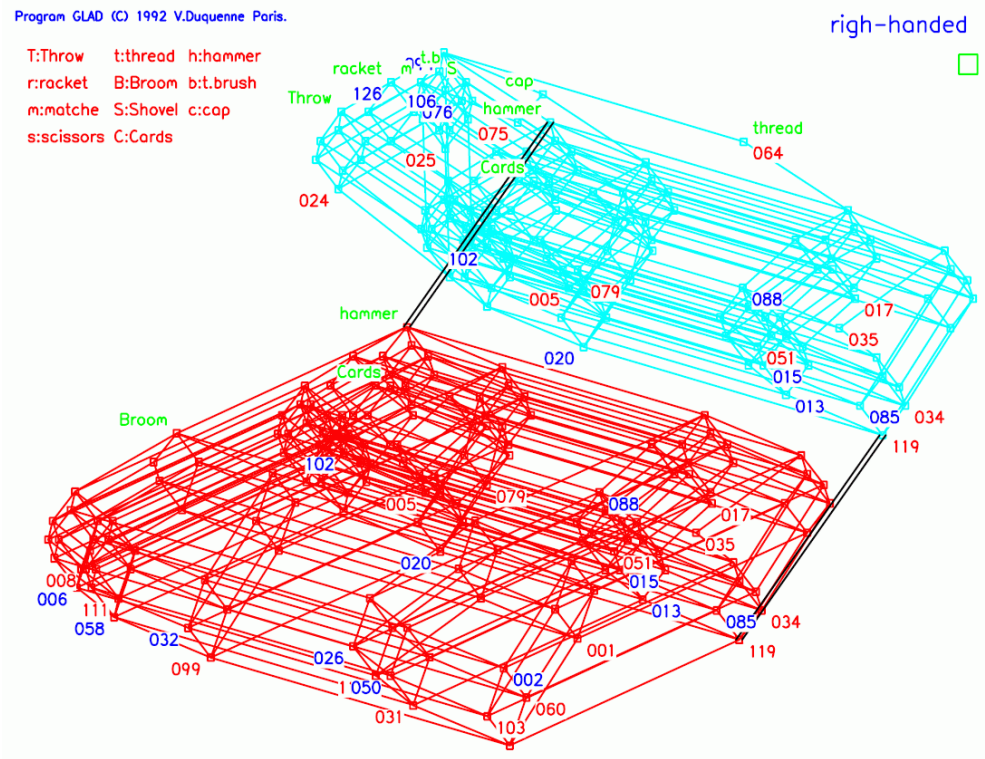


Figure 5.14: A lattice diagram produced by Duquenne's GLAD tool. The lattice represents a *gluing* decomposition of questionnaire results about right-handed writers.

the classic correspondence between lattices and binary relations identified by Birkhoff [19].

Lattice diagrams can be output directly from GLAD in the Hewlett Packard Graphics Language (HPGL) [30] — a vector-based language designed for plotters. Figure 5.14 presents a line diagram produced by GLAD which originally appears as Figure 2 in a paper describing the application of Galois lattices to behavioural genetics [55]. The lattice represents an *un-gluing* decomposition of questionnaire results about right-handed writers. Un-gluing breaks a large lattice into smaller lattices by separating them along common substructures [78].

GLAD contains a large number of features, many of which are undocumented and it also supports "scenarios" which represent a form of macro. These scenarios can be used to regenerate and manipulate a lattice by recalling the list of commands used to construct it.

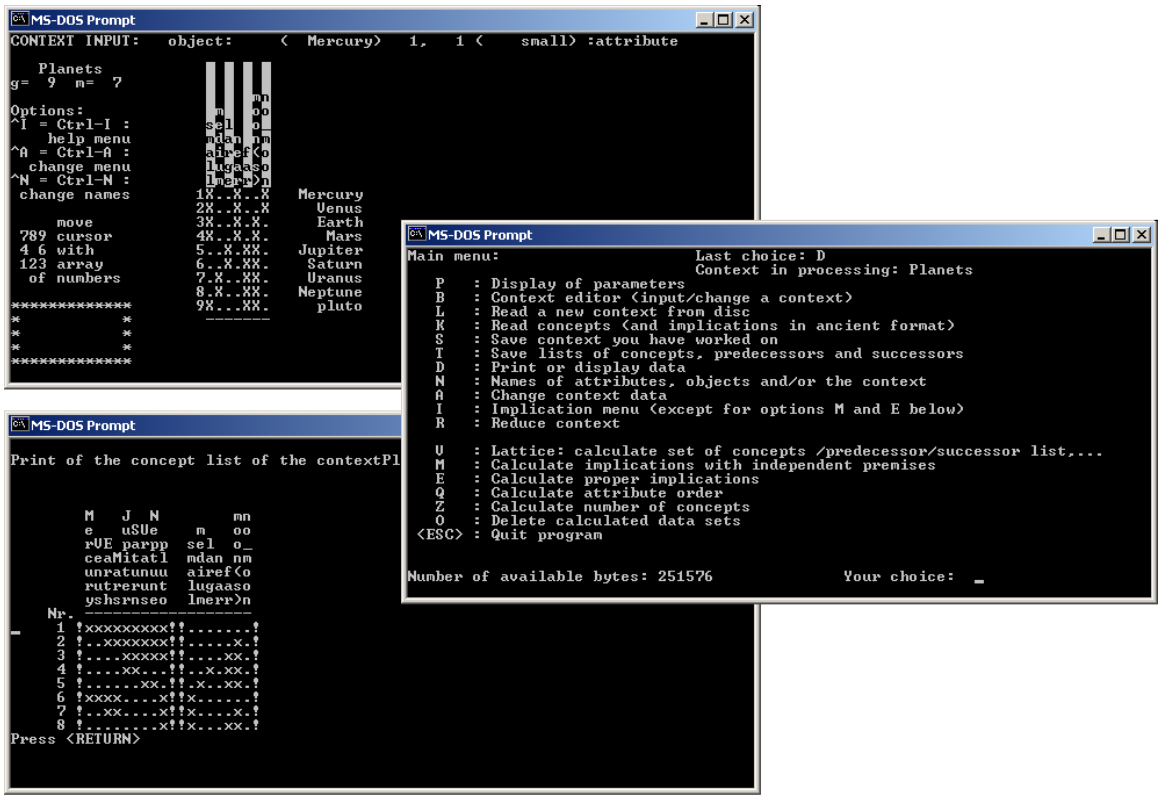


Figure 5.15: Three screenshots of the DOS-based ConImp tool. The context editing screen is shown top left, the display of concepts at bottom left and the main menu on the right.

5.2.2 ConImp

ConImp (**Con**texts and **Imp**lications) is another DOS-based tool implemented by Burmeister [32] who started development in 1986 on an Apple II computer. While ConImp is purely text-based and provides no graphical output for lattices it also supports a wide range of features for manipulating contexts and provides concept listings which can be used for drawing line diagrams by hand.

Three screenshots of the DOS-based ConImp tool are shown in Figure 5.15. The top screenshot shows the planets example from Table 1.1 in the context editor screen. The main menu displaying the large number of available options is shown centre right while the concept list is shown at the bottom.

The *Duquenne-Guiges-base* represents a canonical base of valid implications for a

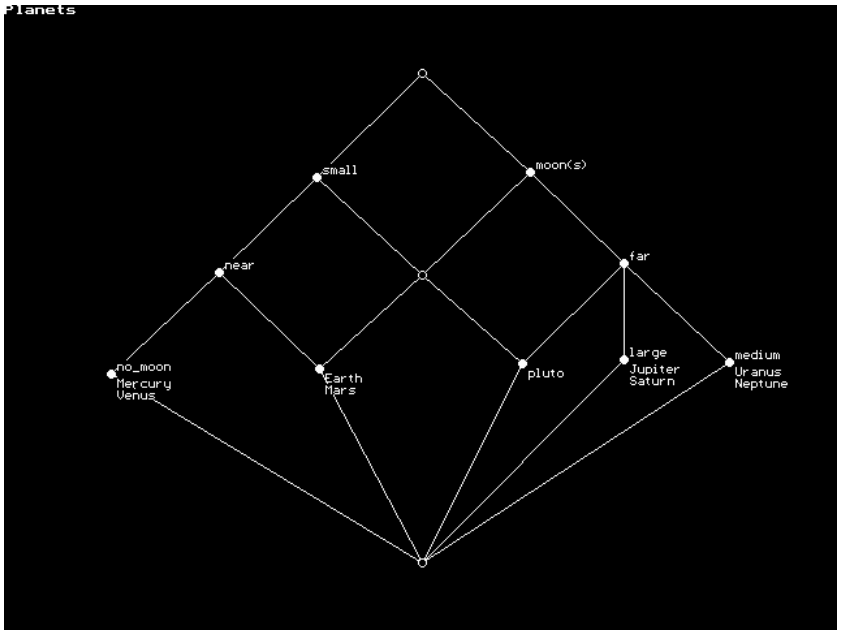


Figure 5.16: Screenshot of the planets example from Figure 1.2 rendered using *Diagram* — a DOS-based tool that supports additive line diagrams.

given context and this is computed and used extensively within ConImp. Interactive attribute exploration is supported which can be used to derive both the Duquenne-Guiges-base and a typical set of objects as described in Section 1.5.7. In addition, a three-valued logic that allows for *true, false* and *unknown* values can also be used.

The round-trip engineering work of Bojic and Velasevic [21] was discussed earlier in Section 2.3. By adapting the output from the Microsoft Visual C^{++} profiler they were able to use ConImp to analyse their data.

While ConImp supports single-valued contexts another tool called *MBA* (possibly from the German for "Many-valued FCA": *"**M**ehrwertige **B**egriffs**A**nalyse"*) can be used to scale and pre-process many-valued contexts [93]. In addition, contexts can be exported from ConImp in the so called "Burmeister Format" ('.CXT') and rendered using another DOS-based tool called *Diagram* [93]. Figure 5.16 presents a screenshot of Diagram. The use of separate tools for the tasks of data preparation, context creation, and line diagram rendering is also reflected in the classic FCA tools ANACONDA and TOSCANA.

5.2.3 ANACONDA and TOSCANA

ANACONDA and TOSCANA are tools used for building conceptual knowledge systems on top of data stored in relational databases. As Wille explains:

> The name "TOSCANA" (= **To**ols **s** of **C**oncept **Ana**lysis) was chosen to indicate that this management system allows us to implement conceptual landscapes of knowledge. In choosing just this name, the main reason was that Tuscany (Italian: TOSCANA) is viewed as the prototype of a cultural landscape which stimulated many important innovations and discoveries, and is rich in its diversity and attractive for wandering in [227].

Figure 5.17 presents an overview of the creation of a conceptual knowledge system as described by Becker and Hereth [16]. The process typically starts with the data to be analysed which is stored in a relational database. A conceptual system engineer uses knowledge from a domain expert to create queries in the form of conceptual scales using a *conceptual system editor*. These scales essentially capture the expert's knowledge and the information is stored in a *conceptual system file*. A user can then exploit the conceptual scales to retrieve or analyse data from the database using a *conceptual system browser*. In traditional TOSCANA systems ANACONDA is the conceptual system editor, TOSCANA is the conceptual system browser, and the data is stored in a Microsoft Access database.

ANACONDA is a tool for the creation and editing of contexts, line diagrams and scales. Figure 5.18 presents a screenshot of ANACONDA creating a version of the line diagram from Figure 3.1. The context, scales and line diagrams are saved in a conceptual schema file which is then used by TOSCANA to analyse the data in the database. While TOSCANA users cannot create new scales, the scales can be composed to produce nested line diagrams. There are three versions of TOSCANA based on Vogt's C++ FCA libraries [218, 89] and more recently a Java-based version — ToscanaJ.

5.2.4 ToscanaJ

ToscanaJ [15, 16] is a platform-independent implementation of TOSCANA that supports nested line diagrams, zooming and ideal/filter highlighting. Originally part of the Tockit

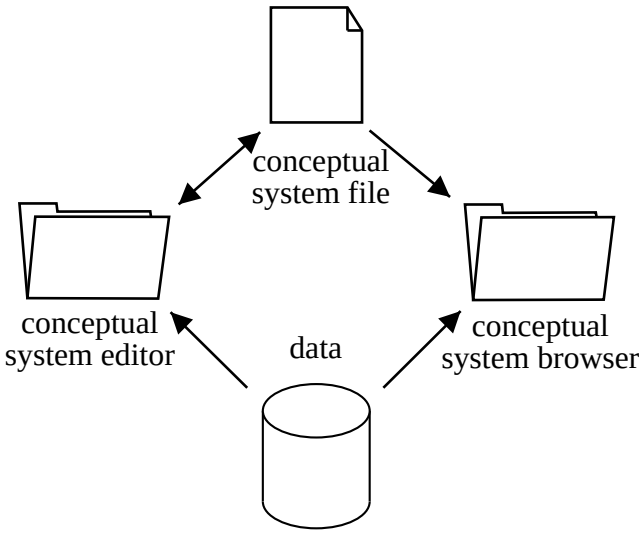


Figure 5.17: The TOSCANA workflow.

project [14] — an open source effort to produce a framework for conceptual knowledge processing in Java — ToscanaJ is now a separate project [15].

In the context of Figure 5.17, ToscanaJ represents the conceptual system browser while the conceptual system editor role is filled by two tools — *Siena* and *Elba*. The two tools can be seen as ANACONDA replacements that are both used for preparing contexts and scales, however, each represents a different workflow. Elba is used for building ToscanaJ systems on top of relational databases while Siena allows contexts to be defined using a simple point and click interface. In addition, Siena provides the facility to import data in ANACONDA '.CSC' format, ConImp's Burmeister '.CXT' format, and the XML export format from *Cernato*. The Cernato tool is introduced in Section 5.2.5.

ToscanaJ can be used to analyse data in relational databases via ODBC (Open Database Connectivity)/JDBC (Java Database Connectivity) or, alternatively, an embedded relational database engine within ToscanaJ can be used. Line diagrams can also be exported in a variety of raster and vector-based formats including Portable Network Graphics (PNG), Joint Photographic Expert Group (JPEG), Encapsulated PostScript (EPS), Portable Document Format (PDF), and Scalable Vector Graphics (SVG). Figure 5.19 shows two screenshots of the Siena editor and ToscanaJ and except where noted otherwise the line

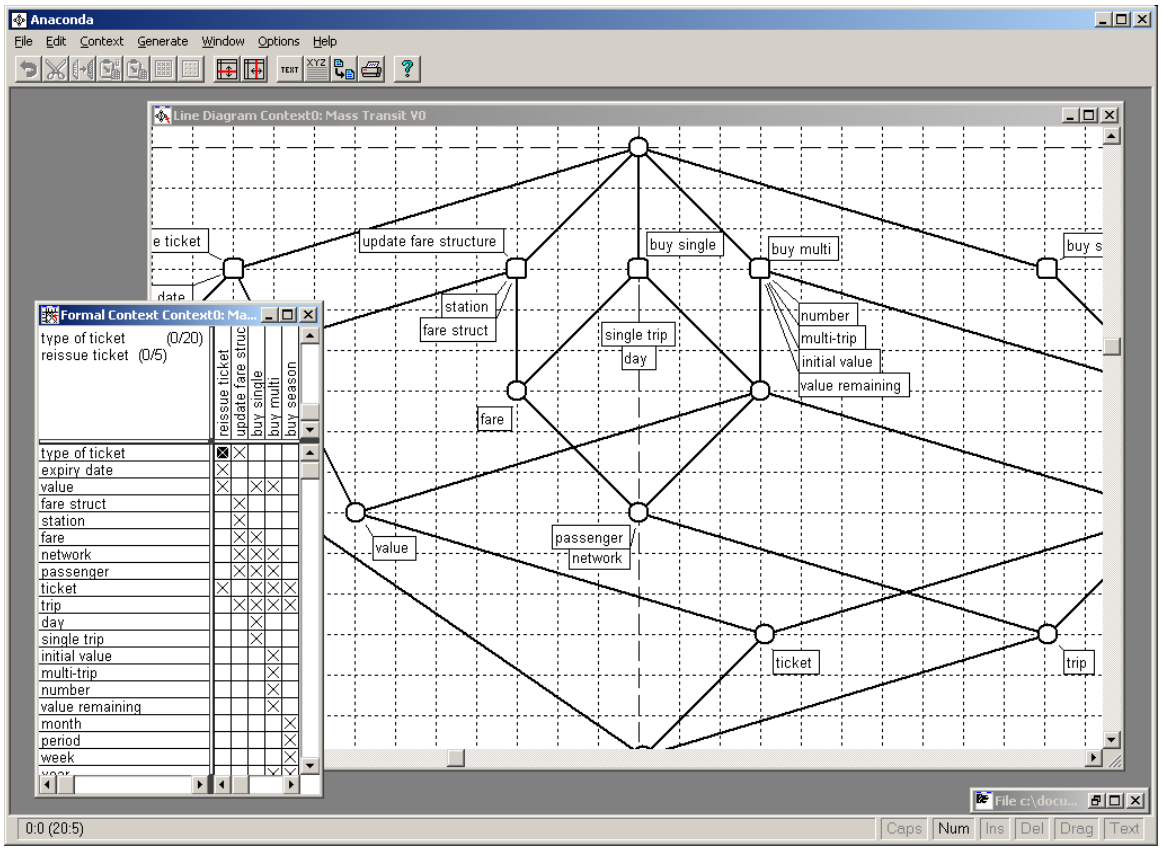


Figure 5.18: Screenshot of ANACONDA showing the formal context and line diagram windows. Note that the line diagram is the same as Figure 3.1 which was rendered using ToscanaJ.

diagrams in this thesis were produced using Siena and ToscanaJ.

An XML-based conceptual schema file (.CSX) is used to store the context and scales produced by Siena and Elba. In addition, an extensible viewer interface allows custom views within ToscanaJ to be defined as well as allowing external data viewers to be specified. The formal specification browser described in Section 5.3.3 makes use of this feature and further details are presented in that section.

The layout and manipulation of line diagrams in Siena and Elba is implemented using an *n*-dimensional layout algorithm in which each attribute in the purified context is assigned to a vector [13]. The layout is then projected onto the Cartesian plane using standard parallel projection and the approach is based on the algorithm used in Cernato.

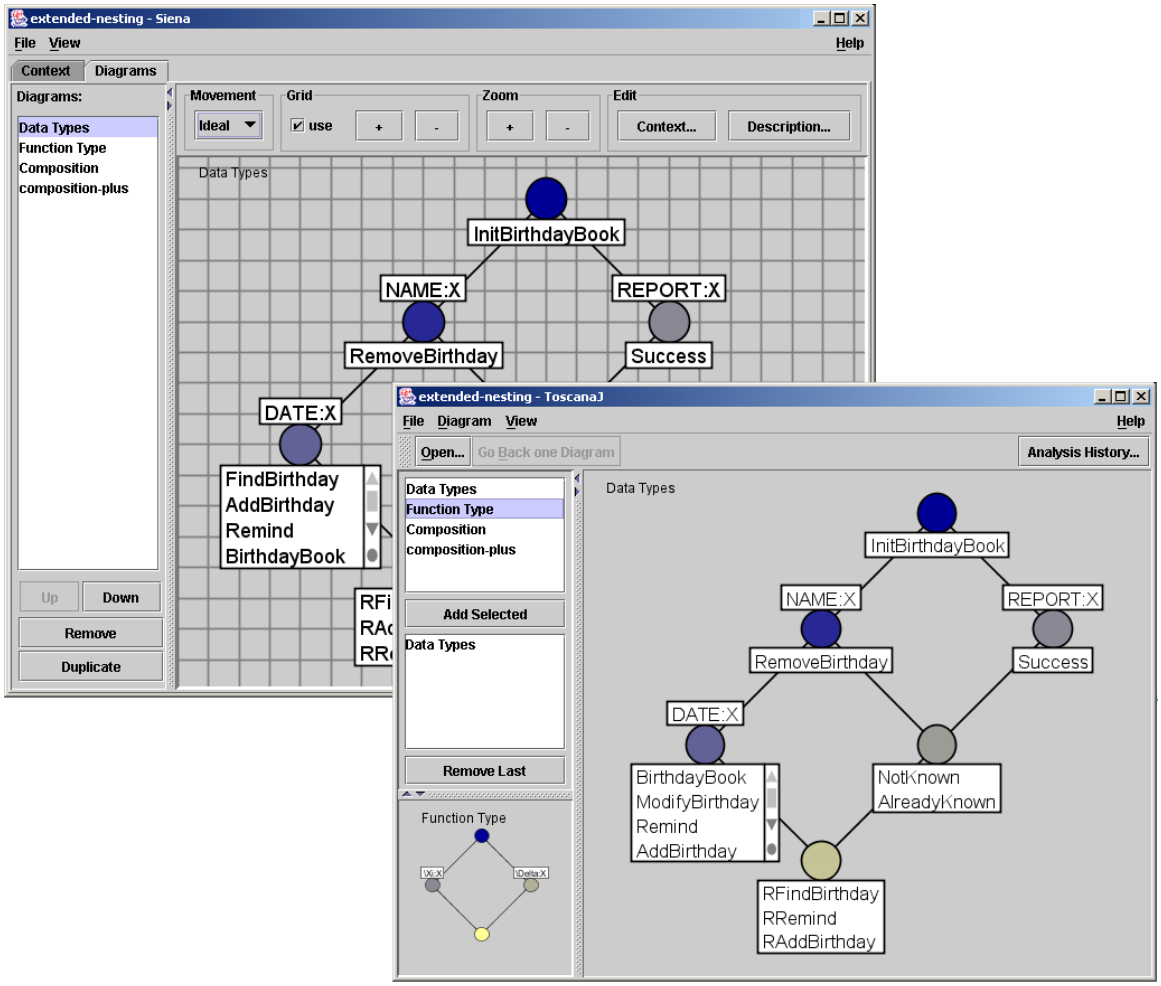


Figure 5.19: Two screenshots showing the Siena editor top left and ToscanaJ version 1.1 lower right. Note the diagram preview shown in the bottom left corner of the ToscanaJ screenshot which can be used to preview conceptual scales.

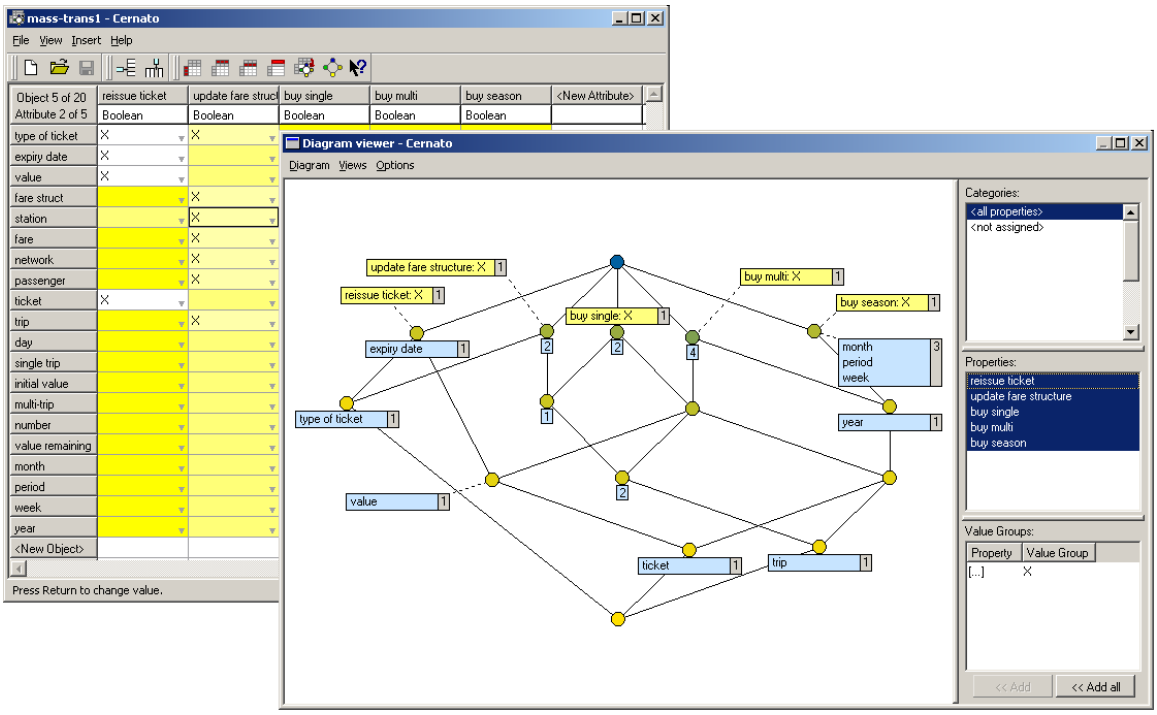


Figure 5.20: Screenshot of the Cernato context and line diagram windows. Note that the line diagram is the same as Figure 3.1 which was rendered using ToscanaJ.

5.2.5 Cernato

Cernato is a commercial FCA tool developed by Navicon [144] that combines some of the features of ANACONDA and TOSCANA into a single tool. Users are presented with a familiar spreadsheet-like interface for creating contexts and data can be imported and exported in Comma Separated Value (CSV) format which facilitates the analysis of data from genuine spreadsheet applications. A screenshot of the Cernato context editing and diagram windows is shown in Figure 5.20.

Line diagrams are constructed incrementally in Cernato and the layout is animated by default. Figure 4.14 in Chapter 4 depicts the animation of a line diagram in Cernato. Zooming and the construction of scales, which are known as “views” in Cernato, are also supported, however, nested line diagrams are not. In addition to the CSV import/export facility a custom XML format can also be used. Furthermore, line diagrams can be exported in a number of raster-based image formats, contexts can be saved as HTML tables and

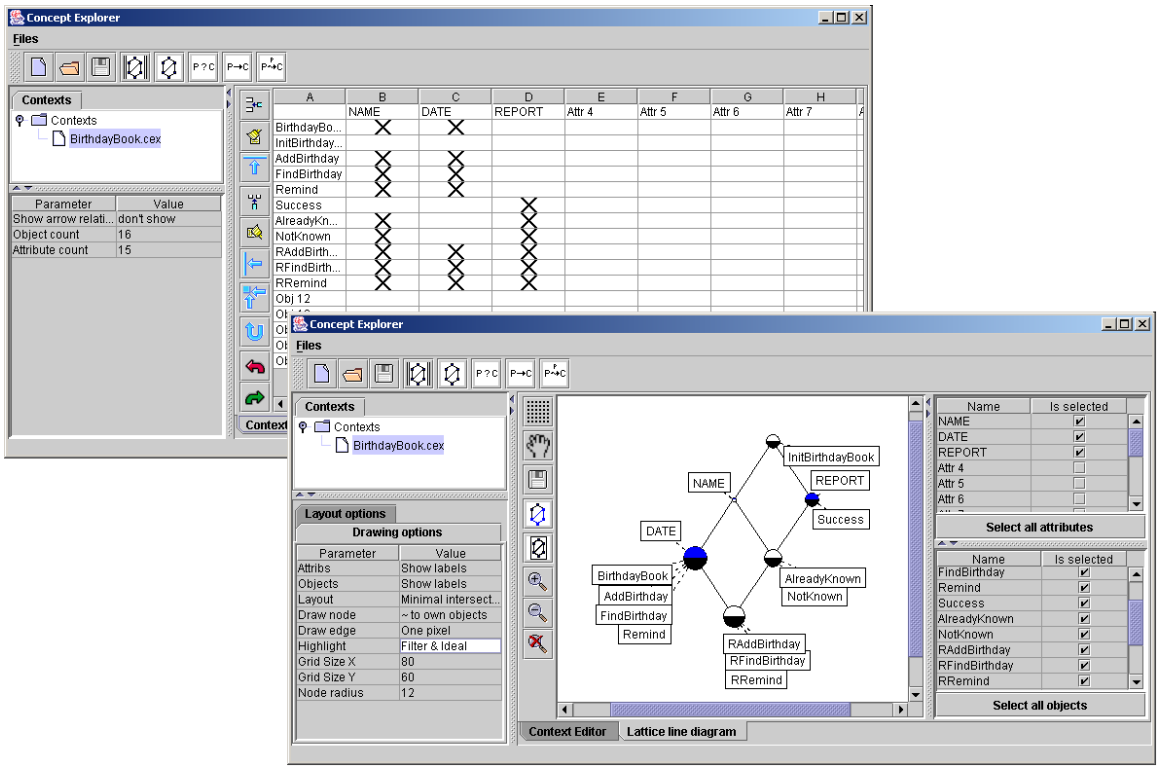


Figure 5.21: Two screenshots of ConExp showing the "context editor" pane (top) and the "lattice line diagram" pane (bottom). Note that the line diagram in the lower image is the same as Figure 4.3 which was rendered using ToscanaJ.

Cernato is also able to export complete TOSCANA systems.

5.2.6 ConExp

ConExp (**Con**cept **Exp**lorer) [234] is another Java-based, open-source FCA project. Like Cernato, ConExp combines context creation and visualisation into a single tool. Two views of the ConExp interface are shown in Figure 5.21. The "context editor" pane is shown at the top while the "lattice line diagram" pane appears at the bottom. Note that the line diagram corresponds to Figure 4.3 which was rendered using ToscanaJ.

While ConExp does not support database connectivity, contexts can be imported and exported in ConImp's '.CXT' format. A number of lattice layout algorithms can be selected including chain decomposition and spring-force algorithms. The line diagrams also support various forms of highlighting including ideal, filter, neighbour and single

concept highlighting and can be exported in JPEG or GIF format.

ConExp currently implements the largest set of operations from Ganter and Wille's FCA book [78] including calculation of association rules and the Duquenne-Guiges-base of implications. The context editor can display the arrow relations $g \nearrow m$ and $g \swarrow m$, and interactive attribute exploration is also supported.

5.2.7 IMPEX

IMPEX is a DOS-based tool that also provides attribute exploration. It is based on algorithms by Ganter [75] and it can calculate implications with background knowledge either automatically or interactively. In addition to a custom '.DAT' format for reading contexts and implications IMPEX can also import and export contexts in '.CXT' format. All output is written to text files which can then be viewed using an inbuilt text-editor.

5.2.8 GaLicia

GaLicia, the Galois Lattice Interactive Constructor [212, 213], is another Java-based FCA tool that provides both context creation and visualisation facilities. GaLicia's heritage lies in a series of incremental data mining algorithms originally entitled the **GA**LOIS **L**ATTICE-BASED **I**NCREMENTAL **C**LOSED **I**TEMSET **A**PPROACH and also a *trie* data-structure-based version called GALICIA-T. These incremental algorithms were used for mining association rules in transaction databases [215, 214] and form the basis for the incremental construction of lattices in GaLicia.

Both single and many-valued contexts can be analysed in GaLicia. In addition, binary relationships between objects can also be described via a context and stored using GaLicia's Relational Context Family ('.RCF') format [97]. These inter-object relationships can be used to produce views like Figure 4.6 showing the relationships between schemas. A number of different lattice and Galois sub-hierarchy construction algorithms are also supported.

GaLicia provides two lattice layout mechanisms including a "magnetic" spring-force algorithm. The lattices can also be viewed using a novel, rotating 3-Dimensional view

	System	Concept listing	Context Scaling	Attribute Exploration	Implications	Association rules	Context Editor	Diagram editor	Context Reduction	Database access	Diagram View	Diagram Printing	Further features	File conversion
ConImp	DOS, Atari (Linux)	×		extended	×		three-valued		×				×	
MBA	DOS, Atari		×				many-valued							
Diagram	DOS										×	DOS		
Cernato	Windows 9x/NT						many-valued				×			
Concept Explorer	Java 2			basic	×	×	binary		×		×			
Anaconda	Windows 9x/NT, Atari						binary	×	×		×	Win	×	
Toscana3	Windows 9x/NT									ODBC	×	Win	×	
ToscanaJ	Java 2									JDBC	×	Java	×	
CXT2CSC	DOS (Linux)													×
CSC2CSX	DOS (Linux)													×

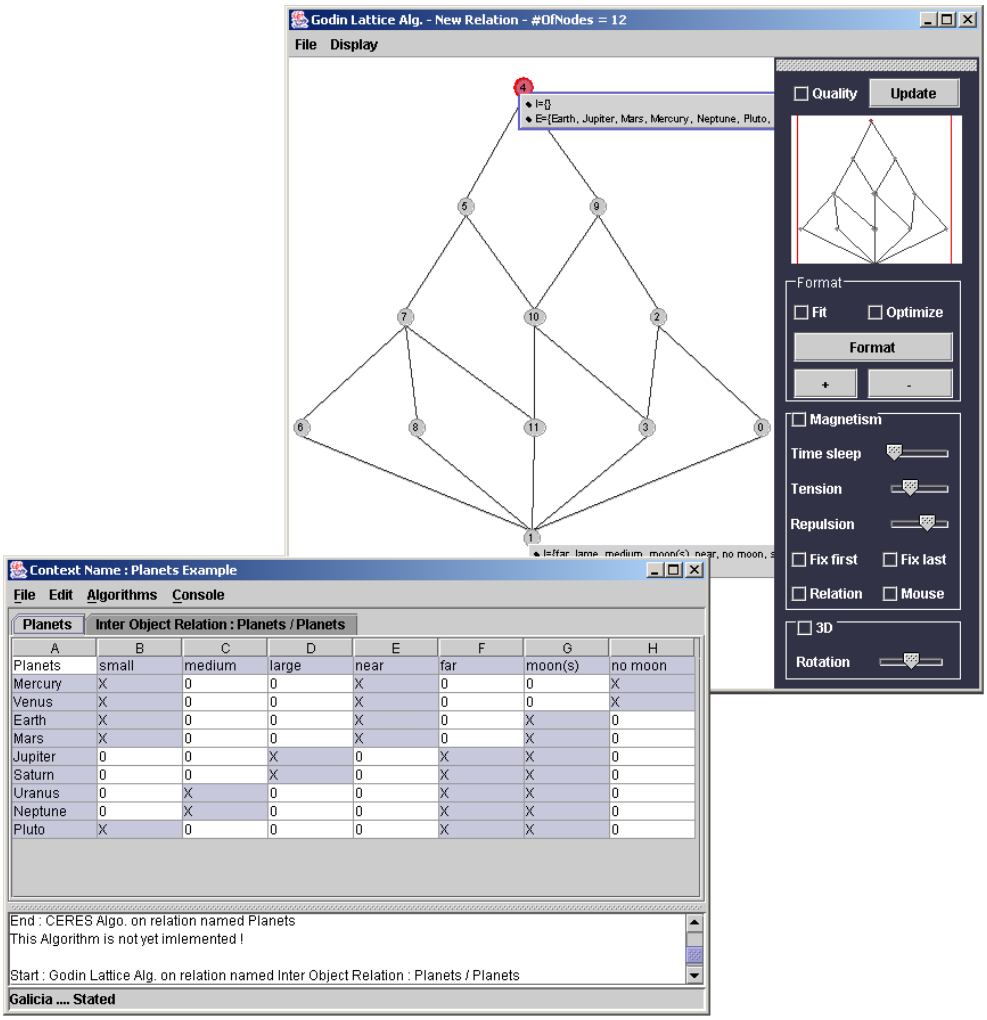
Table 5.1: Multi-valued context summarising tool features from Plüshcke's web-site [153].

where the nodes are laid out on the surface of a sphere. GaLicia can be run as a stand-alone application or it can be used via the Web as a Java applet running in a web-browser. Figure 5.22 presents a screenshot of GaLicia running as a stand-alone application. The *trellis*, or lattice, window is shown top right and the context editing window at the lower left.

5.2.9 Generic Tools Summary

While the preceding sections introduced the generic FCA tools this section now provides an overview of their features via a number of concept lattices. As a starting point Table 5.1 presents a context compiled by Plüschke as part of the requirements analysis process for the Tockit project [153]. This multi-valued context summarises the features of a number of existing FCA tools. The derived one-valued context in Table 5.2 extends Table 5.1 to include all of the tools described in Sections 5.2.1 to 5.2.8. In addition the one-valued context also includes additional attributes for image export formats and the FCA abstractions introduced in Section 4.3 of Chapter 4.

An overview of the basic functionality provided by the tools is presented in Figure 5.23. The line diagram is based on a sub-context of Table 5.2 and summarises the tools' ability to edit, view and print diagrams, edit contexts, access a database, and export images. Note



A	B	C	D	E	F	G	H
Planets	small	medium	large	near	far	moon(s)	no moon
Mercury	X	0	0	X	0	0	X
Venus	X	0	0	X	0	0	X
Earth	X	0	0	X	0	X	0
Mars	X	0	0	X	0	X	0
Jupiter	0	0	X	0	X	X	0
Saturn	0	0	X	0	X	X	0
Uranus	0	X	0	0	X	X	0
Neptune	0	X	0	0	X	X	0
Pluto	X	0	0	0	X	X	0

Figure 5.22: GaLicia screenshot showing a *trellis* (lattice) window top right and the context editing window lower left. The context corresponds to the planets example in Table 1.1 and the lattice to Figure 1.2. Also note the tab for a second context editing pane which can be used to describe binary relationships between the objects.

	Platform					Concept Listing	Scaling	Nested Line Diagrams	Zooming	Animated Layout	Attribute Exploration		Implications	Association Rules	Context Editor			Context Reduction	Diagram Editor	Diagram View	Diagram Printing	Database Access			Other Features	Image Export							
	Atari ST	DOS	Java 2	Linux	Windows						extended	basic			3-valued	*n*-valued	binary					ODBC	JDBC	internal		GIF	JPG	PNG	Other Raster	EMF/WMF	EPS	PDF	SVG
ANACONDA	×				×		×										×	×	×	×	×	×			×					×			
Cernato					×		×		×	×					×	×	×		×	×	×							×	×				
ConExp			×	×	×							×	×	×			×	×	×	×					×	×	×						
ConImp	×	×		×		×					×		×		×		×	×							×								
CSC2CSX		×		×																													
CXT2CSC		×		×																													
Diagram		×																	×	×	×												
Elba			×	×	×		×										×		×	×	×	×	×	×			×	×	×	×	×	×	×	×
GaLicia			×	×	×								×	×	×	×	×		×	×					×								
GLAD		×											×							×	×				×								
IMPEX		×									×		×												×								
MBA	×	×					×								×	×	×																
Siena			×	×	×												×		×	×	×						×	×	×	×	×	×	×	×
TOSCANA3					×		×	×	×											×	×	×			×					×				
ToscanaJ			×	×	×		×	×	×											×	×	×	×	×	×		×	×	×	×	×	×	×	×

Table 5.2: Derived one-valued context summarising features of the general-purpose FCA tools.

that the attributes for GLAD have been taken from the available literature rather than the tool itself so the summary of features may be incomplete.

At the top of the line diagram are three tools that do not provide any of these basic features: CSC2CSX, CXT2CSC and IMPEX. CXT2CSC and CSC2CSX, as their names suggest, are file format conversion tools. As such they are not concerned with context editing or the visualisation of diagrams. CXT2CSC converts contexts stored in '.CXT' format into the '.CSC' format used by ANACONDA and TOSCANA. Similarly, the CSC2CSX tool makes '.CSC' files accessible to ToscanaJ and the associated Elba and Siena editors. The third tool at the top of the diagram, IMPEX, is an implication and attribute exploration tool. It does provide simple text editing facilities, however, all of the other tools categorised with the "Context Editor" attribute provide functionality specifically for context editing.

The role of TOSCANA3 and ToscanaJ as viewing applications can also be observed in the line diagram. TOSCANA3 and ToscanaJ do not have context or diagram editing facilities because this functionality is provided by the corresponding ANACONDA, Elba and Siena editors. ANACONDA and Elba are also the only general-purpose tools that implement all

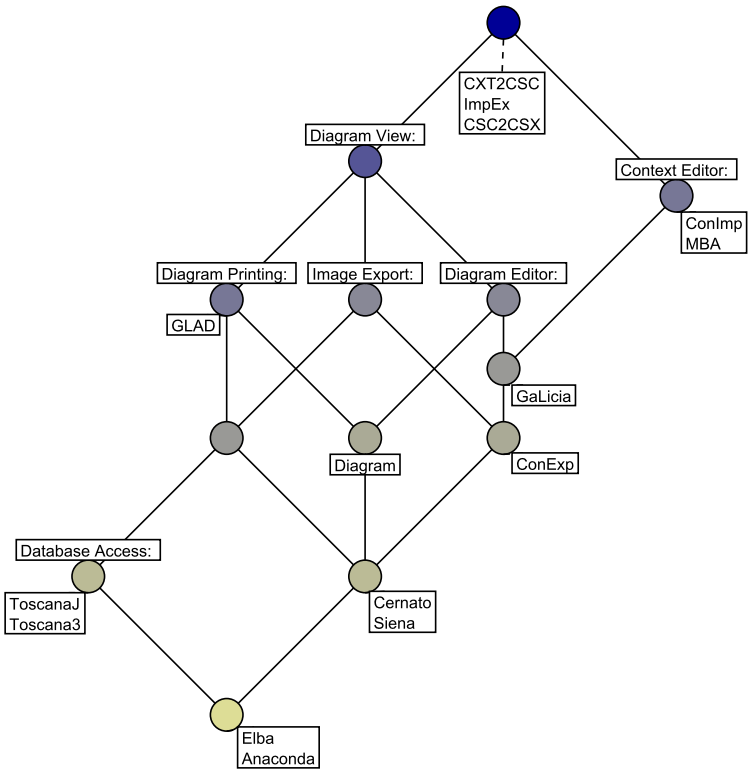


Figure 5.23: Line diagram of a sub-context from Table 5.2 summarising basic features of the generic tools.

of the basic features.

Figure 5.24 presents a line diagram based on a second sub-context of Table 5.2. This diagram provides an overview of the general-purpose tools based on functionality relevant to the discussion in Chapters 3 and 4: FCA abstractions, implication calculation, and attribute exploration.

From the diagram it can be seen that while Cernato, Toscana3 and ToscanaJ all support zooming and scaling there is no single general purpose tool that currently supports all four of the abstractions. Nested line diagrams are only available in Toscana3 and ToscanaJ while Cernato is the only tool that animates the layout of line diagrams. There is also no intersection between tools that support the abstraction mechanisms and those that calculate implications or provide attribute exploration.

Another one-valued context describing the file formats read and written by the generic FCA tools is presented in Table 5.3. Again, it should be noted that the feature summary

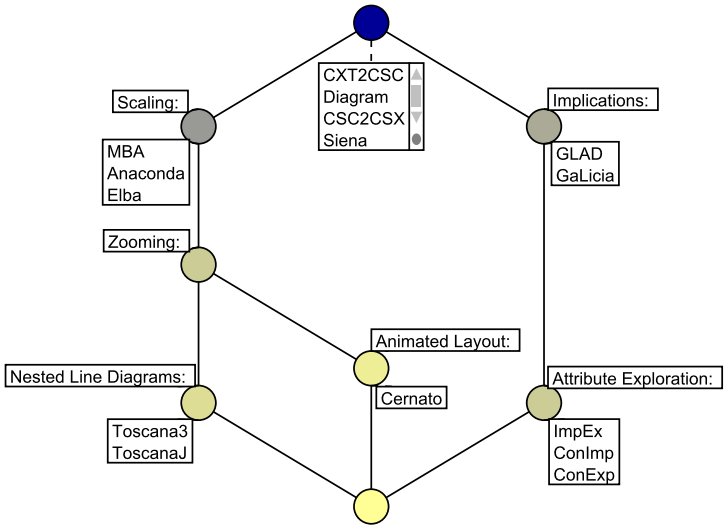


Figure 5.24: Line diagram summarising tool support for FCA abstractions, implications and attribute exploration.

of GLAD is taken from the available literature rather than the tool itself and there is no mention of the read/write formats used.

A line diagram representing the file formats read by the tools in Table 5.3 appears in Figure 5.25. There are three items of interest: the format supported by the most tools; the tool that supports the most formats; and the tool that shares the most formats with other tools.

The format supported by the most tools is the '.CXT' or Burmeister format originally used in ConImp. This text-based format stores the details of a binary context and can be read by seven of the tools described here. The wide adoption of the format is likely due to a number of factors including its simplicity, the obvious need for FCA tools to store and share contexts, and the fact that it was already in existence and being used when the other tools were created.

Beyond one-valued contexts, however, tools also need to store and retrieve a range of information that includes lists of implications, multi-valued contexts and diagram layouts. For example, Diagram relies on other tools to provide contexts in '.CXT' format. It creates line diagrams and then saves the concept lattice layout details in a '.LAT' file. This is the case for many of the tools which use their own custom formats in addition to '.CXT' for

Read File Format

	CXT	BGR	IMP	DUQ	MIN	ECH	HIN	MBA	MWK	SCH	LAT (Diagram)	CTB	CSV	Cernato XML	CSC	CS3	CEX	OAL	CSX	DAT	SLF	TXT (IBM format)	BIN.XML	MVC	MVC.XML	RCF	RCF.XML	LAT	LAT.XML
ANACONDA															×														
Cernato												×	×	×															
ConExp	×												×				×	×											
ConImp	×	×	×	×	×	×	×	×																					
CSC2CSX															×														
CXT2CSC	×																												
Diagram	×										×																		
Elba															×				×										
GaLicia																					×	×	×	×	×	×	×	×	×
GLAD																													
IMPEX	×																			×									
MBA	×								×	×																			
Siena	×													×	×				×										
TOSCANA3															×	×													
ToscanaJ																			×										

Write File Format

	CXT	BGR	DUQ	MIN	ECH	HIN	MBA	MWK	SCH	LAT	CTB	CSV	Cernato XML	HTML	CSC	CEX	CSX	DAT	UNI	IMP	IMX	SLF	TXT(IBM format)	BIN.XML	MVC	MVC.XML	RCF	RCF.XML	LAT (GaLicia)	LAT.XML
ANACONDA															×															
Cernato											×	×	×	×	×															
ConExp	×															×														
ConImp	×	×	×	×	×	×	×																							
CSC2CSX																	×													
CXT2CSC															×															
Diagram										×																				
Elba																	×													
GaLicia																						×	×	×	×	×	×	×	×	×
GLAD																														
IMPEX	×																	×	×	×	×									
MBA	×							×	×																					
Siena																	×													
TOSCANA3															×															
ToscanaJ																														

Table 5.3: Derived one-valued context showing the file formats used by the generic tools.

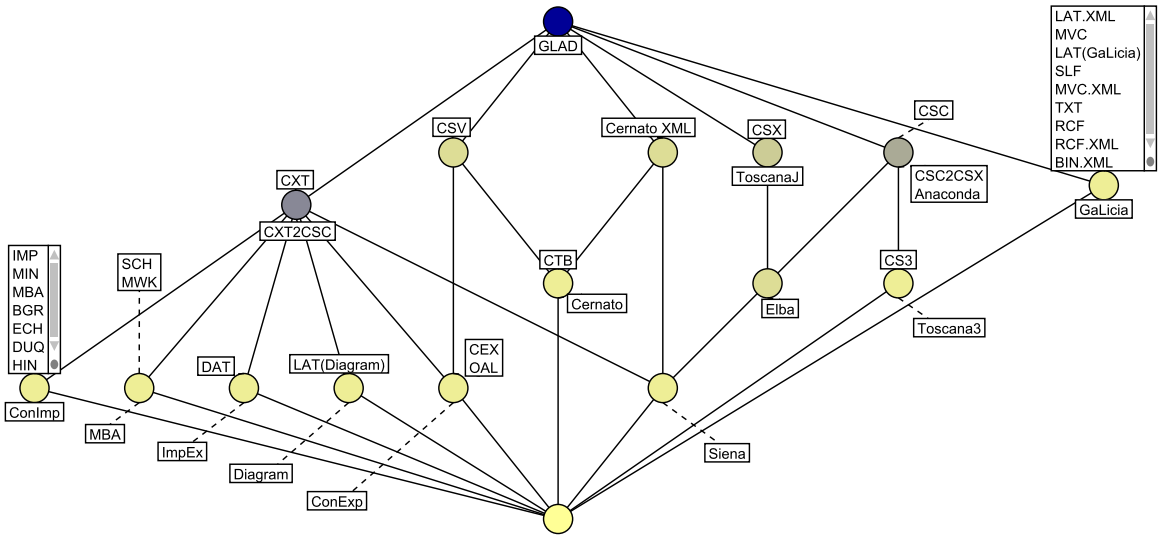


Figure 5.25: Concept lattice based on a sub-context of Table 5.3 showing the file formats read by the generic FCA tools.

storing information. Ultimately this results in the surprisingly large number of formats present in Table 5.3 where there is much "re-invention of the wheel". The need for a suitably flexible yet standard and therefore interoperable extension of the '.CXT' format for storing contexts and diagrams has been discussed within the Tockit project community [17].

While '.CXT' is the most widely read format, GaLicia is the tool that reads the most formats. GaLicia reads and writes nine formats, however, four of these are simply XML versions of other GaLicia formats. For example, '.RCF.XML' is an XML version of the '.RCF' Relational Context Family format. In Figure 5.25 GaLicia also stands out as the only tool that does not share any formats in common with other tools — there is no file-level interoperability with any of the other tools. In contrast to GaLicia, Siena shares the widest intersection of formats with other tools.

5.2.10 Application Specific Tools

In addition to the generic tools described in the preceeding sections there are also a number of application-specific FCA tools. These tools can be broadly classified into two main groups: the *monolithic* and the *modular*. Tools that rely on other programs for part or

all of their functionality will be classified as “modular”. For example, a number of the application-specific tools make use of pre-existing graph drawing applications for lattice layout. In contrast the term “monolithic” will be used to describe those tools which do not rely on other applications to function. This does not, however, exclude the use of pre-existing libraries within the tools code. Additionally, the term should not infer that a tool is poorly engineered or necessarily massive, but rather that the tool has been constructed from scratch. The next section of this chapter introduces the monolithic tools and the modular tools are discussed in Section 5.2.10.

Monolithic Approaches

Düwel’s *BASE* [57] tool supports the identification of class candidates from use-cases using the methodology applied in Chapter 3. The name is taken from the German *“ein **B**egriffsbasiertes **A**nalyseverfahren für die **S**oftware-**E**ntwicklung”* which translates into English as “concept-based analysis during software development”.

Taran and Tkachev’s [197] tool SIZID is designed to support the analysis of sociological and psychological data. SIZID can handle multi-valued contexts and the calculation of implications.

Cole and Eklund have implemented a number of FCA-based document management and information retrieval tools. *Warp-9 FCA* [38] is a tool for managing a collection of medical discharge documents that is implemented using the scripting and extension language Tcl/Tk [198]. A medical ontology is used to index documents and the visualisation supports folding line diagrams. The ideas in *Warp-9 FCA* are further refined and applied to the analysis of email in the tool CEM — the Conceptual Email Manager [38, 39]. More recently a commercial descendant of CEM known as *Mail-Sleuth* has also been released [66].

In Lindig and Snelting’s [132] paper on the structure of legacy code a footnote mentions an inference-based software environment called NORA which was used to produce the analyses described in the paper. NORA stands for “**NO R**eal **A**cronym”. While no details of the NORA environment are presented in the paper, both Snelting and Lindig have produced other tools to support the analysis of software using FCA. Snelting and

Streckenbach's *KABA* tool was briefly mentioned in Section 2.6.1. KABA is a Java-based tool that implements the analysis earlier described by Snelting and Tip [178, 179]. The name KABA is taken from the German "**K**lassen**A**nalyse mit **B**egriffs**A**nalyse" which translates as "class analysis via concept analysis" in English. Apparently "KABA" is also the name of a popular chocolate drink in Germany.

KABA combines concept lattices with dataflow analysis, and type inference. In particular the prototype tool supports the visualisation of horizontal decompositions in Java classes and a 15 KLOC example is reported.

While another prototype tool that implements Lindig's component retrieval ideas could be considered monolithic [128], there have been a number of modular tools developed using Lindig's *concepts* framework.

Modular Approaches

Concepts [131] is an updated version of Lindig's *TkConcept* tool [129, 130] which is implemented in Tcl/Tk. TkConcept is included here as an example of a modular tool because it makes use of a graph layout application called Graphplace [61] to draw lattice diagrams. TkConcept was intended as a framework for concept analysis applications that provides basic abstractions so that software designers can focus on the implementation of domain specific parts of an application.

Van Deursen and Kuipers [216] used Lindig's *concepts* tool in conjunction with Graphplace in the analysis of a 100 KLOC COBOL program. A relational database was used to derive information about the application using a COBOL lexical analysis tool. The data was then extracted and formatted for analysis with *concepts*.

The *ConceptRefinery* tool described by Kuipers and Moonen [123] also uses *concepts* in conjunction with a COBOL parser and a relational database. Concept refinery is implemented using Tcl/Tk and a version of the *dot* directed graph drawing tool was used for visualisation. Dot is part of the GraphViz graph visualisation package [10].

GraphViz and concepts are also used to render lattice diagrams in Eisenbarth, Koschke and Simon's *Bauhaus* tool [64]. Bauhaus makes use of a number of components including the *gcc* compiler and *gprof* profiler which are glued together using Perl [151]. In addition

to their earlier work identifying features in web-browser code, Eisenbarth et al. have also used their tool to analyse a 1,200 KLOC production system [63, 62].

The *Cable* tool implemented by Ammons et al. makes use of FCA to aid in the debugging of temporal specifications [5]. The visualisations presented to Cable users are implemented using the Dotty and Grappa graph visualisation tools which are also part of GraphViz.

JaLaBA is a novel on-line **Ja**va **La**ttice **B**uilding **A**pplication implemented by Janssen [109] that uses Freese's *LatDraw* [73] program for lattice layout. LatDraw makes use of a 3-dimensional spring and force layout algorithm which produces line diagrams similar to GaLicia and ConExp.

The round-trip engineering work of Bojic and Velasevic [21] discussed earlier in Section 2.3 clearly meets the definition of a modular tool. By adapting the output from the Microsoft Visual C++ profiler ConImp was able to analyse their data which was then used to update a UML model using the Rational Rose design tool [98].

Richards and Boettger et al.'s RECOCASE tool [24] is also comprised of a number of other applications. RECOCASE uses the Link Grammar Parser [173, 172] to parse use-cases and ExtrAns [170, 169] is used to generate the flat logical forms which are then analysed using FCA.

The CANTO tool (**C**ode and **A**rchitecture a**N**alysis **TO**ol) [7] described by Tonella [203] has a modular architecture composed of several subsystems. CANTO consists of a front-end for analysing C code, an architecture recovery tool, a flow analysis tool and a customised editor. The components communicate either via sockets or files and apart from the flow analysis tool each of the components is an external application. Visualisations produced by the architecture recovery tool are created using PROVIS — yet another graph drawing application based on Dotty.

Another FCA framework implemented by Arévalo [8, 9] and Buchli [31] is ConAn (**Con**cept **An**alysis) [31]. ConAn is implemented in Smalltalk and consists of a number of tools for the creation and analysis of formal contexts. A tool called *ConAn PaDi* (ConAn **Pa**ttern **Di**splayer) built using the ConAn framework is used for analysing patterns in data

from the *Moose* Smalltalk re-engineering environment [1]. Beyond software engineering applications ConAn also represents a generic and extensible framework. Users can provide objects and attributes (known respectively as *elements* and *properties*) as labels in a table or custom Smalltalk objects can be implemented to represent the elements and properties used by ConAn.

While the preceding sections introduced a range of FCA-based tools the next section describes the implementation of a new tool that mirrors the modular approach taken by Van Deursen and Kuipers. This tool makes use of a parser to extract information which is stored in a database. The information is then formatted for analysis using an external visualisation application. The tool is called SpecTrE — the **Spec**ification and **Tr**ansformation **E**ngine.

5.3 Specification Transformation Engine (SpecTrE)

SpecTrE is a tool for visualising and navigating Z specifications using FCA that implements the ideas presented in Chapter 4. In the tradition of the modular tools described in the previous section, the implementation of SpecTrE makes use of a number of discrete tools and existing applications.

The SpecTrE tool performs two main functions the first of which is the transformation of Z specifications written in Oz style LaTeX into ZML. The second function is the creation of a formal context from the specification which can then be used to visualise and navigate the specification. Both the LaTeX to ZML transformation and the context creation processes can be performed in times comparable with normal LaTeX processing. This is consistent with Clarke and Wing's call for Formal Methods tools that work in times comparable with compilation [37]. An overview of the process starting with a LaTeX source specification is presented in Figure 5.26.

The original LaTeX specification is first transformed into ZML using an application called `tex2zml`. Once in ZML format the specification can be rendered on demand in a web-browser using the appropriate XSL stylesheet. This ZML version of the specification is also used as input to another tool called `spec2db` which parses the specification and creates a formal context which is stored in a relational database. A conceptual schema file

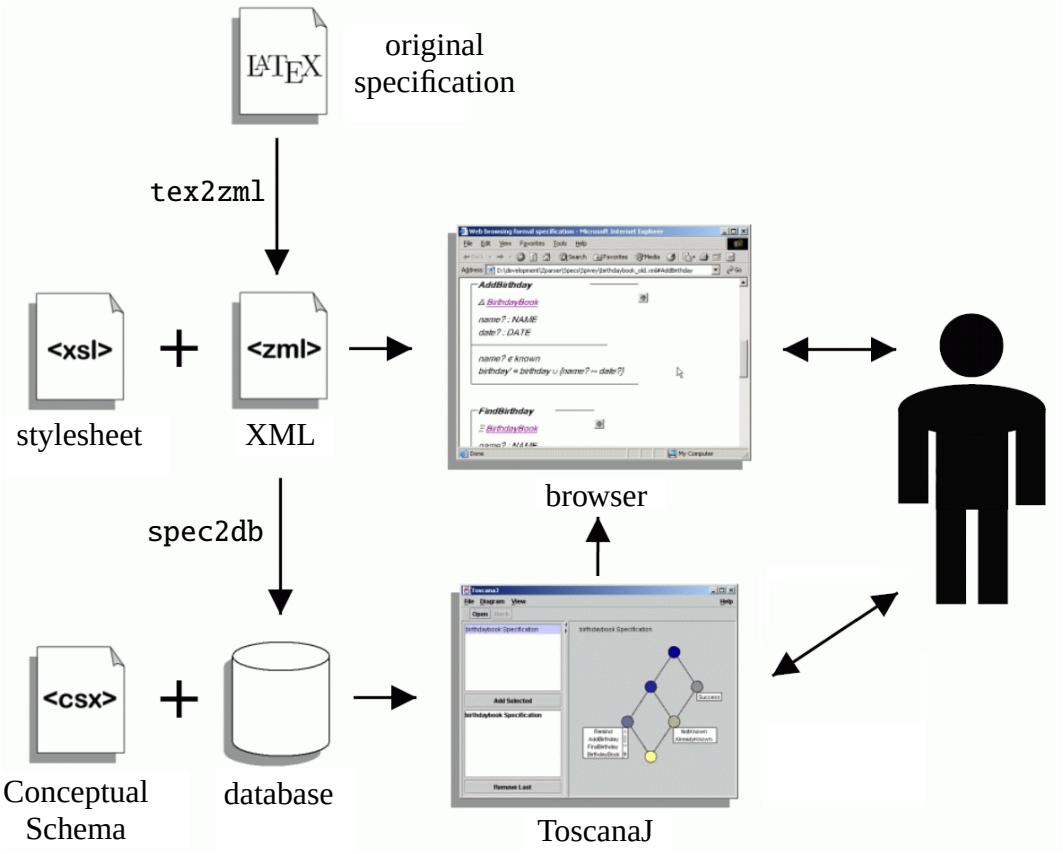


Figure 5.26: Overview of the specification transformation and exploration process using SpecTrE.

describing the structure of the context is also created.

The conceptual schema file created by `spec2db` can be opened using either the Siena or Elba editors for ToscanaJ and suitable scales can then be constructed. ToscanaJ can then be used to visualise and interactively navigate the specification via the application of conceptual scales. If a user wishes to view the original specification they can click on a schema name in the line diagram and a web-browser will then be launched to display the appropriate schema using the ZML version of the specification. This implementation fulfils the aim of providing an FCA-based alternate visual representation for formal specifications that can be used alongside existing tools. Users can explore a specification using FCA and then "drill down" into the original specification as required.

The implementation details of the `tex2zml` transformation and `spec2db` context

```
...

<rule id="41" type="replace">
    <old>\union</old>
    <new>&uni;</new>
</rule>
<rule id="42" type="replace">
    <old>\uni</old>
    <new>&uni;</new>
</rule>

...

<rule id="149" type="pattern" >
    <old>\\Delta\s+([\w]+)\s+(\\\\)?</old>
    <new>    &lt;del&gt;
        &lt;type&gt;$1&lt;/type&gt;
    &lt;/del&gt;</new>
</rule>

...
```

Figure 5.27: Three of the transformation rules used to translate specifications written using Oz LaTeX mark-up into ZML format.

creation tools are discussed in Sections 5.3.1 and 5.3.2 respectively. Issues relating to the web-browser integration with ToscanaJ for viewing specifications are discussed in Section 5.3.3 and a Graphical User Interface (GUI) front-end is described in Section 5.3.4.

5.3.1 Specification Transformation

The implementation of the `tex2zml` tool relies on the fact that the tag names in ZML were chosen to directly correspond with names used in the Oz style LaTeX mark-up. The tool works by applying a series of transformation rules to the original specification until no further transformations are possible. The rules are defined in an XML file and two types of rules are used. See Figure 5.27.

Replacement rules are used to match whole words that are directly substituted. For example, rules 41 and 42 in Figure 5.27 match on either of the two mark-ups for the set union operator `\union` or `\uni` which are then replaced with `&uni;`.

The second type of rule uses patterns to match and transform structures using regular expressions. For example, rule 149 would match on a line containing the LaTeX mark-up

for a ‘Δ’ schema like “`\Delta BirthdayBook \\`” and replace it with:

```
<del>
    <type>BirthdayBook</type>
</del>
```

The “id” numbers within the `<rule>` tags are used for debugging purposes and the pattern rules are implemented using the GNU regexp library for Java [18]. While the replacement rules could also be written as pattern rules the “replace” rule type is included for convenience and readability.

ZML was used for the prototype implementation of SpecTrE because the XSL transformation for browser rendering is performed automatically in XSLT-enabled browsers. Server-side XSLT processing can also be used to support non-XSLT browsers. The intra-specification hyperlinks are also automatically created and the HTML anchors can be exploited by other applications to display any schema within the specification. Additionally, ZML supports the automatic expansion of horizontal schemas and standard XML parsers can be used to read and validate ZML files.

While the straightforward transformation approach described above works for the original version of ZML the more recent annotated syntax versions would require a genuine parser. Tools that already support existing XML representations for Z offer a possible transformation path. The current XML export from CADiℤ is very similar to the new ZML annotated syntax [211]. Sun also reports that a LaTeX to XML translator is either under development or has been developed by the Formal Specification Research Group at the National University of Singapore [193]. Given that SpecTrE has been effectively implemented as a chain of tools then `tex2zml` could simply be replaced by another tool as required. Additionally, SpecTrE can also accept existing ZML files as input which avoids the need for translation. Once in ZML format, however, a context can be created using `spec2db`.

5.3.2 Database and Context Creation

The `spec2db` tool takes a ZML specification and creates a formal context which is stored in a relational database. Given that ZML is XML-based then a generic XML parser can

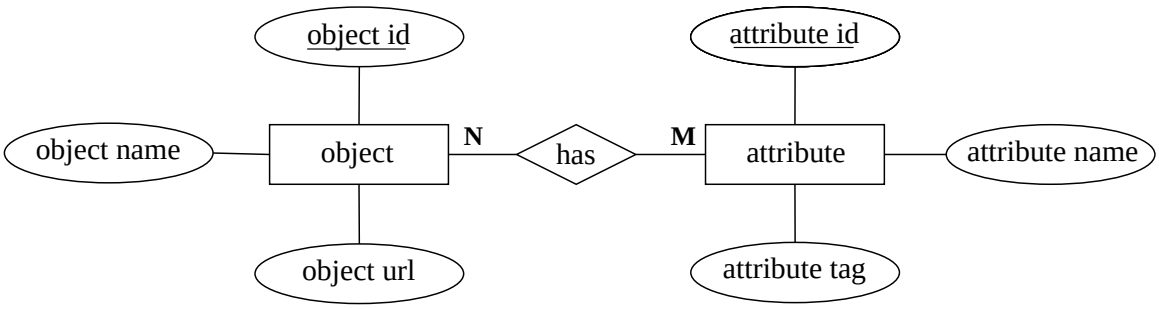


Figure 5.28: ER Diagram representing the database structure for storing the object and attribute information extracted from the specifications.

be used to read the specification files rather than implementing a custom-built parser. The Electric XML API [140] used to parse the transformation rules described in Section 5.3.1 above is also re-used in `spec2db` to parse ZML.

The structure of the database used to store details of the parsed specification consists of three tables reflecting the G, M, I structure of the formal context. An Entity-Relationship (ER) diagram representing the table structure is shown in Figure 5.28 using the notation of Elmasri and Navathe [65].

While the correspondence between the *object* and *attribute* entities with the sets G and M is obvious, the incidence relation I is represented by the "has" relationship. In the relational database model this relationship is also implemented as a table. In addition, note that there are no participation constraints on the relationship corresponding to the incidence relation I. This structure permits objects to be included in the context that have no attributes and conversely, attributes that do not belong to any of the objects. In specification terms this may be useful to explicitly represent the fact that a specification does not make use of a particular mark-up element or operation. This functionality is further supported by the "attribute" and "min attribute" files described in Section 5.3.4. The "attribute" file can be used to exclude specific mark-up elements from the context while the "min attribute" file can be used to include specific attributes even if they are not present in the specification.

Although this three table structure explicitly represents the object set, attribute set, and incidence relation, ToscanaJ actually requires a single database table that more closely resembles a crosstable. This structure is depicted in Figure 5.29.

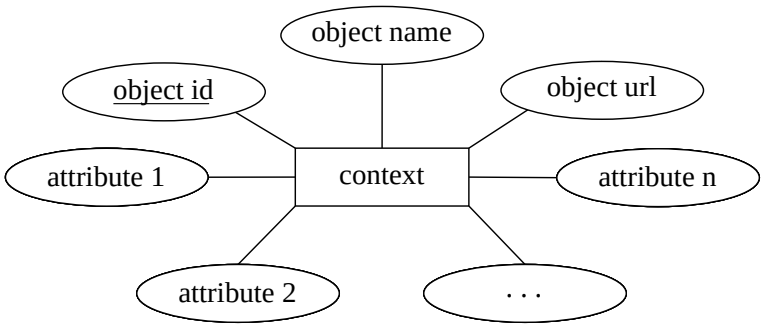


Figure 5.29: ER Diagram representing the single table context used by ToscanaJ.

Representing a formal context as a single database table presents a number of problems when encoding specification details. The first problem arises because the Z notation is case-sensitive. In the three table G, M, I database structure the object and attribute names are both stored as string values which are also case-sensitive. However, in the single-table structure used to create the CSX file the attribute names become column names within a database table. Column names in databases adhering to the SQL-92 standard are not case-sensitive, although there appears to be some variation depending upon the choice of database management system and/or operating system platform.

The second problem arises because a number of quotation characters like '?' and '!' which are commonly used in Z are illegal in database column names. In most databases it is possible to use "illegal" column names by specifying them within double quotes, however, this feature also appears to vary with the choice of database. Reliably using attribute values as database column names therefore requires an encoding scheme that preserves case as well as any illegal characters. This is achieved within `spec2db` by replacing any non-lowercase alphabet character in an attribute name by a literal representation of its Unicode value. For example, the uppercase letter 'A' is encoded as 'u0041', 'Z' as 'u005a', and '!' as 'u0021'. Using this scheme the attribute name *BirthdayBook* would appear in the context table as the column `u0042irthdayu0042ook`. The encoded names need only be machine readable because the CSX file stores attribute labels separately from the definition used to query the database for the details of a specific formal concept. The encoded names are only used for interacting with the database and the original attribute names can still be

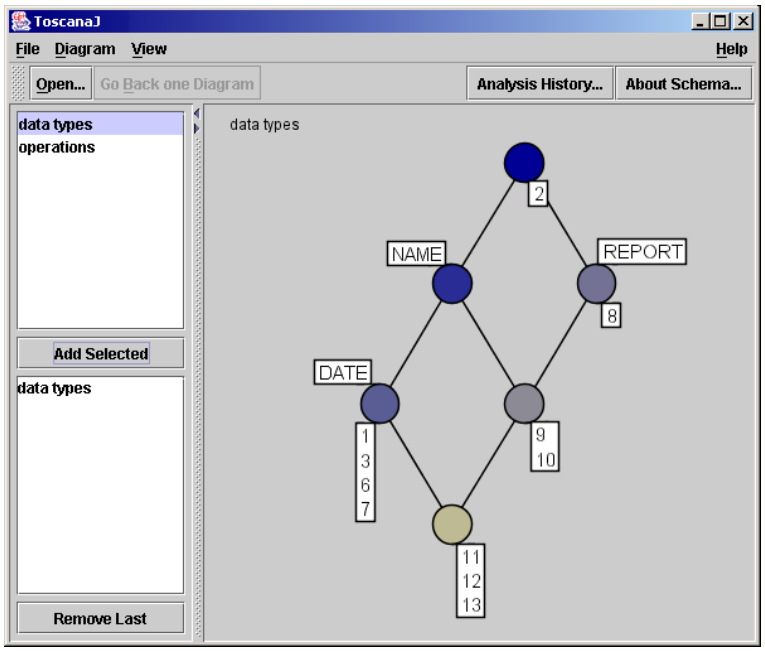


Figure 5.30: ToscanaJ screenshot showing the standard "list object" view. Note that the labels represent the database *object_id* numbers rather than an object count.

```
<queries dropDefaults="false"> <listQuery name="Schema
    Names" head=""> <column
    name="Schema">object_name</column> </listQuery>
</queries>
```

Figure 5.31: This list-query produces the menu option to display schema names shown in Figure 5.32.

displayed on the line diagrams.

A final point regarding the construction of the database and conceptual schema files also concerns the structure represented by Figure 5.29. Given that the primary key from the final database context table is *object_id* then the standard "list object" view in ToscanaJ simply lists the *object_id* numbers rather than the schema names as shown in Figure 5.30. A "schema name" option is added to the menu via the list query shown in Figure 5.31 which is also stored in the CSX file. If this new option is selected the database is queried and the *object_name* values (the actual schema names) are displayed instead of the *object_id* numbers. The resulting menu option is shown in Figure 5.32.

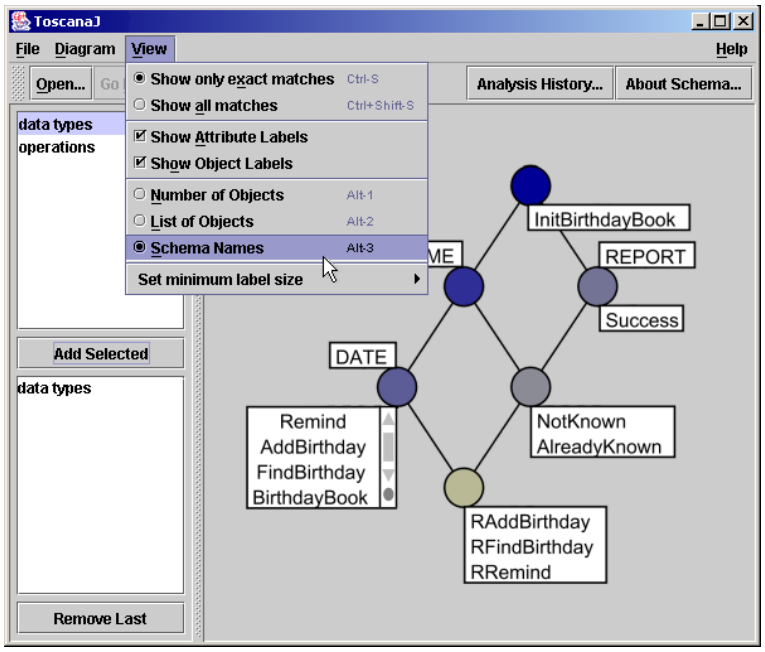


Figure 5.32: ToscanaJ screenshot showing the “schema name” menu item.

In addition to parsing the specification and creating the database tables the `spec2db` tool is also responsible for creating the conceptual schema file. Although this file is created in the CSC format used by ANACONDA it can be imported into either the Siena or Elba editors to create the CSX file which is ultimately used by ToscanaJ. Siena and Elba can also be used to create any predefined conceptual scales that a user may wish to apply.

While the transformation and context creation processes are automated, the conceptual scales must currently be created manually before ToscanaJ can be used. Alternatively, a number of standard scales based on Z language features could be set up and re-used across projects while project specific scales can still be created as required.

Using standard lattice layouts and order embeddings it is also possible to automate scale layout to a certain extent. Scale creation then simply becomes a task of choosing the appropriate attributes. The choice of attributes could also be performed based on a particular ZML tag-type or a naming convention. An example of an automatically generated scale based on the schema input naming convention ‘?’ is shown in Figure 5.33. Once the scales have been constructed, however, ToscanaJ can then be used to explore the concept lattices representing the specification.

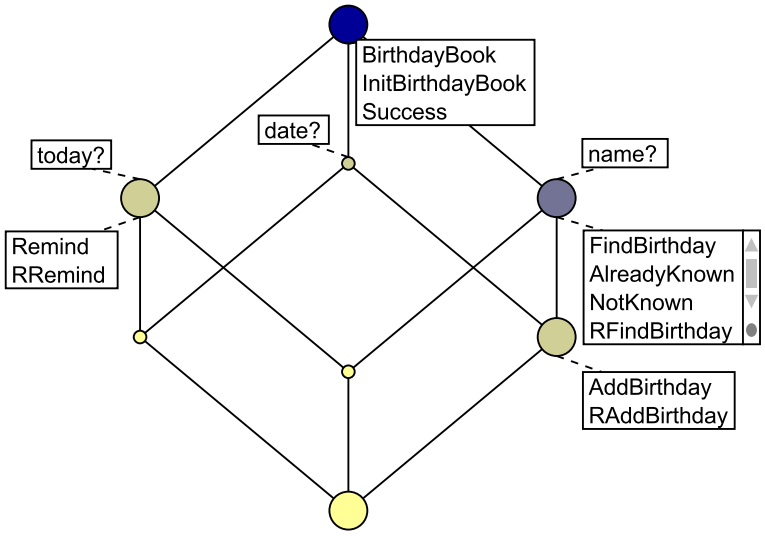


Figure 5.33: An automatically generated scale based on schema inputs.

5.3.3 Browser Integration

ToscanaJ was chosen to provide visualisation in SpecTrE because it supports a number of the required FCA abstractions including conceptual scaling, nested line diagrams and zooming. Furthermore, ToscanaJ supports database connectivity and the Java-based implementation means that the tool is platform independent. The source code is also freely available so the tool could be modified as required, however, the extensible "view" interface meant no hacking or re-compilation of ToscanaJ was necessary.

In addition to customisation via the `<listQuery>` tag described in the previous section, ToscanaJ also allows custom object views to be specified within the conceptual schema file. These views allow users to click on object labels in a line diagram and display additional information about the objects. This view interface can be exploited to display any schema within a Z specification by launching a browser with a URL that concatenates a reference to the ZML version of the specification and the object name as an HTML anchor.

On the Windows platform a shell execute is available to open documents using the default application based on the document's extension type. While this technique can be used to open a URL on the local filesystem like '`C:\Temp\Demo\BirthdayBook.xml`' it cannot be used to open '`C:\Temp\Demo\BirthdayBook.xml#AddBirthday`' because

```
<views>
   <objectView class="net.sourceforge.toscanaj.dbviewer.ProgramCallDatabaseViewer"
               name="Goto spec...">
       <parameter name="openDelimiter" value="%%%"/>
       <parameter name="closeDelimiter" value="$$$"/>
       <parameter name="commandLine"
                  value='rundll32 url.dll,FileProtocolHandler
                         javascript:location.href="%%%object_url$$$"'/>
   </objectView>
</views>
```

Figure 5.34: The Database Viewer code to implement ToscanaJ and browser integration from within a "CSX" file. The object view enables the pop-up menu shown in Figure 5.35 for displaying schemas within the browser.

the extension is not recognised. As a workaround for this problem the URL can be encapsulated within some Javascript and the default browser will then be launched as the default application. The Javascript simply opens the URL and the required <objectView> mark-up is shown in Figure 5.34.

The "name" attribute in the <objectView> tag adds a popup-menu option in ToscanaJ that can then be used to display the desired schema using the web-browser. A screenshot of the menu is shown in Figure 5.35. The value of the 'object_url' attribute between the delimiting tags $$$ and %%% is retrieved from the database for this object and substituted in the command line. The browser is then launched to display the appropriate schema which is rendered automatically using ZML and XSL [1].

The only side-effect of this approach is that it requires Javascript to be enabled on the user's browser which has an associated security risk on the Web at large. This implementation is also platform dependent; however, both the Netscape and Mozilla browsers on the Unix/Linux platform have a remote control facility that could be exploited in a similar fashion [141].

[1] An example CSX file generated from the *BirthdayBook* specification is available online: http://www.kvocentral.org/software/spectre.html

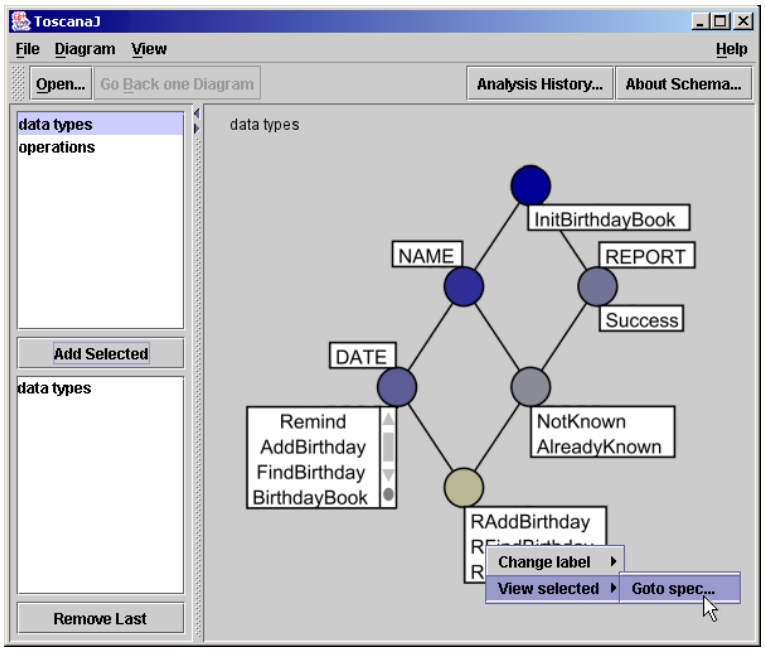


Figure 5.35: ToscanaJ screenshot showing the effect of a right mouse button click on a schema name. Selecting "*Goto spec...*" from the pop-up menu will launch a web-browser displaying this schema within the original specification.

5.3.4 GUI Front-end

Although SpecTrE is implemented via a series of independent tools, a GUI front-end is provided to assist users in the transformation and context creation process as shown in Figure 5.36. All of the available options can also be specified via a command line interface.

A "skinned" version of the SpecTrE front-end is also available as shown in Figure 5.37. This is activated via the "`-blofeld`" command line option and in this mode the interface also plays appropriate sound-samples in response to button click, startup, and exit events.

Via the GUI front-end a user can select a source specification written in either ZML or LaTeX. The database where the formal context will be stored must also be specified. In addition the user can also optionally select two files which contain lists of attributes to be included during the context creation process.

The "attribute" file provides a list of valid mark-up tags that are to be included in the context if they are found while parsing the source specification. Via this mechanism irrelevant mark-up tags can be ignored while different attribute lists could also be used to

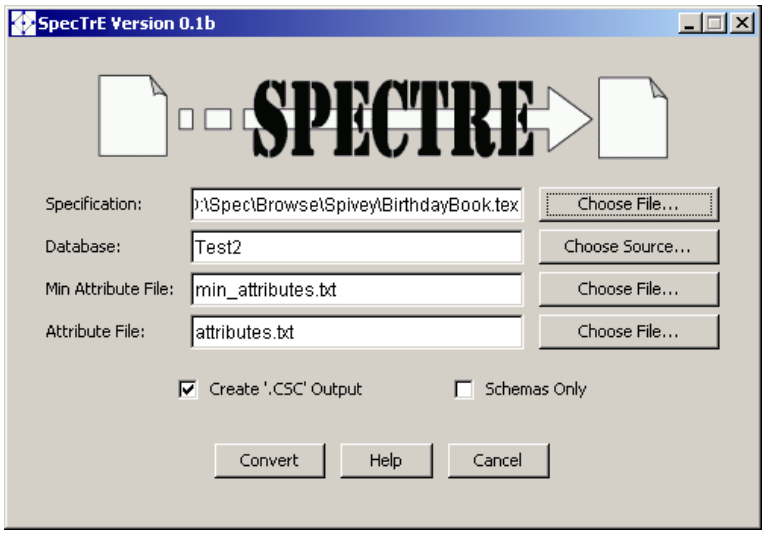


Figure 5.36: Screenshot of the SpecTrE GUI.

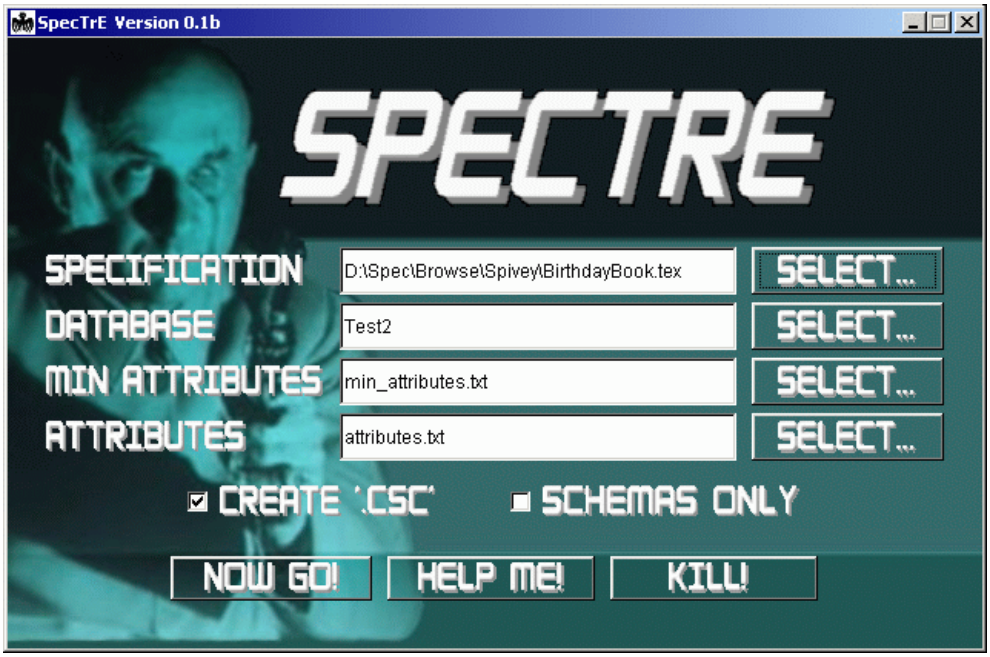


Figure 5.37: Screenshot of the SpecTrE interface in “Blofeld” mode.

facilitate the parsing of different mark-ups. For example, the new ZML fully-annotated syntax or CADiℤ style XML could be supported.

The "min attribute" file contains a list of attributes that must be included in the context even if they are not found in the specification. In some cases it can be useful to see which attributes are not used in any schemas by specifically including them in the context.

A "create '.CSC' " checkbox at the lower left of the GUI indicates that in addition to populating the database SpecTrE should also export a Conceptual Schema file. The CSC2CSX conversion tool was originally used to convert the data into the XML format used by ToscanaJ, however, both the Elba and Siena editors now support direct CSC import.

The "schemas only" checkbox is used to create contexts where both the object and attribute sets only contain schema names. If this option is selected then all schema names are included during parsing. Normally only those schemas involved in schema calculi operations are included in the context. These contexts can be used to generate line diagrams describing the relationships between schemas like those appearing in Figures 4.6 and 4.5.

5.4 Conclusion

This chapter has described the implementation of the prototype SpecTrE tool that embodies the ideas described in Chapter 4. The chapter opened with a discussion of Z representation issues and the ZML format was introduced. Section 5.2 then provided an overview of a number of FCA tools including ToscanaJ before Section 5.3 described the modular implementation of SpecTrE using ToscanaJ and ZML in conjunction with two custom tools: `tex2zml` and `spec2db`.

ZML was chosen as the Z representation format for SpecTrE because it supports sharing on the Web, automatic rendering in XSLT-enabled browsers, automatic hyperlinking and schema expansion. ToscanaJ was used to visualise the line diagrams because as an open source project the source code is readily available and it could be modified as required. Ultimately, however, the extensible view interface facilitated the web-browser integration without the need to modify or recompile any code. Furthermore, ToscanaJ supports conceptual scaling, nested-line diagrams, zooming, database connectivity and it is also

platform independent.

Knight, DeJong, Gibble and Nakano [119] proposed a framework to evaluate the specification of a control system for a nuclear research reactor using three different formal methods including Z. With regards to the usability of formal methods they concluded that:

> The ability to locate relevant information is a vital part of the utility of a specification. The ability to search, for example with regular expressions is valuable, but not sufficient.

SpecTrE provides this utility via conceptual scaling, zooming, and the ability to compose multiple scales into nested line diagrams. Knight et al. go on to state that:

> The formal method should also provide structuring mechanisms to aid in navigation since the specification document is likely to be large. In a natural language document, the table of contents and index assist in the location of information; many tools allow them to be generated automatically from the text. Another useful capability seen in text editing is the use of hypertext links to a related section or glossary entry. Formal methods must address the usability of the resulting specification documents.

This chapter has described the implementation of a tool that can generate and implement alternative document structuring and search mechanisms that can be generated automatically from the specification. While it may be possible to view and read specification properties directly from line diagrams without reference to the specification document itself, SpecTrE also represents a powerful navigation tool that addresses the usability of specification documents described above. Although part of this functionality comes through the utility of ZML, the line diagrams embody different views or different document structuring mechanisms while still retaining direct access to schemas via hyperlinks.

The next chapter concludes the thesis and presents some possible directions for future work.

Chapter 6

Conclusion

This final chapter concludes the thesis. Section 6.1 summarises the development of the thesis and the contributions made in each of the preceding chapters. Section 6.2 then introduces some related work to visualise software structure that has many parallels with the approach presented here. Finally, future directions and extensions to this work are presented in Section 6.3.

6.1 Thesis Summary

Chapter 1 of the thesis introduced the motivation for the work described here, revealed the overall structure of the thesis, and then presented the necessary background for both FCA and the Z notation. The motivation for this work is threefold. First, the majority of FCA applications in software engineering have focussed on late-phase software maintenance and re-engineering tasks. In contrast the focus of this thesis is the application of FCA to a number of early-phase activities within the software engineering life-cycle. While formal methods can be applied to all phases of the software engineering life-cycle [142, 143] the process of formal specification fits within the design phase.

The second motivation for this work relates to formal specification and in particular to existing attempts to increase the usability of Z by incorporating alternate graphical representations, most notably UML. As an alternative to this approach, the thesis described the visualisation and navigation of Z specifications via line diagrams representing Formal

Concept Lattices.

Within the formal methods community, tool support is seen as another path to increase the usability and thereby the adoption of formal methods like Z. The continued call for formal methods tool support represents the third motivation for this work. In response to this call the thesis described the implementation of a prototype tool developed by the author for visualising and navigating Z specifications based on FCA.

In support of the claim that the majority of FCA applications in software engineering have focussed on late-phase and software maintenance tasks, Chapter 2 presented the results of a comprehensive literature survey. The survey included a number of different views over the academic literature reporting the application of FCA in software engineering. These views categorised the survey papers according to: target language; application size; ISO12207 categorisation; author collaboration; and perceived impact via citation closure. The survey found that the majority of papers report applications to software maintenance and re-engineering tasks and that there was little work in early-phase software engineering design using FCA.

While many of the views were specific to this literature a number of generic views were presented as the basis for an FCA-based methodology for literature reviews in general. The major contributions of Chapter 2 are: the first broad survey of the FCA in software engineering literature; and a generic, FCA-based methodology for literature surveys.

In keeping with the first motivation outlined above, Chapters 3 and 4 then described early-phase software engineering applications of FCA. Chapter 3 presented a case study in the requirements engineering space comparing two class hierarchies that model aspects of a mass-transit railway system. The first hierarchy was produced for an existing Object-Z specification of the system while the second was derived using FCA. An informal description of the railway system was treated as a set of use-cases and the approach outline by Düwel was then used to identify class candidates.

While the resulting FCA structure was essentially the same as the existing hierarchy, the differences highlighted artefacts that had been introduced into the original structure during formal specification. The approach represents an informal form of object exploration and

serves to demonstrates the value of FCA as both a discussion promotion and question-answering tool. The formal application of object exploration represents an obvious extension to this work that is discussed further in Section 6.3.3.

Chapter 4 presented an application within the design phase of the software engineering life-cycle that also addresses the second motivation for this work: using alternate graphical representations to increase the usability of Z. The chapter briefly discussed existing work to increase the usability of Z by incorporating graphical notations, principally UML. Context creation and specification parsing issues were then discussed and the abstractions afforded by FCA were introduced. Conceptual scaling, nested line diagrams, zooming, animation and folding were all illustrated using the *BirthdayBook* specification as an example.

The major contribution of Chapter 4 is an FCA-based, alternative visual representation for visualising specification properties. The representation exploited a number of abstractions and the approach is amenable to partial automation with tool support.

The third motivation for this work is the continued call for tool support from the formal methods community. In response to this call, Chapter 5 described the implementation of a prototype tool for visualising and navigating Z specifications. The tool embodies the ideas introduced in Chapter 4 and exploits a number of existing technologies.

The chapter discussed a number of approaches to representing Z and also provided an overview of tool support for FCA. The implementation of a tool based on ZML and ToscanaJ was then described. SpecTrE — a graphical user interface front end for the specification transformation, parsing and context creation processes — was also introduced and a number of specification browser implementation issues were discussed. The major contribution of Chapter 5 is a prototype implementation of a platform independent, FCA-based tool for visualising and navigating Z specifications.

The next section of this chapter discusses a related approach to visualise software structure using FCA that parallels much of the work described in the thesis.

6.2 Related Work

The **C**onceptual **A**nalysis of **S**oftware **S**tructure (CASS) tool described by Cole and Tilley [40] is an FCA-based tool for analysing the structure of Java classes. While it is not directly related to formal specification the approach parallels much of the context creation and visualisation work described in Chapter 4, the tool implementation in Chapter 5, and some of the class-hierarchy work from Chapter 3.

Rather than only using Java source code as the input for analysis, CASS takes Java class files and considers the software as an algebraic structure. This allows a user to abstract over the syntax of the programming language and directly explore properties of interest within the software structure.

The process of software design and implementation often contains many arbitrary decisions — from the choice of method or variable names through to the structure of a class hierarchy. Within so called "agile methods" (and e**X**treme **P**rogramming (XP) in particular) regular refactoring activities are undertaken to revise the software structure [72, 12]. The CASS tool seeks to support these kinds of development processes by providing insight into the structure of the current design. Ideally, CASS would ultimately become a plug-in for use as an analysis tool within an Integrated Development Environment (IDE) like IBM's Eclipse [60] for Java. An overview of the CASS architecture is shown in Figure 6.1.

Source code analysers and profilers are used to extract information about the software which is stored and queried as a series of *information graphs*. These information graphs consist of triples of the form *(subject, predicate, object),* for example:

```
"java.util.List" is-a interface,
"java.util.List.isEmpty()" in "java.util.List".
"java.util.List.isEmpty()" is-a method.
```

These three triples assert that the class *java.util.List.isEmpty()* is a method within the interface *java.util.List.* A rule based system is then used to extend the Knowledge Base (KB) with new relationships and artefacts. For example, rules describing transitivity can be applied so that if a method *A* is in a class *B,* and class *B* is in package *C,* then the method *A* is also in the package *C*.

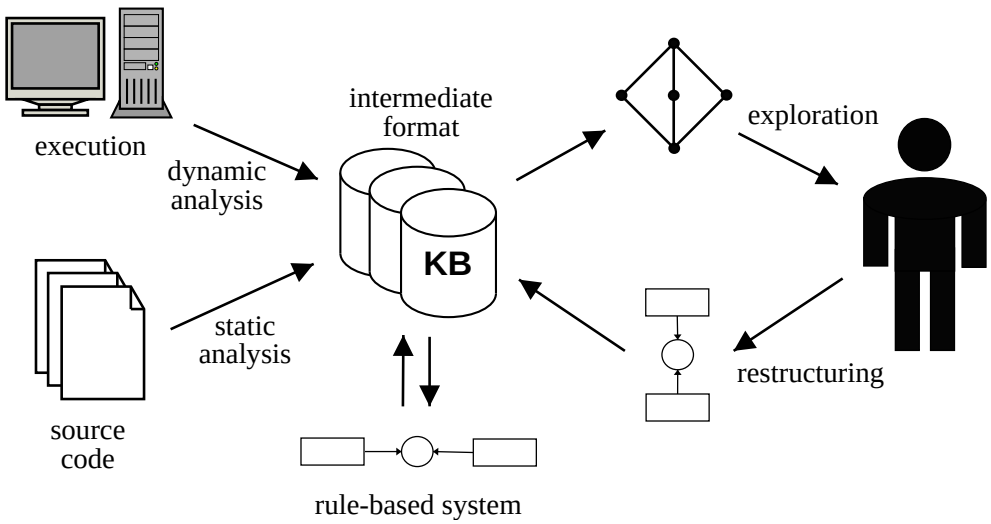


Figure 6.1: Architecture of the CASS tool.

Information graphs can also be used to define aspects of the software to be explored. These graphs can be used to query the knowledge base and generate result sets which are then visualised using concept lattices which are rendered by ToscanaJ. These hypotheses or questions may then be investigated either by generating new lattices, perhaps displaying new aspects of interest within the software structure, or by navigating back to the source artefacts within the software or its documentation.

Since each concept lattice is generated from a query graph, a natural refinement ordering allows general views to be elaborated and made more specific. Thus, the user is able to progress from a general view to a more specific view, or vice versa. In addition, two or more aspects of the software structure can be combined and visualised using nested line diagrams.

The use of information graphs for storage, knowledge base enrichment, and querying makes CASS a very flexible analysis tool. While CASS deals with software rather than formal specifications, there are many obvious parallels with the work described in the thesis. The parallels range from the overall aim to facilitate the exploration of the structure of the software through to the ultimate use of ToscanaJ for viewing the resulting line diagrams. The use of a CASS-like architecture for visualising Z specifications could create

a more flexible version of the SpecTrE tool and this idea is developed further in the next section.

6.3 Future Work

The CASS methodology and implementation architecture described above represents a possible extension to the specification navigation and visualisation work described in the thesis. However, there are also a number of other possible directions and extensions for this work.

6.3.1 Conceptual Analysis of Specification Structure

To extract triples from Java class files CASS currently uses IBM's CFParse class file analyser library [4]. The adaptation of CASS to create a tool for visualising Z specifications would require an equivalent mechanism to extract triples from Z specifications.

If we assume ZML input then it may be possible to exploit some of the work coming out of the CZT initiative [135] or build a simple triple generator based on a custom ZML or generic XML parser. The existing transformation approach described in Section 5.3.1 could still be used as required to transform source specifications in Oz style LaTeX mark-up into ZML.

In terms of the existing tool architecture presented in Chapter 5 a triple extractor and knowledge base would replace the existing parsing, context and database creation processes. ToscanaJ is already used to visualise the lattice diagrams produced by CASS and the existing browser integration technique could be adapted provided the anchor URLs are also stored in the knowledge base.

Given that Java is an object-oriented language then this approach should also facilitate the visualisation of Object-Z. Information graphs could be used to exploit richer relationships within the specifications and may also facilitate some level of schema expansion within the knowledge base.

6.3.2 Usability Testing

While Chapter 4 discussed how FCA could be used to visualise Z specifications and Chapter 5 described a tool implementing these ideas no claims were made about the usability of the tool. To make such claims for the SpecTrE tool described here would require usability testing and a comparison with existing tools such as the ORA Z Browser described in Section 5.1.1 and CADiℤ.

One approach to usability testing is to introduce the tool and evaluate its performance as part of an introductory Z or software engineering course at university. This is the approach taken by Richards et al. who conducted a survey using 201 second-year Analysis and Design students to evaluate their vocabulary guidelines and the line diagrams produced by their RECOCASE tool [158, 159, 157, 156]. Finney [68] and Finney, Fenton and Fedorec [69] also describe Z comprehensibility studies conducted with both undergraduate and postgraduate students while Mikušiak, Vojtek, Hasaralejko and Hanzelová used postgraduates and staff to evaluate their Z Browser [137].

University students are an obvious source of test subjects for academics, however, the results obtained do not necessarily reflect the experiences of real world users. Evaluations of this type may continue to perpetuate the view of Z tools as either tools for academics or research prototypes that fall short of the robust, industrial strength tools that the formal methods community require [33, 209].

Independent of any usability testing, the approach described here should be applied to other well known specifications such as the "library problem" [228, 232]. This would facilitate comparison with other approaches such as the UML-based visualisation work of Kim and Carrington [117] which uses this example.

6.3.3 Extending the Use-case Approach

In Section 3.5 *object exploration* was presented as a formal mechanism for both enriching the context based upon implications and as a way of defining an end point for the iterative process. Chapter 3, however, did not formally apply object exploration to the mass-transit example and it was left as an *ad hoc* process relying on the insight and intuition of the

designers to decide when to stop iterating. It would be interesting to formally apply object exploration to this example and contrast the resulting structure against both the original Object-Z hierarchy and the informal FCA-hierarchy presented in the chapter. The scalability issues should not present a significant problem for an example of this size and the ConImp, ConExp, or IMPEX tools could be used to support the process.

The possible automation of the initial noun extraction from the use-cases was also briefly mentioned in Section 3.3. Again, it would be an interesting exercise to compare both the Object-Z and FCA-hierarchies as presented in the chapter against a hierarchy where the initial noun extraction was performed automatically using either a controlled vocabulary or a suitable ontology of terms.

6.3.4 A Return to the Lattice of Specifications

The final extension proposed here seeks to exploit the semantics of Z rather than just the syntax. This could be achieved by representing some of the richer relationships contained in Z specifications as formal contexts. In what could be seen as a return to the earliest work of Mili et al. [138] introduced in Chapter 1 the idea of specification refinement is used to illustrate three possible approaches.

Specification Refinement

The Z notation can be used to provide specifications at different levels of abstraction. For example, an initial high-level specification may simply be concerned with inputs and outputs. This can later be refined to include error checking as seen in the development of the *BirthdayBook* specification with the introduction of the "robust" schemas. Using direct refinement a sequence of specifications can move from an initial abstract representation to a concrete one that can then be implemented. Each refinement includes more details and an overview of the technique based on Spivey's *BirthdayBook* is presented here [184].

The *BirthdayBook*1 schema shown below represents a refinement of the *BirthdayBook* schema with a concrete state space. The state space is modelled using two arrays: *names* which is used for storing the names, and *dates* which is used for storing the birthdates. The arrays are modelled by functions which map from the set of positive integers $\mathbb{N}_1$ to the

NAME and DATE data-types. In addition a variable called *hwm* is introduced to represent the "high water mark" — an index representing how much of the arrays are in use:

*BirthdayBook*1

$$names : \mathbb{N}_1 \rightarrow NAME$$
$$dates : \mathbb{N}_1 \rightarrow DATE$$
$$hwm : \mathbb{N}$$

$$\forall i, j : 1 .. hwm \bullet i \neq j \Rightarrow names(i) \neq names(j)$$

The predicate part of *BirthdayBook*1 simply checks that there are no repeated names contained in the *names* array. The relationship between the abstract state space defined by *BirthdayBook* and the new concrete state space in *BirthdayBook*1 can now be described:

Abs

BirthdayBook
*BirthdayBook*1

$$known = \{i : 1 .. hwm \bullet names(i)\}$$
$$\forall i : 1 .. hwm \bullet birthday(names(i)) = dates(i)$$

Having defined a concrete state space and the relationship between the abstract and concrete state spaces in the schema *Abs*, the original operations can now be refined. For example, the *AddBirthday*1 schema represents an initial array-based implementation of the *AddBirthday* schema:

*AddBirthday*1

Δ*BirthdayBook*1
$name? : NAME$
$date? : DATE$

$$\forall i : 1 .. hwm \bullet name? \neq names(i)$$
$$hwm' = hwm + 1$$
$$names' = names \oplus \{hwm' \rightarrow name?\}$$
$$dates' = dates \oplus \{hwm' \rightarrow date?\}$$

Given that this refinement now only contains notation with direct counterparts in a

programming language an actual implementation of the *AddBirthday* operation could be provided. While this example demonstrates only a single level of refinement there could be a number of successive refinements with corresponding proofs to show that each of the refinement steps are correct.

The existing visualisations described in Section 4.3 could provide significant insight into the structure of this refined version of the Birthday Book specification. For example, a line diagram representing schema composition would reveal two parallel structures with similar schema names based on the inclusion of either the *BirthdayBook* or *BirthdayBook*1 state space schemas. The effect would be similar to the structure revealed in Figure 4.6 where the use of the *Success* schema by the "robust" operations can clearly be seen. The application of scales containing either the state space schemas or input/output names would also reveal the parallel operation structures in the refined specification.

Provided naming conventions are used consistently within a specification, the patterns within schema names could also be used to provide different views over the structure of Z specifications. Example patterns from the *BirthddayBook* specification could include the sets of names {*RAddBirthday*, *RFindBirthday*, *RRemind*}, {*AddBirthday*, *RAddBirthday*, *AddBirthday*1} and {*AddBirthday*1, *FindBirthday*1, *Remind*1}. In the CASS-based specification tool described in Section 6.3.1 these patterns could be specified as a series of information graphs.

Two alternative approaches that could be used to represent relationships like schema refinement within specifications are *multicontexts* and *power context families*.

Multicontexts

The use of multicontexts [226] as a mechanism for representing richer relationship structures in specifications is another possible direction for this work. A formal multicontext consists of a number of sets and a number of binary relations that are represented as a network of formal contexts. With respect to the visualisation of Z specifications, a multicontext could be used to represent refinement where different contexts correspond to the specification at different levels of refinement. This approach could potentially form the basis for a tool that can not only provide abstractions but also

	uses	Ξ schema	Δ schema	via calculus
(InitBirthdayBook, BirthdayBook)	×			
(AddBirthday, BirthdayBook)	×		×	
(FindBirthday, BirthdayBook)	×	×		
(Remind, BirthdayBook)	×	×		
(AlreadyKnown, BirthdayBook)	×	×		
(NotKnown, BirthdayBook)	×	×		
(RAddBirthday, BirthdayBook)	×		×	
(RAddBirthday, AddBirthday)	×			×
(RAddBirthday, Success)	×			×
(RAddBirthday, AlreadyKnown)	×			×
(RFindBirthday, BirthdayBook)	×	×		
(RFindBirthday, FindBirthday)	×			×
(RFindBirthday, Success)	×			×
(RFindBirthday, NotKnown)	×			×
(RRemind, BirthdayBook)	×	×		
(RRemind, Remind)	×			×
(RRemind, Success)	×			×

Table 6.1: A formal context representing schema composition in $\mathbb{K}_2$. This context provides an alternate representation of the context in Table 4.6 using binary relationships between schemas.

the ability to relate views at different levels of abstraction [80].

Power Context Families

Power Context Families can also be used to represent the relationships between objects in formal contexts [46, 90, 89]. Formally, a power context family is a sequence $\vec{\mathbb{K}} := (\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2, ...)$ of formal contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ with $G_k \subseteq (G_0)^k$ for $k = 1, 2, ...$ The formal concepts of $\mathbb{K}_k$ with $k = 1, 2, ...$ are called *relation concepts* because they represent the k-ary relations on the object set G_0 by their extents. For example, while $\mathbb{K}_0$ represents the objects themselves, $\mathbb{K}_1$ represents unary relations between objects, $\mathbb{K}_2$ binary relationships, and so on.

Typically, only $\mathbb{K}_0$ and $\mathbb{K}_2$ are used and an example using $\mathbb{K}_2$ to represent binary relationships between schemas in the *BirthdayBook* specification is presented in Table 6.1. This context presents an alternate representation of the information in Table 4.6 that makes the composition type explicit.

The same approach could also be used to represent refinement between pairs of specifications and the refinements described at the start of this section are shown in

	refined in
(BirthdayBook, BirthdayBook1)	×
(AddBirthday, AddBirthday1)	×
(FindBirthday, FindBirthday1)	×
(Remind, Remind1)	×

Table 6.2: A formal context representing schema refinement in $\mathbb{K}_2$.

Table 6.2. Power context families represent another mechanism that could be further explored to exploit the relationships inherent in Z specifications.

Appendix A

BirthdayBook Specification

This appendix presents each of the schema and type declarations from Spivey's *BirthdayBook* specification [184] in its rendered form along with the corresponding Oz style LaTeX [118] and ZML mark-ups. The ZML shown here is consistent with the original version reported by Sun et al. [195]. A more recent version of ZML is discussed in Section 5.1.3 of Chapter 5.

PostScript/PDF:

$$[NAME, DATE]$$

Oz style LaTeX:

```
\begin{zed}
    [NAME, DATE]
\end{zed}
```

ZML:

```
<tydef align="left">
    [<name>DATE</name>,<name>NAME</name>]
</tydef>
```

PostScript/PDF:

$$REPORT ::= ok \mid already\_known \mid not\_known$$

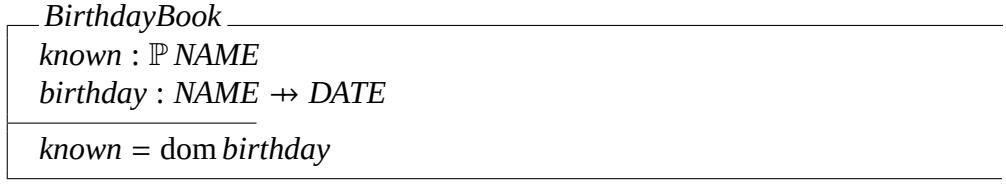
Oz style LaTeX:

```
\begin{zed}
    REPORT \ddef ok \bbar already\_known \bbar not\_known
\end{zed}
```

ZML:

```
<tydef align="left">
    <name>REPORT</name> &defs; ok &bbar; already_known &bbar;
    not_known
</tydef>
```

PostScript/PDF:



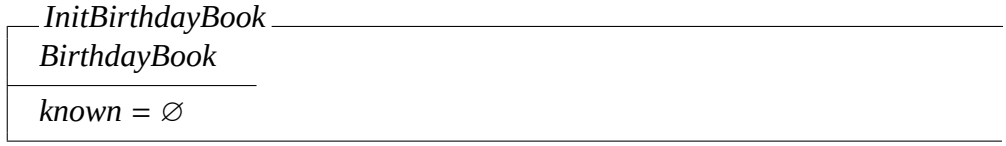
Oz style LaTeX:

```
\begin{schema}{BirthdayBook}
    known: \power NAME \\
    birthday: NAME \pfun DATE
\ST
    known = \dom birthday
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>BirthdayBook</name>
    <decl>
        <name>known</name>
        <dtype>
            &pset; <type>NAME</type>
        </dtype>
    </decl>
    <decl>
        <name>birthday</name>
        <dtype>
            <type>NAME</type> &pfun; <type>DATE</type>
        </dtype>
    </decl>
    <st/>
    <predicate>known = &dom; birthday</predicate>
</schemadef>
```

PostScript/PDF:



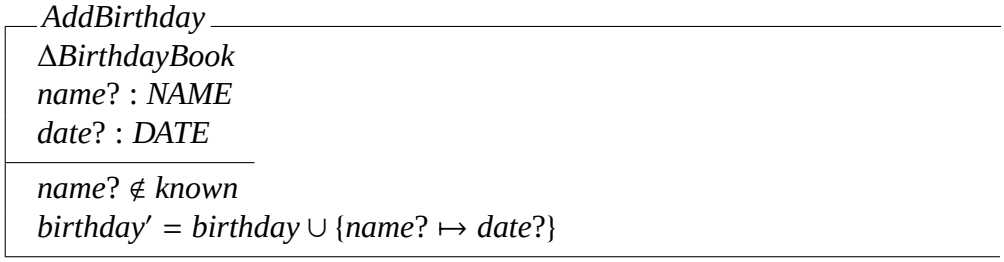
Oz style LaTeX:

```
\begin{schema}{InitBirthdayBook}
    BirthdayBook \\
\ST
    known = \emptyset
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>InitBirthdayBook</name>
    <incl>
        <type>BirthdayBook</type>
    </incl>BirthdayBook
    <st/>
    <predicate>known = &emptyset;</predicate>
</schemadef>
```

PostScript/PDF:



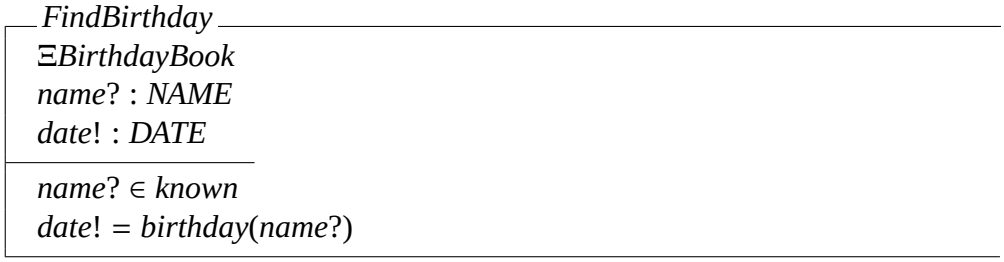
Oz style LaTeX:

```
\begin{schema}{AddBirthday}
    \Delta BirthdayBook \\
    name? : NAME \\
    date? : DATE
\ST
    name? \nem known \\
    birthday' = birthday \union \{name? \map date?\}
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>AddBirthday</name>
    <del>
        <type>BirthdayBook</type>
    </del>
    <decl>
        <name>name?</name>
        <dtype>
            <type>NAME</type>
        </dtype>
    </decl>
    <decl>
        <name>date?</name>
        <dtype>
            <type>DATE</type>
        </dtype>
    </decl>
    <st/>
    <predicate>name? &nem; known</predicate>
    <predicate>birthday' = birthday &uni; {name? &map; date?}</predicate>
</schemadef>
```

PostScript/PDF:



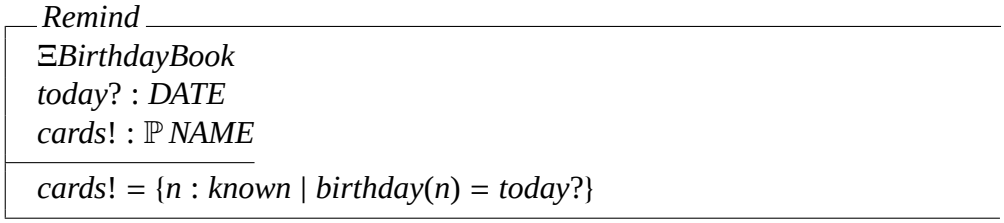
Oz style LaTeX:

```
\begin{schema}{FindBirthday}
    \Xi BirthdayBook \\
    name? : NAME \\
    date! : DATE
\ST
    name? \mem known \\
    date! = birthday(name?)
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>FindBirthday</name>
    <xi>
        <type>BirthdayBook</type>
    </xi>
    <decl>
        <name>name?</name>
        <dtype>
            <type>NAME</type>
        </dtype>
    </decl>
    <decl>
        <name>date!</name>
        <dtype>
            <type>DATE</type>
        </dtype>
    </decl>
    <st/>
    <predicate>name? &mem; known</predicate>
    <predicate>date! = birthday(name?)</predicate>
</schemadef>
```

PostScript/PDF:



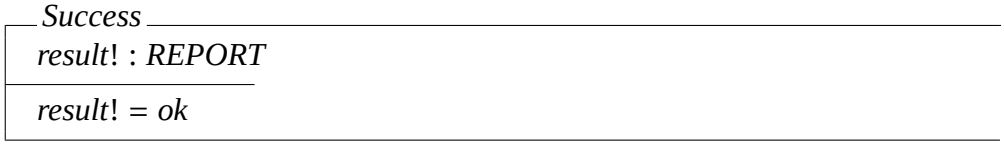
Oz style LaTeX:

```
\begin{schema}{Remind}
    \Xi BirthdayBook \\
    today? : DATE \\
    cards! : \power NAME
\ST
    cards! = \{ n : known \cbar birthday(n) = today? \}
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>Remind</name>
    <xi>
        <type>BirthdayBook</type>
    </xi>
    <decl>
        <name>today?</name>
        <dtype>
            <type>DATE</type>
        </dtype>
    </decl>
    <decl>
        <name>cards!</name>
        <dtype>
            &pset; <type>NAME</type>
        </dtype>
    </decl>
    <st/>
    <predicate>cards! = {n : known &bbar; birthday(n) = today?}</predicate>
</schemadef>
```

PostScript/PDF:



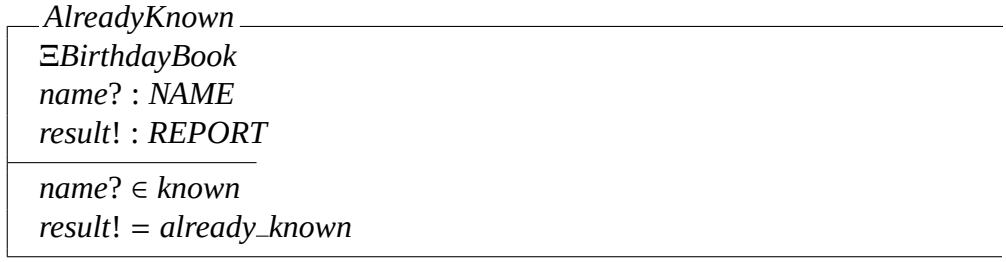
Oz style LaTeX:

```
\begin{schema}{Success}
    result! : REPORT
\ST
    result! = ok
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>Success</name>
    <decl>
        <name>result!</name>
        <dtype>
            <type>REPORT</type>
        </dtype>
    </decl>
    <st/>
    <predicate>result! = ok<predicate>
</schemadef>
```

PostScript/PDF:



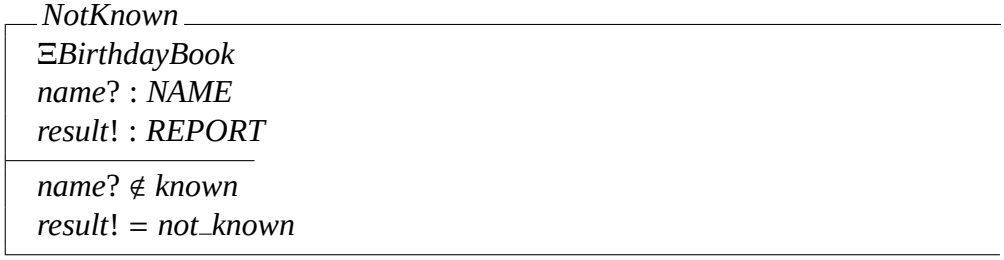
Oz style LaTeX:

```
\begin{schema}{AlreadyKnown}
    \Xi BirthdayBook \\
    name? : NAME \\
    result! : REPORT
\ST
    name? \mem known \\
    result! = already\_known
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>AlreadyKnown</name>
    <xi>
        <type>BirthdayBook</type>
    </xi>
    <decl>
        <name>name?</name>
        <dtype>
            <type>NAME</type>
        </dtype>
    </decl>
    <decl>
        <name>result!</name>
        <dtype>
            <type>REPORT</type>
        </dtype>
    </decl>
<st/>
    <predicate>name? &mem; known</predicate>
    <predicate>result! = already_known</predicate>
</schemadef>
```

PostScript/PDF:



Oz style LaTeX:

```
\begin{schema}{NotKnown}
    \Xi BirthdayBook \\
    name? : NAME \\
    result! : REPORT
\ST
    name? \nem known \\
    result! = not\_known
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>NotKnown</name>
    <xi>
        <type>BirthdayBook</type>
    </xi>
    <decl>
        <name>name?</name>
        <dtype>
            <type>NAME</type>
        </dtype>
    </decl>
    <decl>
        <name>result!</name>
        <dtype>
            <type>REPORT</type>
        </dtype>
    </decl>
    <st/>
    <predicate>name? &nem; known</predicate>
    <predicate>result! = not_known</predicate>
</schemadef>
```

PostScript/PDF:

$$RAddBirthday \mathrel{\widehat=} (AddBirthday \land Success) \lor AlreadyKnown$$

Oz style LaTeX:

```
\begin{zed}
    RAddBirthday \sdef (AddBirthday \land Success)
    \lor AlreadyKnown
\end{zed}
```

ZML:

```
<schemadef layout="calc" align="left">
    <name>RAddBirthday</name>
    <predcalc op="or">
        <predcalc op="and">
            <type>AddBirthday</type>
            <type>Success</type>
        </predcalc>
        <type>AlreadyKnown</type>
    </predcalc>&lor; AlreadyKnown
</schemadef>
```

PostScript/PDF:

$$RFindBirthday \mathrel{\widehat{=}} (FindBirthday \land Success) \lor NotKnown$$

Oz style LaTeX:

```
\begin{zed}
    RFindBirthday \sdef (FindBirthday \land Success) \lor NotKnown
\end{zed}
```

ZML:

```
<schemadef layout="calc" align="left">
    <name>RFindBirthday</name>
    <predcalc op="or">
        <predcalc op="and">
            <type>FindBirthday</type>
            <type>Success</type>
        </predcalc>
        <type>NotKnown</type>
    </predcalc>
</schemadef>
```

PostScript/PDF:

$$RRemind \mathrel{\widehat=} Remind \land Success$$

Oz style LaTeX:

```
\begin{zed}
    RRemind \sdef Remind \land Success
\end{zed}
```

ZML:

```
<schemadef layout="calc" align="left">
    <name>RRemind</name>
    <predcalc op="and">
        <type>Remind</type>
        <type>Success</type>
    </predcalc>
</schemadef>
```

LaTeX and ZML documents require opening and closing mark-up which are included here for completeness.

PostScript/PDF:

Not applicable.

Oz style LaTeX:

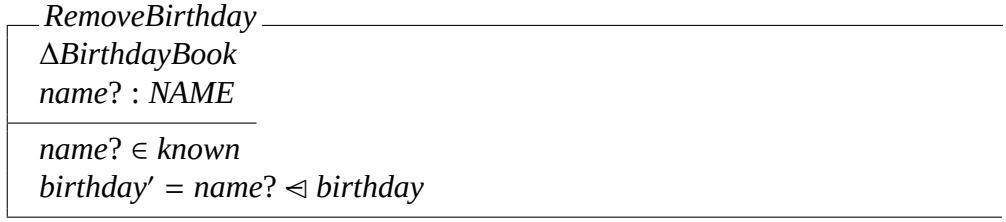
```
\documentclass[a4paper]{oz2e}
\begin{document}

    ...

\end{document}
```

ZML:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
    href="http://nt-appn.comp.nus.edu.sg/fm/zml/objectzed.xsl"?>
<!DOCTYPE unicode SYSTEM
    "http://nt-appn.comp.nus.edu.sg/fm/zml/unicode.dtd">

<objectZnotation xmlns="x-schema:objectZschema.xml"
    xmlns:HTML="http://www.w3.org/Profiles/XHTML-transitional">

    ...

</objectZnotation>
```

The *RemoveBirthday* schema shown here and the *ModifyBirthday* schema on the following page are extensions to the original BrithdayBook specification. They are discussed in Section 4.3.3 of the thesis.

PostScript/PDF:



Oz style LaTeX:

```
\begin{schema}{RemoveBirthday}
    \Delta BirthdayBook \\
    name? : NAME \\
\ST
    name? \mem known \\
    birthday' = {name?} \dsub birthday
\end{schema}
```

ZML:

```
<schemadef layout="simpl" align="left">
    <name>RemoveBirthday</name>
    <del>
        <type>BirthdayBook</type>
    </del>
    <decl>
        <name>name?</name>
        <dtype>
            <type>NAME</type>
        </dtype>
    </decl>
<st/>
    <predicate>name? &mem; known</predicate>
    <predicate>birthday' = {name?} &dsub; birthday</predicate>
</schemadef>
```

PostScript/PDF:

$$ModifyBirthday \mathrel{\widehat{=}} RemoveBirthday \fatsemi AddBirthday$$

Oz style LaTeX:

```
\begin{zed}
    ModifyBirthday \sdef RemoveBirthday \zcmp AddBirthday
\end{zed}
```

ZML:

```
<schemadef layout="calc" align="left">
    <name>ModifyBirthday</name>
    <predcalc op="com">
        <type>RemoveBirthday</type>
        <type>AddBirthday</type>
    </predcalc>
</schemadef>
```

Bibliography

[1] F. Achermann and O. Nierstrasz, “Moose: a language-independent reengineering environment,” May 2003. [Online]. Available: http://www.iam.unibe.ch/˜scg/Research/Moose/

[2] S. Agerholm and P. Larsen, “A lightweight approach to formal methods,” in *Applied Formal Methods — FM-Trends 98*, ser. LNAI 1641, D. Hutter, W. Stephan, P. Traverso, and M. Ullman, Eds. Berlin: Springer-Verlag, October 1998, pp. 168–183.

[3] A. Alencar and J. Goguen, “OOZE: An object-oriented Z environment,” in *European Conference on Object Oriented Programming*, ser. LNCS 512, P. America, Ed. New York: Springer-Verlag, 1991, pp. 180–199.

[4] “alphaWorks: CFParse,” alphaWorks, September 2000. [Online]. Available: http://www.alphaworks.ibm.com/tech/cfparse/

[5] G. Ammons, D. Mandelin, R. Bodik, and J. Larus, “Debugging temporal specifications with concept analysis,” in *Proceedings of the Conference on Programming Language Design and Implementation PLDI’03*. ACM, June 2003.

[6] U. Andelfinger, *Diskursive Anforderungsanalyse. Ein Beitrag zum Reduktionsproblem bei Systementwicklungen in der Informatik.* Frankfurt: Peter Lang, 1997.

[7] G. Antoniol, R. Fiutem, G. Lutteri, P. Tonella, and S. Zanfei, “Program understanding and maintenance with the CANTO environment,” in *Proceedings International Conference on Software Maintenance*, October 1997, pp. 72–81.

[8] G. Arévalo, “Understanding behavioral dependencies in class hierarchies using

concept analysis," in *Proceedings of LMO 2003 (Langages et Modéles á Object)*. Paris (France): Hermes, February 2003.

[9] G. Arévalo, S. Ducass, and O. Nierstrasz, "Understanding classes using x-ray views," in *MASPEGHI 2003, MAnaging SPEcialization/Generalization HIerarchies (MASPEGHI) Workshop at ASE 2003*, Montreal, Canada, 2003, preliminary version.

[10] "Graphviz," AT&T Labs-Research, October 2000. [Online]. Available: http://www.research.att.com/sw/tools/graphviz/

[11] T. Ball, "The concept of dynamic analysis," in *Proceedings of ACM SIGSOFT Symposium on the Foundations of Software Engineering*, September 1999, pp. 216–234.

[12] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

[13] P. Becker, "Multi-dimensional representations of conceptual hierarchies," in *Conceptual Structures—Extracting and Representing Semantics, Contributions to ICCS 2001*, 2001, pp. 145–158.

[14] P. Becker, "Tockit – a framework for conceptual knowledge processing," June 2003. [Online]. Available: http://tockit.sourceforge.net/

[15] P. Becker, "ToscanaJ: Welcome," February 2003. [Online]. Available: http://toscanaj.sourceforge.net/

[16] P. Becker and J. H. Correia, "The ToscanaJ suite for implementing conceptual information systems," in *Proceedings of the First International Conference on Formal Concept Analysis — ICFCA'03*, G. Stumme, Ed. Springer-Verlag, 2004, to appear.

[17] P. Becker, "Potential data formats for FCA," March 2004. [Online]. Available: http://kvo.itee.uq.edu.au/twiki/bin/view/Tockit/PotentialDataFormats

[18] W. Biggs, "package gnu.regexp — regular expressions for Java," June 2001, version 1.1.3. [Online]. Available: http://www.cacas.org/java/gnu/regexp/

[19] G. Birkhoff, *Lattice Theory*, 2nd ed. New York: American Mathematical Society, 1948.

[20] B. Boehm, “A spiral model of software development and enhancement,” in *Tutorial: Software Engineering Project Management*, R. Thayer, Ed. Washington: IEEE Computer Society, 1987, pp. 128–142.

[21] D. Bojic and D. Velasevic, “Reverse engineering of use case realizations in UML,” in *Symposium on Applied Computing — SAC2000*. ACM, 2000. [Online]. Available: http://www.acm.org/conferences/sac/sac00/Proceed/FinalPapers

[22] G. Booch, I. Jacobson, and J. Rumbaugh, *The unified modeling language user guide*, ser. Addison-Wesley object technology. Reading, Massachusetts: Addison-Wesley, 1999.

[23] E. Börger, “High level system design using abstract state machines,” in *Applied Formal Methods – FM–Trends 98*, ser. LNAI 1641, D. Hutter, W. Stephan, P. Traverso, and M. Ullman, Eds. Berlin: Springer-Verlag, October 1998, pp. 1–43.

[24] K. Böttger, R. Schwitter, D. Richards, O. Aguilera, and D. Mollá, “Reconciling use cases via controlled language and graphical models,” in *INAP’2001 — Proceedings of the 14th International Conference on Applications of Prolog*. Japan: University of Tokyo, October 2001, pp. 20–22.

[25] J. Bowen, “comp.specification.z frequently asked questions (FAQ),” May 2003. [Online]. Available: http://www.faqs.org/faqs/z-faq/

[26] J. Bowen, “The world wide web virtual library: The Z notation,” June 2003. [Online]. Available: http://www.zuser.org/z/

[27] J. Bowen and D. Chippington, “Z on the web using Java,” in *ZUM’98: The Z Formal Specification Notation, 11th International Conference of Z Users*, ser. LNCS 1493, J. Bowen, A. Fett, and M. Hinchey, Eds. Berlin: Springer-Verlag, 1998, pp. 66–80.

[28] J. Bowen and M. Hinchey, “Seven more myths of formal methods,” *IEEE Software*, vol. 12, no. 4, pp. 34–41, July 1995.

[29] J. Bowen and M. Hinchey, Eds., *Applications of Formal Methods*. London: Prentice Hall, 1996.

[30] P. Buch, “HPGL – Hewlett Packard Graphics Language,” February 2003. [Online].

Available: http://www.piclist.com/techref/language/hpgl.htm

[31] F. Buchli, "Detecting software patterns using formal concept analysis," Institut für Informatik und angewandte Mathematik, Universität Bern, Switzerland, Technical Report IAM-03-010, September 2003. [Online]. Available: http://www.buchli.org/frank/work/master/diplomaBuchliFrank.pdf

[32] P. Burmeister, "Formal concept analysis with ConImp: Introduction to the basic features," TU-Darmstadt, Darmstadt, Germany, Tech. Rep., 1996. [Online]. Available: http://www.mathematik.tu-darmstadt.de/~burmeister/

[33] R. Butler and C. Holloway, "Impediments to industrial use of formal methods," *IEEE Computer*, pp. 25–26, April 1996.

[34] G. Canfora, A. Cimitile, A. De Lucia, and G. Di Lucca, "A case study of applying an eclectic approach to identify objects in code," in *Workshop on Program Comprehension*. IEEE, 1999, pp. 136–143.

[35] C. Carpineto and G. Romano, "Order-theoretical ranking," *Journal of the American Society for Information Sciences (JASIS)*, vol. 7, no. 51, pp. 587–601, 2000.

[36] P. Ciancarini, C. Mascolo, and F. Vitali, "Visualizing Z notation in HTML documents," in *ZUM'98: The Z Formal Specification Notation, 11th International Conference of Z Users*, ser. LNCS 1493, J. Bowen, A. Fett, and M. Hinchey, Eds. Berlin: Springer-Verlag, 1998, pp. 81–95.

[37] E. Clarke and J. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, December 1996.

[38] R. Cole and P. Eklund, "Scalability in formal concept analysis," *Computational Intelligence,*, vol. 15, no. 1, pp. 11–27, 1999.

[39] R. Cole, P. Eklund, and G. Stumme, "CEM — a program for visualization and discovery in email," in *4th European conference on principles and practice of knowledge discovery in databases, PKDD 2000*, ser. LNAI 1910, D. Zighed, J. Komorowski, and J. Zytkow, Eds. Berlin: Springer-Verlag, September 2000, pp. 367–374.

[40] R. Cole and T. Tilley, "Conceptual analysis of software structure," in *Proceedings*

of Fifteenth International Conference on Software Engineering and Knowledge Engineering, SEKE'03. USA: Knowledge Systems Institute, June 2003, pp. 726–733.

[41] R. Cole, "The management and visualisation of document collections using formal concept analysis," Ph.D. dissertation, Griffith University, School of Information and Communication Technology, Parklands Drive, Southport QLD 4215, December 2000.

[42] R. Cole and P. Eklund, "Browsing semi-structured web texts using formal concept analysis," in *Proceedings 9th International Conference on Conceptual Structures*, ser. LNAI 2120. Berlin: Springer-Verlag, 2001, pp. 319–332.

[43] R. Corderoy, "troff.org – the text processor for typesetters," 2003. [Online]. Available: http://www.troff.org

[44] B. Davey and H. Priestly, *Introduction to Lattices and Order*, 2nd ed. Press Syndicate of the University of Cambridge, 2002.

[45] U. Dekel, "Applications of concept lattices to code inspection and review," in *The Israeli Workshop on Programming Languages and Development Environments*. Israel: IBM Haifa Research Lab, July 2002, ch. 6. [Online]. Available: http://www.haifa.il.ibm.com/info/ple/papers/inspec_summary.pdf

[46] H. Delugach, M. Keeler, D. Lukose, L. Searle, and J. Sowa, Eds., *Conceptual Graphs and Formal Concept Analysis*, ser. LNCS 1257. Berlin: Springer-Verlag, 1997.

[47] A. Diller, *Z: An Introduction to Formal Methods*, 2nd ed. Chichester: John Wiley and Sons, 1994.

[48] J. Dong, "Z family on the web with their UML photos," Nov 2001. [Online]. Available: http://nt-appn.comp.nus.edu.sg/fm/zml/

[49] J. Dong, "XML schema definition for the Z family formal specification languages (Z/Object-Z/TCOZ)," 2002. [Online]. Available: http://nt-appn.comp.nus.edu.sg/fm/zml/zml.xsd

[50] J. Dong, "XML schema definition of Z specification language," Aug 2002. [Online]. Available: http://nt-appn.comp.nus.edu.sg/fm/zml/z-stand/zml-xsd.htm

[51] J. Dong, Y. Li, J. Sun, J. Sun, and H. Wang, "XML-based static type checking and dynamic visualization for TCOZ," in *4th International Conference on Formal Engineering Methods*. Springer-Verlag, October 2002, pp. 311–322.

[52] R. Duke and G. Rose, *Formal Object-Oriented Specification Using Object-Z*. MacMillan Press, 2000.

[53] V. Duquenne, "Latticial structures in data analysis," *Theoretical Computer Science*, vol. 217, no. 2, pp. 407–436, 1999.

[54] V. Duquenne, "GLAD: A program for general lattice analysis and design," in *Proceedings of the First International Conference on Formal Concept Analysis — ICFCA'03*, G. Stumme, Ed. Springer-Verlag, 2004, to appear.

[55] V. Duquenne, C. Chabert, A. Cherfouh, J.-M. Delabar, A.-L. Doyen, and D. Pickering, "Structuration of phenotypes/genotypes through galois lattices and implications," in *CLKDD'01: Concept Lattices-based Theory, Methods and Tools for Knowledge Discovery in Databases*, E. Nguifo, M. Liquiére, and V. Duquenne, Eds., vol. 42. CEUR, 2001, pp. 21–32.

[56] S. Düwel, "Enhancing system analysis by means of formal concept analysis," in *Conference on Advanced Information Systems Engineering 6th Doctoral Consortium*, Heidelberg, Germany, June 1999.

[57] S. Düwel, "BASE — ein begriffsbasiertes analyseverfahren für die software-entwicklung," Ph.D. dissertation, Philipps-Universität, Marburg, 2000. [Online]. Available: http://www.ub.uni-marburg.de/digibib/ediss/welcome.html

[58] S. Düwel and W. Hesse, "Identifying candidate objects during system analysis," in *Proceedings of CAiSE'98/IFIP 8.1 Third International Workshop on Evaluation of Modelling Methods in System Analysis and Design (EMMSAD'98)*, Pisa, 1998.

[59] S. Düwel and W. Hesse, "Bridging the gap between use case analysis and class structure design by formal concept analysis," in *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik. Proceedings "Modellierung 2000"*, J. Ebert and U. Frank, Eds. Koblenz: Fölbach-Verlag, 2000, pp. 27–40.

[60] "Eclipse.org main page," Eclipse Consortium, June 2003. [Online]. Available:

http://www.eclipse.org

[61] J. Eijndhoven, "Graphplace," December 1995. [Online]. Available: ftp://ftp.dcs.warwick.ac.uk/people/Martyn.Amos/packages/graphplace/graphplace.tar.gz

[62] T. Eisenbarth, R. Koschke, and D. Simon, "Aiding program comprehension by static and dynamic feature analysis," in *Proceedings of ICSM2001 — The International Conference on Software Maintenance*. IEEE Computer Society Press, 2001, pp. 602–611.

[63] T. Eisenbarth, R. Koschke, and D. Simon, "Feature-driven program understanding using concept analysis of execution traces," in *9th International Workshop on Program Comprehension*. IEEE, 2001, pp. 300–309.

[64] T. Eisenbarth, R. Koschke, and D. Simon, "Locating features in source code," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 195–209, March 2003.

[65] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 3rd ed. Addison-Wesley, 2000.

[66] "Mail-Sleuth," Email Analysis Pty Ltd, June 2003. [Online]. Available: http://www.mail-sleuth.com

[67] A. Evans and A. Clark, "Foundations of the unified modeling language," in *BCS-FACS Northern Formal Methods Workshop*, ser. Electronic Workshops in Computing, D. Duke and A. Evans, Eds. Springer Verlag, 1998.

[68] K. Finney, "Mathematical notation in formal specification: Too difficult for the masses?" *IEEE Transactions on Software Engineering*, vol. 22, no. 2, pp. 158–159, February 1996.

[69] K. Finney, N. Fenton, and A. Fedorec, "Effects of structure on the comprehensibility of formal specifications," *IEE Proceedings — Software Engineering*, vol. 146, no. 4, pp. 193–202, August 1999.

[70] B. Fischer, "Specification-based browsing of software component libraries," in *Automated Software Engineering*, 1998, pp. 74–83. [Online]. Available: http://citeseer.nj.nec.com/fischer99specificationbased.html

[71] "FME: Tools database," Formal Methods Europe. [Online]. Available:

http://www.fmeurope.org/databases/tools.html

[72] M. Fowler, *Refactoring, Improving the Design of Existing Code*. Addison Wesley, 1999.

[73] R. Freese, "Lattice drawing," August 2003. [Online]. Available: http://www.math.hawaii.edu/~ralph/LatDraw/

[74] P. Funk, A. Lewien, and G. Snelting, "Algorithms for concept lattice decomposition and their applications," TU Braunschweig, Tech. Rep. 95-09, December 1995. [Online]. Available: http://citeseer.nj.nec.com/117356.html

[75] B. Ganter, "Attribute exploration with background knowledge," *Theoretical Computer Science*, vol. 217, no. 2, pp. 215–233, 1999. [Online]. Available: citeseer.nj.nec.com/ganter96attribute.html

[76] B. Ganter and R. Wille, "Conceptual scaling," in *Applications of combinatorics and graph theory to the biological and social sciences*, F. Roberts, Ed. New York: Springer-Verlag, 1989, pp. 139–167.

[77] B. Ganter and R. Wille, "Applied lattice theory: Formal concept analysis," 1997. [Online]. Available: http://citeseer.nj.nec.com/ganter97applied.html

[78] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*. Berlin: Springer-Verlag, 1999.

[79] D. Germán and D. Cowan, "Experiments with the Z interchange format and SGML," in *Proceedings of ZUM'95 — 9th International Conference of Z Users*, J. Bowen and M. Hinchey, Eds. Springer-Verlag, 1995, pp. 224–233.

[80] S. German, "Research goals for formal methods," *ACM Computing Surveys*, vol. 28, no. 4es, December 1996.

[81] C. L. Giles, K. Bollacker, and S. Lawrence, "CiteSeer: An automatic citation indexing system," in *Digital Libraries 98 — The Third ACM Conference on Digital Libraries*, I. Witten, R. Akscyn, and F. M. Shipman III, Eds. Pittsburgh, PA: ACM Press, June 23–26 1998, pp. 89–98. [Online]. Available: http://citeseer.nj.nec.com/giles98citeseer.html

[82] R. Godin, H. Mili, G. W. Mineau, R. Missaoui, A. Arfi, and T.-T. Chau, "Design

of class hierarchies based on concept (galois) lattices," *Theory and Application of Object Systems (TAPOS)*, vol. 4, no. 2, pp. 117–134, 1998.

[83] R. Godin, G. Mineau, and R. Missaoui, "Incremental structuring of knowledge bases," in *Proceedings of the International Knowledge Retrieval, Use, and Storage for Efficiency Symposium (KRUSE'95)*, ser. LNAI, vol. 9, no. 2. Springer-Verlag, 1995, pp. 179–198.

[84] R. Godin, G. Mineau, R. Missaoui, M. St-Germain, and N. Faraj, "Applying concept formation methods to software reuse," *International Journal of Knowledge Engineering and Software Engineering*, vol. 5, no. 1, pp. 119–142, 1995.

[85] R. Godin and R. Missaoui, "An incremental concept formation approach for learning from databases," *Theoretical Computer Science, Special Issue on Formal Methods in Databases and Software Engineering*, vol. 133, pp. 387–419, 1994.

[86] R. Godin, R. Missaoui, and A. April, "Experimental comparison of navigation in a galois lattice with conventional information retrieval methods," *International journal of Man-Machine Studies*, vol. 38, no. 5, pp. 747–767, May 1993.

[87] R. Godin and H. Mili, "Building and maintaining analysis-level class hierarchies using galois lattices," in *Proceedings of the OOPSLA'93 Conference on Object-oriented Programming Systems, Languages and Applications*, 1993, pp. 394–410. [Online]. Available: http://citeseer.nj.nec.com/godin93building.html

[88] W. Grieskamp, "The ZETA system," October 1998. [Online]. Available: http://uebb.cs.tu-berlin.de/zeta/

[89] B. Groh, "A contextual-logic framework based on relational power context families," Ph.D. dissertation, Griffith University, School of Information and Communication Technology, March 2002.

[90] B. Groh and P. Eklund, "Algorithms for creating relational power context families from conceptual graphs," in *Conceptual Structures: Standards and Practices, ICCS'99*, ser. LNAI 1640, W. Tepfenhart and W. Cyre, Eds. Berlin: Springer-Verlag, 1999, pp. 389–400.

[91] A. Hall, "Seven myths of formal methods," *IEEE Software*, pp. 11–19, September

1990.

[92] I. Hayes, Ed., *Specification Case Studies*. Prentice Hall, 1987.

[93] J. Hereth, "DOS programs of the Darmstadt research group on formal concept analysis," June 1999. [Online]. Available: http://www.mathematik.tu-darmstadt.de/ags/ag1/Software/DOS-Programme/Welcome_en.html

[94] W. Hesse and T. Tilley, *FCA — The state of the art*, ser. LNCS. Springer-Verlag, 2004, ch. Formal Concept Analysis used for software analysis and modelling, to appear.

[95] C. Holloway, "NASA LaRC fornal methods humor," May 2002. [Online]. Available: http://shemesh.larc.nasa.gov/fm/fm-humor.html

[96] M. Huchard and H. Leblanc, "From Java classes to Java interfaces through galois lattices," in *Actes de ORDAL'99: 3rd International Conference on Orders, Algorithms and Applications*, Montpellier, 1999, pp. 211–216.

[97] M. Huchard, C. Roume, and P. Valtchev, "When concepts point at other concepts: the case of UML diagram reconstruction," in *Advances in Formal Concept Analysis for Knowledge Discovery in Databases, FCAKDD 2002*, 2002, pp. 32–43.

[98] "Visual modeling with rational rose home," IBM, October 1999. [Online]. Available: http://www.rational.com/products/rose/

[99] IEEE, *IEEE Std 610.12-1990 — IEEE Standard Glossary of Software Engineering Terminology*. New York: IEEE, September 1990.

[100] IEEE, *IEEE/EIA 12207.0-1996 — Standard for Information Technology — Software life cycle processes*. New York: IEEE, March 1998.

[101] "Information processing — text and office systems — standard generalized markup language (sgml)," ISO 8879:1986, 1986.

[102] "Information technology — universal multiple-octet coded character set (UCS) — part 1: Architecture and basic multilingual plane," ISO/IEC 10646-1, 2000.

[103] "Information technology — universal multiple-octet coded character set (UCS) — part 2: Supplementary planes," ISO/IEC 10646-2, 2001.

[104] "Information technology — Z formal specification notation — syntax, type system

and semantics," ISO/IEC 13568:2002, January 2002.

[105] D. Jackson, "A comparison of object modelling notations: Alloy, UML and Z," August 1999, unpublished manuscript. [Online]. Available: http://sdg.lcs.mit.edu/~dnj/pubs/alloy-comparison.pdf

[106] D. Jackson, I. Schechter, and I. Shlyakhter, "Alcoa: the Alloy constraint analyzer," in *Proceedings of the International Conference on Software Engineering*, Limerick, Ireland, June 2000, pp. 730–733.

[107] D. Jackson and M. Vaziri, "Some shortcomings of OCL, the object constraint language of UML," December 1999, response to Object Management Group's Request for Information on UML 2.0. [Online]. Available: http://sdg.lcs.mit.edu/~dnj/publications.html

[108] J. Jacky, "Z2HTML translator," March 2001. [Online]. Available: http://staff.washington.edu/~jon/z/z2html/z2html.html

[109] M. Janssen, "Online Java lattice building application," August 2003. [Online]. Available: http://juffer.xs4all.nl/cgi-bin/jalaba/JaLaBA.pl

[110] X. Jia, "Z type checker," October 2002. [Online]. Available: http://venus.cs.depaul.edu/fm/ztc.htm

[111] X. Jia, "ZANS tool," October 2002. [Online]. Available: http://venus.cs.depaul.edu/fm/zans.htm

[112] W. Johnston, "A type checker for Object-Z," Software Verification Research Centre, School of Information Technology, The University of Queensland, Brisbane 4072. Australia, Technical report 96-24, September 1996. [Online]. Available: http://svrc.it.uq.edu.au/Bibliography/svrc-tr.html?96-24

[113] Y. Kalfoglou, "Applications for FCA in AKT," July 2003. [Online]. Available: http://www.aktors.org/technologies/fca/fca_techProfile.htm

[114] S.-K. Kim and D. Carrington, "Formalizing the UML class diagram using Object-Z," in *Proceedings of the Second IEEE conference on UML: UML'99*, ser. LNCS 1723. Springer-Verlag, 1999, pp. 83–98.

[115] S.-K. Kim and D. Carrington, "An integrated framework with UML and Object-Z

for developing a precise and understandable specification: The light control case study," in *Seventh Asia-Pacific Software Engineering Conference*. Los Alamitos, California: IEEE Computer Society, December 2000, pp. 240–248.

[116] S.-K. Kim and D. Carrington, "A formal metamodeling approach to a transformation between UML state machine and Object-Z," in *Conference on Formal Engineering Methods (ICFEM2002)*, ser. LNCS 2495, C. George and H. Miao, Eds. Berlin: Springer-Verlag, 2002, pp. 548–560.

[117] S.-K. Kim and D. Carrington, "Visualization of formal specifications," in *Sixth Asia-Pacific Software Engineering Conference*. Los Alamitos, California: IEEE Computer Society, December 1999, pp. 38–45.

[118] P. King, *Printing Z and Object-Z LaTeX documents*, Software Verification Research Centre, University of Queensland, Australia, May 1990.

[119] J. Knight, C. DeJong, M. Gibble, and L. Nakano, "Why are formal methods not used more widely?" in *Fourth NASA Formal Methods Workshop*, Hampton, VA, September 1997. [Online]. Available: http://www.cs.virginia.edu/~jck/publications/lfm.97.pdf

[120] D. Knuth, "Literate programming," *The Computer Journal*, vol. 27, pp. 97–111, May 1984.

[121] W. Kollewe, M. Skorsky, F. Vogt, and R. Wille, "TOSCANA — ein werkzeug zur begrifflichen analyse und erkundung von daten," in *Begriffliche Wissensverarbeitung — Grundfragen und Aufgaben*, R. Wille and M. Zickwolff, Eds. Mannheim-Leipzig-Wien-Zuerich: B.-I. Wissenschaftsverlag, 1994, pp. 267–288.

[122] M. Krone and G. Snelting, "On the inference of configuration structures from source code," in *Proceedings of the International Conference on Software Engineering (ICSE 1994)*, 1994, pp. 49–57.

[123] T. Kuipers and L. Moonen, "Types and concept analysis for legacy systems," Centrum voor Wiskunde en Informatica, Tech. Rep. SEN-R0017, July 2000.

[124] L. Lamport, *LaTeX: A Document Preparation System*, 2nd ed. Addison-Wesley, 1994.

[125] K. Lano, “Z++: an object-oriented extension to Z,” in *Z Users Workshop: Proceedings of the 4th Annual Z User Meeting*, ser. Workshops in Computing, J. Nicholls, Ed. Berlin: Springer-Verlag, 1991, pp. 151–172.

[126] H. Leblanc, C. Dony, M. Huchard, and T. Libourel, “An environment for building and maintaining class hierarchies,” in *ECOOP’99: Workshop “Object-Oriented Architectural Evolution”*, ser. LNCS 1743, A. Moreira and S. Demeyer, Eds. Springer-Verlag, 1999, pp. 77–78.

[127] “The ProofPower web pages,” Lemma 1 Ltd, May 2003. [Online]. Available: http://www.lemma-one.com/ProofPower/index/

[128] C. Lindig, “Concept-based component retrieval,” in *Working Notes of the IJCAI-95 Workshop: Formal Approaches to the Reuse of Plans, Proofs, and Programs*, J. Köhler, F., Giunchiglia, C. Green, and C. Walther, Eds., August 1995, pp. 21–25.

[129] C. Lindig, “TKConcept,” January 1996. [Online]. Available: http://sensei.ieec.uned.es/manuales/tkconcept/welcome.html

[130] C. Lindig, “A concept analysis framework,” 1998. [Online]. Available: http://www.st.cs.uni-sb.de/~lindig/papers/tkconcept/iccs.pdf

[131] C. Lindig, “Concepts,” September 2003. [Online]. Available: http://www.st.cs.uni-sb.de/~lindig/src/concepts.html

[132] C. Lindig and G. Snelting, “Assessing modular structure of legacy code based on mathematical concept analysis,” in *Proceedings of the International Conference on Software Engineering (ICSE 97)*, Boston, 1997, pp. 349–359.

[133] B. Mahony and J. Dong, “Timed communicating Object-Z,” *IEEE Transactions on Software Engineering*, vol. 26, no. 2, pp. 150–177, February 2000.

[134] B. Mahony and J. Dong, “Deep semantic links of TCSP and Object-Z: TCOZ approach,” *Formal Aspects of Computing*, vol. 13, pp. 142–160, 2002.

[135] A. Martin, “Community Z tools initiative,” July 2003. [Online]. Available: http://web.comlab.ox.ac.uk/oucl/work/andrew.martin/CZT/

[136] H. Miao, L. Liu, and L. Li, “Formalizing UML models with Object-Z,” in *Conference on Formal Engineering Methods (ICFEM2002)*, ser. LNCS 2495,

C. George and H. Miao, Eds. Berlin: Springer-Verlag, 2002, pp. 523–534.

[137] L. Mikušiak, V. Vojtek, J. Hasaralejko, and J. Hanzelová, "Z browser — a tool for visualization of Z specifications," in *Proceedings of ZUM'95 — 9th International Conference of Z Users*, J. Bowen and M. Hinchey, Eds. Springer-Verlag, 1995, pp. 510–523.

[138] A. Mili, N. Boudrigua, and F. Elloumi, "On the lattice of specifications: Applications to a specification methodology," *Formal Aspects of Computing*, vol. 4, no. 6, pp. 544–571, December 1992.

[139] S. Miller, T. Vitale, and M. Guy, "The Miranda programming language," December 1997. [Online]. Available: http://medialab.freaknet.org/˜martin/libri/Miranda/Description.html

[140] "Electric XML," The Mind Electric, June 2003. [Online]. Available: http://www.themindelectric.com/exml/

[141] "remote control of unix mozilla," The Mozilla Organization, January 2003. [Online]. Available: http://www.mozilla.org/unix/remote.html

[142] NASA, *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems*. Washington, DC: National Aeronautics and Space Administration, May 1997, vol. 2, NASA-GB-001-97.

[143] NASA, *Formal Methods Specification and Verification Guidebook for the Software and Computer Systems*. Washington, DC: National Aeronautics and Space Administration, December 1998, vol. 1, NASA/TP-98-208193.

[144] "NaviCon," NaviCon AG, May 2003. [Online]. Available: http://www.navicon.de

[145] "ResearchIndex terms of service," NEC Research Institute, 2001. [Online]. Available: http://citeseer.nj.nec.com/terms.html

[146] "CiteSeer: The NEC research institute scientific literature digital library," NEC Research Institute, 2002. [Online]. Available: http://citeseer.nj.nec.com

[147] "Z browser," ORA Canada, August 1999. [Online]. Available: http://www.ora.on.ca/z-eves/zbrowser.html

[148] "Z browser plug-in," ORA Canada, August 1999. [Online]. Available:

http://www.ora.on.ca/z-eves/zbplugin.html

[149] "ORA canada: Z/EVES," ORA Canada, July 2001. [Online]. Available: http://www.ora.on.ca/z-eves/welcome.html

[150] "Welcome to the programming research group," Oxford University Computing Laboratory, November 2002. [Online]. Available: http://web.comlab.ox.ac.uk/oucl/about/prg/

[151] "The Perl directory at perl.org," The Perl Foundation, 2003. [Online]. Available: http://www.perl.org

[152] J. Peterson and O. Chitil, "The Haskell home page," July 2003. [Online]. Available: http://www.haskell.org

[153] A. Plüschke, "Programs for formal concept analysis," April 2002. [Online]. Available: http://www.mathematik.tu-darmstadt.de/~plueschke/fcatools/programs.html

[154] R. Pressman, *Software Engineering: a practitioner's approach*, 3rd ed. Singapore: McGraw-Hill, 1992.

[155] D. Richards and K. Boettger, "Assisting decision making in requirements reconciliation," in *Seventh International Conference on Computer Supported Cooperative Work in Design (CSCWD 2002)*, Rio de Jainero, September 2002.

[156] D. Richards and K. Boettger, "Representing requirements in natural language as concept lattices," in *22nd Annual International Conference of the British Computer Society's Specialist Group on Artificial Intelligence (SGES), (ES2002)*, Cambridge, December 2002.

[157] D. Richards, K. Boettger, and O. Aguilera, "A controlled language to assist conversion of use case descriptions into concept lattices," in *Proceedings of 15th Australian Joint Conference on Artificial Intelligence*, 2002.

[158] D. Richards, K. Boettger, and A. Fure, "RECOCASE-tool: A CASE tool for RECOnciling requirements viewpoints," in *Proceedings of the 7th Australian Workshop on Requirements Engineering, AWRE'2002*, 2002.

[159] D. Richards, K. Boettger, and A. Fure, "Using RECOCASE to compare use cases

from multiple viewpoints," in *Proceedings of the 13th Australasian Conference on Information Systems ACIS 2002*, Melbourne, December 2002.

[160] P. Robinson, "Qu-Prolog home page," November 2003. [Online]. Available: http://www.itee.uq.edu.au/~pjr/HomePages/QuPrologHome.html

[161] T. Rock and R. Wille, "Ein TOSCANA—erkundungsystem zur literatursuche," in *Begriffliche Wissensverabeitung: Merthoden und Anwendungen*, G. Stumme and R. Wille, Eds. Berlin-Heidelberg: Springer-Verlag, 2000, pp. 239–253.

[162] P. Rook, "Controlling software projects," *Software Engineering Journal*, vol. 1, no. 1, pp. 7–16, January 1996.

[163] W. W. Royce, "Managing the development of large software systems," in *Tutorial: Software Engineering Project Management*, R. Thayer, Ed. Washington: IEEE Computer Society, 1987, pp. 118–127, originally published in Proceedings of WESCON'97.

[164] A. Ryman, "Formal methods and literate programming," in *Proceedings of the Third IBM Software Engineering ITL*, June 1993.

[165] M. Saaltink, "The Z/EVES system," in *ZUM'97: The Z Formal Specification Notation*, ser. LNCS 1212. Springer-Verlag, 1997, pp. 72–85.

[166] H. Sahraoui, W. Melo, H. Lounis, and F. Dumont, "Applying concept formation methods to object identification in procedural code," in *Proceedings of International Conference on Automated Software Engineering (ASE '97)*. IEEE, November 1997, pp. 210–218.

[167] I. Schmitt and S. Conrad, "Restructuring object-oriented database schemata by concept analysis," in *Fundamentals of Information Systems (Post-Proceedings 7th International Workshop on Foundations of Models and Languages for Data and Objects FoMLaDO'98)*, T. Polle, T. Ripke, and K.-D. Schewe, Eds. Boston: Kluwer Academic Publishers, 1999, pp. 177–185.

[168] S. Schupp, M. Krishnamoorthy, M. Zalewski, and J. Kilbride, "The "right" level of abstraction — assessing reusable software with formal concept analysis," in *Foundations and Applications of Conceptual Structures — Contributions to ICCS*

2002, G. Angelova, D. Corbett, and U. Priss, Eds. Bulgarian Academy of Sciences, 2002, pp. 74–91.

[169] R. Schwitter, "The ExtrAns-project," February 2000. [Online]. Available: http://www.ifi.unizh.ch/cl/extrans/overview.html

[170] R. Schwitter, D. Mollá, and M. Hess, "ExtrAns – answer extraction from technical documents by minimal logical forms and selective highlighting," September 1999.

[171] M. Siff and T. Reps, "Identifying modules via concept analysis," in *Proceedings of the International Conference on Software Maintenance*. IEEE Computer Society Press, 1997, pp. 170–179. [Online]. Available: http://citeseer.nj.nec.com/siff97identifying.html

[172] D. Sleator, "Link grammar," August 2000. [Online]. Available: http://bobo.link.cs.cmu.edu/link

[173] D. Sleator and D. Temperley, "Parsing english with a link grammar," in *Third International Workshop on Parsing Technologies*, 1991.

[174] G. Smith, *The Object-Z Specification Language*, ser. Advances in Formal Methods. Kluwer Academic Publishers, 1999.

[175] G. Snelting, "Reengineering of configurations based on mathematical concept analysis," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 2, pp. 146–189, April 1996.

[176] G. Snelting, "Concept analysis — a new framework for program understanding," in *SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, Montreal, Canada, June 1998, pp. 1–10.

[177] G. Snelting, "Software reengineering based on concept lattices," in *Proceedings 4th European Conference on Software Maintenance and Reengineeering*. IEEE, 2000, pp. 3–12. [Online]. Available: http://citeseer.nj.nec.com/snelting00software.html

[178] G. Snelting and F. Tip, "Reengineering class hierarchies using concept analysis," IBM T.J. Watson Research Center, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA, Tech. Rep. RC 21164(94592)24APR97, 1997. [Online]. Available: http://citeseer.nj.nec.com/snelting98reengineering.html

[179] G. Snelting and F. Tip, “Reengineering class hierarchies using concept analysis,” in *Proceedings of ACMSIGSOFT Symposium on the Foundations of Software Engineering*, November 1998, pp. 99–110.

[180] G. Snelting and F. Tip, “Understanding class hierarchies using concept analysis,” *ACM Transactions on Programming Languages and Systems*, pp. 540–582, May 2000.

[181] N. Spangenberg, “The conceptual structure of countertransference associations: an examination of diagnostic associations through an analysis of their semantic features,” in *Psychoanalytic research by means of formal concept analysis*, ser. Special des Sigmund-Freud-Instituts. Münster: Springer-Verlag, 1999.

[182] N. Spangenberg and K. E. Wolff, “Comparison between biplot analysis and formal concept analysis of repertory grids,” in *Classification, data analysis, and knowledge organization*. Berlin-Heidelberg: Springer, 1991, pp. 104–112.

[183] J. Spivey, “An introduction to Z and formal specifications,” *Software Engineering Journal*, vol. 4, no. 1, pp. 40–50, January 1989.

[184] J. Spivey, *The Z notation : a reference manual*. Prentice-Hall International, 1989.

[185] J. Spivey, *A guide to the* `zed` *style option*, Oxford University Computing Laboratory, December 1990. [Online]. Available: http://www.zuser.org/pub/zguide.ps.Z

[186] J. Spivey, *The fuzz Manual*, 2nd ed., The Spivey Partnership, Oxford, England, December 1992.

[187] M. Spivey, “The Fuzz type-checker,” September 1999. [Online]. Available: http://web.comlab.ox.ac.uk/oucl/software/fuzz.html

[188] S. Stepney, “Z and HTML,” April 2003. [Online]. Available: http://www-users.cs.york.ac.uk/~susan/abs/zhtml.htm

[189] M. Streckenbach and G. Snelting, “Understanding class hierarchies with KABA,” in *Workshop on Object-Oriented Reengineering — WOOR’99*, Toulouse, France, September 1999. [Online]. Available: http://www.infosum.fmi.uni-passau.de/st/papers/woor-99/

[190] G. Stumme, “Attribute exploration with background implications and exceptions,”

in *Data Analysis and Information Systems: Statistical and Conceptual approaches, Proceedings of GfKl'95. Studies in Classification, Data Analysis, and Knowledge Organization 7*, H. Bock and W. Polasek, Eds. Heidelberg: Springer, 1996, pp. 457–469.

[191] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal, "Computing iceberg concept lattices with TITANIC," *Data Knowledge Engineering*, vol. 42, no. 2, pp. 189–222, 2002. [Online]. Available: http://citeseer.ist.psu.edu/article/stumme02computing.html

[192] J. Sun, "Discussion of XML schema/DTD from jing sun (singapore)," January 2002. [Online]. Available: http://web.comlab.ox.ac.uk/oucl/work/andrew.martin/CZT/jing-sun/

[193] J. Sun, "Tools and verification techniques for integrated formal methods," Ph.D. dissertation, Department of Computer Science, National University of Singapore, 2003, draft.

[194] J. Sun, J. S. Dong, J. Liu, and H. Wang, "A formal object approach to the design of ZML," *Annals of Software Engineering: An International Journal*, vol. 13, pp. 329–356, 2002. [Online]. Available: http://www.comp.nus.edu.sg/~dongjs/papers/ase02sdlw.pdf

[195] J. Sun, J. Dong, J. Lui, and H. Wang, "Object-Z web environment and projections to UML," in *WWW10 — 10th International World Wide Web Conference*. New York: ACM, 2001, pp. 725–734.

[196] J. Sun, J. Dong, J. Lui, and H. Wang, "An XML/XSL approach to visualize and animate TCOZ," in *8th Asia-Pacific Software Engineering Conference (APSEC'01)*. IEEE Press, 2001, pp. 453–460.

[197] T. Taran and O. Tkachev, "Applications of formal concept analysis in humane studies," in *Using Conceptual Structures: Contributions to ICCS 2003*, B. Ganter and A. de Moor, Eds. Shaker Verlag, 2003, pp. 271–274.

[198] "Tcl developer site," Tcl Developer Xchange, May 2003. [Online]. Available: http://www.tcl.tk/

[199] T. Tilley, “Towards an FCA based tool for visualising formal specifications,” in *Using Conceptual Structures: Contributions to ICCS 2003*, B. Ganter and A. de Moor, Eds. Shaker Verlag, 2003, pp. 227–240.

[200] T. Tilley, R. Cole, P. Becker, and P. Eklund, “A survey of formal concept analysis support for software engineering activities,” in *Proceedings of the First International Conference on Formal Concept Analysis — ICFCA’03*, G. Stumme, Ed. Springer-Verlag, 2004, to appear.

[201] T. Tilley, W. Hesse, and R. Duke, “A software modelling exercise using FCA,” in *Using Conceptual Structures: Contributions to ICCS 2003*, B. Ganter and A. de Moor, Eds. Shaker Verlag, 2003, pp. 213–226.

[202] T. Tilley, “Tool support for FCA,” in *Concept Lattices: Proceedings of the Second International Conference on Formal Concept Analysis, ICFCA 2004*, ser. LNAI 2961, P. Eklund, Ed. Berlin: Springer-Verlag, 2004, pp. 104–111.

[203] P. Tonella, “Concept analysis for module restructuring,” *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 351–363, April 2001.

[204] P. Tonella and G. Antoniol, “Object-oriented design pattern inference,” in *Proceedings of CSM 1999*, 1999, pp. 230–240.

[205] I. Toyn, “CADiZ: Home page — release 4.1,” June 2002. [Online]. Available: http://www-users.cs.york.ac.uk/˜ian/cadiz/

[206] I. Toyn and J. McDermid, “CADiZ: An architecture for Z tools and its implementation,” *Software — Practice and Experience*, vol. 25, no. 3, pp. 305–330, March 1995.

[207] I. Toyn and S. Stepney, “Characters + mark-up = Z lexis,” in *ZB2002: Second International Conference of B and Z Users*, ser. LNCS 2272, D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, Eds. Springer-Verlag, January 2002, pp. 100–119.

[208] K. Turner, Ed., *Using Formal Description Techniques (An Introduction to Estelle, LOTOS and SDL)*, ser. Wiley series in Communication and Distributed Systems. Chichester: John Wiley and Sons Ltd., 1993.

[209] “Protocol specification in Estelle at univ. of delaware,” University of

Delaware Protocol Engineering Laboratory, April 2000. [Online]. Available: http://www.eecis.udel.edu/~amer/PEL/estelle/index.html

[210] M. Utting, "CZT project," July 2003. [Online]. Available: http://czt.sourceforge.net/

[211] M. Utting, I. Toyn, J. Sun, A. Martin, J. Dong, N. Daley, and D. Currie, "ZML: XML support for standard Z," in *3rd International Conference of Z and B Users (ZB'03)*, ser. LNCS 2651. Berlin: Springer-Verlag, June 2003, pp. 437–456.

[212] P. Valtchev, "Galicia home page," February 2003. [Online]. Available: http://www.iro.umontreal.ca/~valtchev/galicia/

[213] P. Valtchev, D. Gosser, C. Roume, and M. Hacene, "Galicia: an open platform for lattices," in *Using Conceptual Structures: Contributions to ICCS 2003*, B. Ganter and A. de Moor, Eds. Shaker Verlag, 2003, pp. 241–254.

[214] P. Valtchev, R. Missaoui, and R. Godin, "A framework for incremental generation of frequent closed itemsets," in *Proceedings of the Workshop on Discrete Mathematics and Data Mining (DM&DM2002)*, Arlington, VA, 2002.

[215] P. Valtchev, R. Missaoui, R. Godin, and M. Meridji, "Generating frequent itemsets incrementally: two novel approaches based on galois lattice theory," *Journal of Experimental and Theoretical Artificial Intelligence (JETAI) : Special Issue on Concept Lattice-based theory, methods and tools for Knowledge Discovery in Databases*, vol. 14, no. 2, pp. 115–142, 2002.

[216] A. van Deursen and T. Kuipers, "Identifying objects using cluster and concept analysis," in *Proceedings of the 21st International Conference on Software Engineering, ICSE-99*. ACM, 1999, pp. 246–255.

[217] R. Vienneau, "A review of formal methods," in *Software Engineering*, M. Dorfman and R. Thayer, Eds. Computer Society Press, 1996.

[218] F. Vogt and R. Wille, "TOSCANA — a graphical tool for analyzing and exploring data," in *Proceedings of the DIMACS International Workshop on Graph Drawing (GD'94)*, ser. LNCS 894, R. Tamassia and I. Tollis, Eds. Berlin-Heidelberg: Springer-Verlag, 1995, pp. 226–233.

[219] "Extensible markup language (XML)," W3C — World Wide Web Consortium, June

2003. [Online]. Available: http://www.w3.org/XML

[220] "The extensible stylesheet language family (XSL)," W3C — World Wide Web Consortium, June 2003. [Online]. Available: http://www.w3.org/Style/XSL

[221] "W3C math home," W3C — World Wide Web Consortium, April 2003. [Online]. Available: http://www.w3.org/Math/

[222] "W3C MathML implementations page," W3C — World Wide Web Consortium, March 2003. [Online]. Available: http://www.w3.org/Math/implementations.html

[223] "W3C XML schema," W3C — World Wide Web Consortium, September 2003. [Online]. Available: http://www.w3.org/XML/Schema

[224] E. Wafula and P. Swatman, "FOOM: A diagrammatic illustration of inter-object communication in Object-Z specifications," in *The 1995 Asia-Pacific Software Engineering Conference (APSEC'95)*. IEEE Computer Society Press, 1995.

[225] R. Wille, "Restructuring lattice theory: An approach based on hierarchies of concepts," *Ordered Sets*, pp. 445–470, 1982.

[226] R. Wille, "Conceptual structures of multicontexts," in *Conceptual Structures: Knowledge Representation as Interlingua. Proceedings of the 4th International Conference on Conceptual Structures*, ser. LNAI 1115. Springer-Verlag, 1996, pp. 23–39. [Online]. Available: http://citeseer.nj.nec.com/36991.html

[227] R. Wille, "Conceptual landscapes of knowledge: A pragmatic paradigm for knowledge processing," in *Proceedings of International Symposium on Knowledge Retrieval, Use, and Storage for Efficiency, KRUSE'97*, Vancouver, August 1997, pp. 2–13.

[228] J. Wing, "A study of 12 specifications of the library problem," *IEEE Software*, vol. 5, no. 4, pp. 66–76, July 1988.

[229] J. Wing, "Specifier's introduction to formal methods," *IEEE Computer*, vol. 23, no. 9, pp. 8–24, September 1990.

[230] J. Woodcock and J. Davies, *Using Z: Specification, Refinement and Proof*. Prentice Hall, 1996.

[231] J. Wordsworth, "An XML DTD for Z," October 1999.

[232] J. Wordsworth, *Software Development with Z: A Practical Approach to Formal Methods in Software Engineering*, ser. International Computer Science. Addison-Wesley, 1992.

[233] A. Yahia, L. Lakhal, J. P. Bordat, and R. Cicchetti, "An algorithmic method for building inheritance graphs in object database design," in *Proceedings of the 15th International Conference on Conceptual Modeling, ER'96*, ser. LNCS 1157, B. Thalheim, Ed. Cottbus, Germany: Springer, October 1996, pp. 422–437.

[234] S. Yevtushenko, "Sourceforge.net: Project info — concept explorer," May 2003, version 1.1. [Online]. Available: http://sourceforge.net/projects/conexp